UNIVERZITET U NOVOM PAZARU

DEPARTMAN ZA RAČUNARSKE NAUKE

DOKTORSKA DISERTACIJA

Mr. Munir T. Šabanović, dipl. inž.

Realizacija novog modela projektovanja i razvoja web zasnovanih softverskih poslovnih aplikacija

Mentor:

Prof. dr Muzafer Saračević

Novi Pazar, 2018.

UNIVERSITY OF NOVI PAZAR

DEPARTMENT OF COMPUTER SCIENCES

Mr Munir T. Šabanović, graduate eng.

DOCTORAL DISSERTATION

Implementation of the new model design and development of web-based business software applications

Supervisor:

Prof. dr Muzafer Saračević

Novi Pazar, 2018.

Disertaciju posvećujem svojim roditeljima rhm. Tahiru Šabanović i majci Fatimi.

Sadržaj

Predgovor

Oblasti koje proučava Doktorska disertacija su web programiranje i baze podataka, sa posebnim osvrtom na upravljanje, arhiviranje i pretraživanje sistema arhiviranja na internetu, koristeći različite tehnologije prenosa podataka. Osnovna suština rada je ustanoviti koji način pakovanja podataka je optimalan, uzimajući u obzir složenost koda na strani servera i klijenta, realizacijom klijent-server modela baziranog na troslojnoj arhitekturi. U disertaciji su analizirane web, web2.0, iznad web2.0[4,8] i desktop aplikacije. Radna aplikacija omogućava skladištenje podataka na strani servera, pristupajući tim podacima, koristeći različite načine formatiranja podataka, i operacija koje se primenjuju nad njima. Efekat modela je prezentiran na RIA aplikaciji. Aplikacija tipa klijent-server, bazirana na platformi Wampserver2.4-x86, realizovana u PHP/MySQL okruženju, ispunjava sve uslove troslojne veb platforme, i njen rad ne zavisi od operativnog sistema, kao ni od hardverske strukture računara. Dobijeni rezultati istraživanja se mogu podeliti u tri dela. U prvoj grupi istraživanja rezultati se odnose na različite tehnologije prenosa podataka, analizu različitih tipova aplikacija i sagledavanju zauzetosti memorije i procesora u zavisnosti od interakcije korisnika sa radnom aplikacijom. Drugu grupu čine rezultati koji predstavljaju primenu različitih tehnologija prenosa podataka. U radu je prezentirano nekoliko primera za izračunavanje različitih tehnologija prenosa podataka. U trećoj grupi su rezultati, koji opisuju primenu datog modela, na prenos podataka. Disertacija je podeljena u šest glava koje su podijeljene na poglavlja i dalje poglavlja na odeljke. U sekcijama su izneti alati, načini funkcionisanja, prednosti i mane pojedinih aplikacija.

Prvo glava sadrži pregled razvoja ideja koji su doveli do realizacije ovog doktorskog rada. Ilustrovani su osnovni podaci i otkrivene činjenice. Prezentiran je problem, predmet, ciljevi, hipoteze i tehnike istraživanja.

U drugoj glavi prezentirani su osnovni pojmovi, definicije, principi funkcionisanja web aplikacija primenom softvera otvorenog koda, gde se razmatraju različite arhitekture između servera i klijenta, opisi tehnologija koje se koriste na strani servera i klijenta, načina prenosa podataka.

U trećoj glavi je opisana radna RIA aplikacija, prezentirani su primeri primene različite vrste pakovanja podataka između klijenta i servera.

U petoj glavi dat je predlog modela projektovanja i razvoja savremenih (web i web2.0) poslovnih softverskih aplikacija.

Na kraju rada u zaključnim razmatranjima, predstavljen je model kao nova naučna informacija, u zavisnosti od zahteva korisnika, uzimajući u obzir njihove različite zahteve. Sve je bazirano na ispitivanjima sprovedenim na konkretnom modelu u radu, kao i na osnovu analiza pojedinih vrsta aplikacija koje smo predstavili u radu. Prezentirani su mogući smerovi razvoja klijent-server modela, zasnovanog na optimalnom formatiranju podataka i upotrebi sistema iznad Web 2.0. U radu su navedeni i kodovi neophodni za razumevanje rada. Disertacija se oslanja na publikovane radove u časopisima [52,53,54], ali obuhvata i rezultate do kojih smo došli u procesu istraživinja, te ih ovom prilikom prvi put prezentiramo. Predloženi model u radu, se može primeniti u različitim okolnostima, koje rešavaju slične problematike.

1. UVOD

Postoji veći broj metoda za projektovanje softvera, UML, Waterfall, agilne metode. Pošto se radi o objektnom sistemu, odabran je UML jezik za modelovanje radne aplikacije, jednako je dobar za opis desktop, web, web2.0, i iznad web2.0 aplikacije. U radu su upoređene desktop, web, web2.0 i iznad web2.0[4,8,52] aplikacije prema interfejsu[54]. Opisana je MVC arhitektura[56,57], kontroler je deo MVC platforme, MVC se koristi na klijentskoj strani, i na serverskoj kada se koristi Java server. Analizirana je prednost obrade podataka na strani servera i klijenta. I kao glavno, način komunikacije klijentske i serverske strane kod web2.0 aplikacija: XML i http servisi, json, amf, web servisi, soap, rest. Predstavljene su razlike među ovim načinima u pogledu jednostavnosti, sigurnosti, brzine, i ostalih mogućnosti.

Na kraju je data preporuka kada desktop, kada web, kada web2.0 aplikacije i kada aplikacije koje su nastale posle web2.0. Analizirana je tehnologija prenosa podataka kod web2.0, kada koristiti xml, a kada json, ili amf ili web servise i zašto, kada obrada na serverskoj, kada na klijentskoj, a kada raspodeljeno na obe strane, i zašto. I to je u stvari model razvoja aplikacija koji se nudi kao nova naučna informacija.

Za testiranje, analizu dobijenih rezultata, napravljena je klasična web radna aplikaciju koja koristi php i mysql[5,6,69,70] za crud (create, read, update, delete) operacije, sa html i css korisničkim interfejsom. Napravljena je radna RIA (web2.0) aplikacija[1] analogna klasičnoj web aplikaciji, ali sa FLEX korisničkim i nterfejsom, koja takođe izvršava CRUD operacije nad bazom podataka, sa sledećim varijacijama:

- XML REST- servisi za komunikaciju klijent–server
- JSON REST- servisi za komunikaciju klijent–server
- AMF RPC- za komunikaciju klijent–server

U svim slučajevima[11,12,16,17] testirana je brzina izvršenja crud operacija nad test bazom sa nekoliko stotina slogova.

Diskutovani su sigurnosni aspekti za svaki od ovih načina komunikacije sa bazom,
moguća preraspodela izvršenja operacija sa serverske i klijentske strane i uticaj na brzinu izvršenja i sigurnost. Na osnovu ovog treba napraviti model, preporuke kada koristiti web, kada web2, i kada koji način komunikacije u web2, u zavisnosti od potreba aplikacije.

2. METODOLOGIJA RADA

Metodološki okvir istraživanja je podeljen na više faza.

2.1 Predmet, polazišta i ciljevi istraživanja

Danas se u razvijenom svetu odvija nova revolucija- prelazak iz industrijskog u informatičko društvo. Istovremeno, mnoge zemlje se nalaze još u prvoj fazi industrijskog društva i o promenama koje pretstoje, u ovim zemljama se veoma malo zna.

Činjenica je da je u razvijenim zemljama broj zaposlenih u informatičkim delatnostima prešao 70% od ukupnog broja zaposlenih. Informatičke tehnologije danas imaju ulogu koju je parna mašina odigrala u nastanku industrijske revolucije. One omogućuju novi način proizvodnje i dovode do sasvim nove filozofije proizvodnje i odnosa prema kupcu. Ove promene su veoma brze i burne. Nove ideje i pogledi na proizvodnju i poslovanje javljaju se početkom sedamdesetih a razvijaju se posebno naglo u toku poslednjih par godina, pod uticajem naglog razvoja informatičko -komuniukacionih tehnologija.

Virtuelne korporacije, reinženjering proizvodno-poslovnih procesa, objedinjavanje svih poslovnih aktivnosti u jedinstven poslovni proces gde se posluje brzinom misli, danas su ne samo izazov, već i realnost u najuspešnijim firmama razvijenih zapadnih zemalja.

Zbog svega pobrojanog, desktop softverske aplikacije postaju zastareo i nedovoljan način korištenja informaciono-komunikacionih (ICT) tehnologija. Da bi se mogli ralizovati ciljevi i zadaci modernog poslovanja (poslovanja u dobu znanja), neophodna je primena web zasnovanih poslovnih softverskih aplikacija. Internet 2 donosi mogućnost da se kreiraju takozvane RIA (Rich internet applications), ili web2.0 aplikacije, koje iako rade koristeći internet protokole i servise, po svom grafičkom korisničkom interfejsu, i po brzini izvršenja aplikacije, ni malo ne zaostaju za desktop aplikacijama. Ove aplikacije funkcionišu uglavnom kao klijent-server aplikacije, pri čemu su baze podataka često distribuirane.

Principi projektovanja ovakvih aplikacija su najčešće preuzeti principi projektovanja desktop aplikacija. Deo koda koji predstavlja grafički korisnički interfejs (GUI) se sve češće izvršava na strani klijenta, poslovna logika i na klijentskoj i na serverskoj strani, a logika za upravljanje bazom podataka uvek na serverskoj strani. Pri tome svaki od ova tri sloja koristi "svoj" programski jezik, i ima "svoje" specifičnosti. Principi komuniciranja ova tri sloja u web zasnovanoj troslojnoj aplikaciji mogu biti veoma različiti, i često su nestandardizovani, a

bezbednosni rizici komunikacije između slojeva i funkcionisanja aplikacije u celini, nedovoljno su istraženi.

Naučni cilj ovog rada je sistematizovanje najnovijih saznanja iz oblasti razvoja web2.0 i posle web2.0 softverskih aplikacija zasnovanih na softveru otvorenog koda, analiza dostignutog nivoa primene, načina komunikacije u troslojnoj arhitekturi i sigurnosnih rizika ovakve aplikacije, te predlog modela projektovanja i razvoja web zasnovanih aplikacija prema specifikaciji ciljeva koji se aplikacijom žele postići, i potrebom za alokacijom ljudskih, materijalnihi vremenskih resursa u razvoju aplikacije.

Društveni cilj proizilazi iz naučnog. Primenom novih saznanja, naša zemlja će se uključiti u svetske tokove privrede na odgovarajući način i time stvoriti sebi šansu za dalji rast i razvoj kroz ubrzano povećanje produktivnosti.

2.2. Značaj i aktuelnost istraživanja

U industrijskoj eri je proizvodnja organizovana na postavkama čije se poreklo može trasirati unazad sve do prototipne fabrike eksera koju je Adam Smit opisao 1776. godine u ”Bogatstvu naroda”. Princip podele rada i masovne proizvodnje koji je on postavio usavršavali su kasnije Tejlor, Ford i mnogi drugi. Ovakav model proizvodnje bio je savršen početkom prošlog veka, kada je tržište bilo gladno proizvoda i dobara, kada je kupac bio sretan da može da ima auto ili neki drugi proizvod, (ma kakav on bio) i kada je kvalifikovane radne snage bilo nedovoljano za narastajuće potrebe. Radnici su bili priučeni, neobrazovani i podela rada je bila način da se oni uposle. Seckanje proizvodnih procesa u cilju podele rada, prouzrokovalo je međutim potrebu za snažnim upravljačko-nadzornim aparatom koji će objediniti deliće procesa u celinu i obezbediti njihovo nesmetano funkcionisanje. Fabrike pravljene da proizvode jedne iste proizvode, na isti način, kroz ceo svoj vek, nisu više prihvatljive. Tržišta su prezasićena robama i kupci su postali probirljivi. Danas kupac diktira proizvođaču kakav će biti proizvod, koja je prihvatljiva cena, kakvi moraju biti načini plaćanja, isporuke, servisiranja, itd. Fleksibilnost proizvodnje je imperativni zahtev današnjice.

Informatičke tehnologije se u takvoj privredi koriste u cilju poboljšanja funkionisanja firmi na postavkama serijske proizvodnje, uglavnom radi automatizacije administrativnih poslova, i veoma retko u cilju efikasnijeg upravljanja i nadzora. Bez reinženjeringa

proizvodno-poslovnih procesa, bez obezbeđivanja fleksibilne proizvodnje, sposobnosti da se proizvodi po želji i zahtevu kupca, da proizvodi budu virtuelni, skupe investicije u preduzeća čija je struktura zasnovana na premisama iz prošlog veka, su neefikasne i neisplative. Sa druge strane, informatičko komunikacione tehnologije su te koje se mogu koristiti za novu proizvodnju, fleksibilnu i sposobnu da objedini sve dobre osobine zanatske i masovne proizvodnje i eliminiše sve mane jednog i drugog načina poslovanja. Iskorištenost informatičkih tehnologija u našoj privredi danas ne dostiže ni 10%. Značaj ovog rada se ogleda u tome da ukaže na ove nove trendove i ponudi formalni model projektovanja, razvoja i primene web zasnovanih softverskih aplikacija u firmama koji će omogućiti revolucionarni skok u poslovanju i proizvodnji.

2.3 Hipoteze istraživanja

Glavna hipoteza u ovom istraživanju je sledeća:

- Moguće je razviti novi model projektovanja i razvoja web zasnovanih softverskih poslovnih aplikacija koji će omogućiti da se sleđenjem formalne procedure, optimalnim alociranjem resursa, razviju poslovne aplikacije koje omogućuju novi način poslovanja saglasan potrebama doba znanja

Pomoćne hipoteze:

- Informatičke tehnologije još uvek u proizvodnom menadžmentu nisu našle pravo mesto, te bi primenom novih modela bitno unapredili poslovanje.
- Informatičke tehnologije se u našoj privredi koriste za marginalne poslove, a ne za upravljanje preduzećima i proizvodnjom.
- Postojeći metodi razvoja softverskih aplikacija su nedovoljno odgovarajući za razvoj web zasnovanih aplikacija
- Moguće je definisati kada se koristi koji način u komuniciranju između prezentacionog sloja, sloja poslovne logike i sloja upravljanja bazom podataka, sa ciljem optimalne alokacije resursa u razvoju softverskih poslovnih aplikacija
- Od načina komunikacije između slojeva troslojne softverske web aplikacije, zavisi i ukupna sigurnost primene web aplikacije

2.4. Teorijsko-metodološki okvir istraživanja

U ovom radu potrebno je obaviti deskriptivno istraživanje, i to dve varijante deskriptivnog istraživanja - **analizu stručne literature, i servej,** i to **Delfi** metodu kao varijantu serveja. Nakon toga, primeniće se kauzalno istraživanje.

Analiza stručne literature obaviće se u cilju prikupljanja, obrade i sistematizovanja saznanja o mogućnosti projektovanja i razvoja softverskih poslovnih aplikacija primenom softvera otvorenog koda (open source), kao i primeni informatičko-komunikacionih tehnologija koje omogućuju nove modele rada firmi. Takođe će biti analizirane i studije slučaja opisane u najnovijoj literaturi iz ove oblasti.

Primerima razvoja poslovnih aplikacija kao desktop, i zatim web, pa web2.0 zasnovanih, sa raznim varijantama komuniciranja između prezentacionog, sloja poslovne logike i sloja upravljanja bazom podataka, i analizom ponuđenih rešenja, razviće se uporedni pokazatelji na koji način svaka od ovih vrsta aplikacija zadovoljava savremena merila poslovanja saglasno poslovanju u dobu znanja, kao i kakvi su sigurnosni aspekti i rizici svakog od ovih tipova softverskih aplikacija, kakvi su zahtevi za alokacijom resursa u razvoju aplikacija, te na osnovu toga kada je potrebno koristiti koje elemente u projektovanju i razvoju softverske poslovne aplikacije.

2.5.Zadaci istraživanja

Da bi se dokazale ili opovrgle hipoteze istraživanja, neophodno je realizovati postavljene zadatke istraživanja. U okviru rada potrebno je istražiti:

a. Koji su novi metodi proizvodnje i poslovanja u svetu u poslednjih nekoliko godina, koje prednosti donose, koje su pretpostavke primene, i koja su dosadašnja iskustva u njihovoj primeni.
b. Koja je uloga novih informatičko-komunikacionih tehnologija u realizaciji analiziranih metoda poslovanja i proizvodnje. Koja je nova paradigma primene informatičko-komunikacionih tehnologija u proizvodnji.
c. Koje su mogućnosti razvoja web zasnovanih softverskih poslovnih aplikacija primenom softvera otvorenog koda.
d. Koje su razlike između desktop, web i web2.0 (RIA) softverskih poslovnih aplikacija

e. Koji su načini komuniciranja između logičkih slojeva aplikacije sa klijentske i serverske strane, i njihovo međusobno poređenje sa aspekta složenosti, sigurnosti aplikacije i zahtevanih razvojnih resursa.
f. Koji su sigurnosni aspekti web i web2.0 poslovnih softverskih aplikacija
g. Da li je moguće razviti novi model projektovanja i razvoja web i web2.0 poslovnih softverskih aplikacija

Eksperimentalno istraživanje

Eksperimentalno istraživanje obuhvata:

- Projektovanje i razvoj desktop test poslovne softverske aplikacije i odgovarajućih web i web2.0 aplikacija, na primeru aplikacije za magacinsko poslovanje kao uobičajenog i najčešće zastupljenog segmenta poslovanja.
- Poređenje ova tri tipa aplikacije sa aspekta načina projektovanja i razvoja korisničkog interfejsa, poslovne logike i sistema za upravljanje bazom podataka
- Poređenje različitih načina komuniciranja između prezentacijskog, sloja poslovne logike i sistema za upravljanje bazama podataka
- Poređenje sigurnosnih apsekata kod ova tri tipa aplikacije i sigurnosnih apsekata kod web2.0 aplikacija zavisno od načina komuniciranja između slojeva
- Poređenje brzine rada desktop, web i različitih podvarijanti web2.0 aplikacija
- Analiza složenosti i kompleksnosti razvoja, stepena zadovoljenja ciljeva savremenih merila poslovanja u dobu znanja i cost-benefit analiza za svaki od pobrojanih tipova aplikacija
- Predlog modela projektovanja i razvoja poslovnih softverskih aplikacija saglasno uslovima poslovanja u dobu znanja, a prema dobijenim rezultatima prethodno pobrojanih analiza

Merni instrumenti

Kao merni instrumenti u ovom istraživanju koristiće se upitnici konstruisani za potrebe istraživanja. Upitnici će u okviru Delfi metoda biti dati na popunjavanje ekspertima koji će testirati razvijene aplikacije.

Takođe će biti korišteni komercijalno dostupni softveri za analizu brzine izvršavanja softverskih aplikacija, za analizu sigurnosti i bezbednosti softverskih aplikacija, i za praćenje komunikacionih paketa u mreži prilikom komuniciranja između logičkih slojeva aplikacije.

3.PRINCIPI FUNKCIONISANJA WEB APLIKACIJA PRIMENOM SOFTVERA OTVORENOG KODA

3.1 Klijent server arhitektura

Osnovni elementi u ovakvoj arhitekturi su server, klijent i mreža koja povezuje ova dva elementa. Postoje dvoslojne i troslojne klijent server platforme. Radna aplikacija je realizovana u MVC arhitekturi na klijentskoj strani i na serverskoj kada se koristi Java server. U radnoj aplikaciji tri sloja su ne deljiva na klijentskoj i serverskoj strani. Naime, najzastupljenije arhitekture su:

3.1.1 Dvoslojna arhitektura – inteligentni server

U ovakvoj arhitekturi, poslovna logika i baza podataka nalaze se na serveru, dok se prezentacija nalazi na strani klijenta. Kada je potrebno izmeniti sadržaj u prezentacionom delu, nije potrebno menjati i prezentacioni deo, već se potrebni podaci šalju serveru za ažuriranje koje inicira prezentacioni sloj. Ovakva dvoslojna poslovna logika je pogodna, kada postoji veliki broj klijentskih konzola.

3.1.2 Dvoslojna arhitektura – inteligentni klijent

U ovakvoj arhitekturi kontroler se nalazi na strani klijenta koja vrši obradu podataka, tako da je server manje opterećen. U slučaju kada klijentska strani koristi Flex a serverskoa PHP jezik, kao klijentski jezik koristi se ActionScript, koji vrši kompletnu obradu podataka na strani klijenta. U ovom slučaju svaki put kada se obrađuju podaci, dovlače se podaci sa servera na klijentu, posle obrade ažurira se stanje na serveru. Dakle, u ovom slučaju neophodno je veliku količinu podataka pri ažuriranju smestiti sa strane klijenta. Ovakva dvoslojna platforma je optimalna, kada nije mnogo klijentskih aplikacija, i ne vrši se česta izmena podataka

3.1.3 Troslojna-višeslojna arhitektura

U ovakvoj arhitekturi, tri sloja su fizički razdvojena[56]. Nezavisni slojevi su prezentacija, poslovna logika i izvor podataka. Izmene na jednom sloju neće prouzrokovati promene na drugom sloju. U ovom slučaju, može se izmeniti algoritam u poslovnoj logici, ali prezentacija ostaje ista kao i baza podataka. Takođe, ako se izmeni prezentacija, to se neće reflektovati na druga dva sloja. Poslovna logika može da se raspodeli na serversku i klijentsku logiku. Serverski jezici su odgovorni za logiku na strani servera, u našem slučaju to je široko korišćeni jezik PHP, a na strani klijenta koristi se klijentski jezik ActionScript. Programer pri kreiranju MVC arhitekture odlučuje stepen obrade na strani klijenta i servera. U autorskoj aplikaciji klijentska strana je realizovana u MVC strukturi[56,57].

3.1.4 Internet arhitektura

Arhitektura ovakve veze između računara, podrazumeva da se baza podataka nalazi na server, poslovna logika je raspodeljena između serverske i klijentske strane, a prezentacioni deo je na web klijentu. S obzirom da je poslovna logika raspodeljena, angažovana je serverska i klijentska obrada podataka, i ako se u ovakvoj arhitekturi akcenat stavlja na poslovnu logiku klijenta. Projektant informacionih sistema, pre nego što počne bilo šta da radi, mora izabrati odgovarajuću arhitekturu. Ako koristi usluge hosting provajdera, onda provajder kreira datu arhitekturu, a programer koristi servise provajdera. Arhitektura za dati domen problema je jako bitna, i zato programer joj mora pažljivo pristupiti i za najmanji projekat.

3.1.5 Peer to Peer arhitektura

Ovde se radi o distribuiranoj arhitekturi, gde su računari rasuti, i svaki računar je ravnopravan, što znači da svaki računar može da bude server i klijent.

3.2 Troslojna arhitektura softverskih aplikacija

MVC predstavlja model, gde je korisnički zahtev, procesuiranje zahteva, i grafički odziv na zahtev, razdvojen. Ovaj model sadrži tri tipa objekata, gde je svaki specijalizovan, za određenu vrstu posla[56].

3.3 Klijentske tehnologije

Na strani klijenta zavisno od namene aplikacije koriste se različite tehnologije, HTML i CSS[5] za web1.0 aplikacije. U RIA radnoj aplikaciji klijentske tehnologije su ActionScript3. Veoma često se kao klijentska tehnologija koristi JavaScript i DOM.

3.3.1 HTML i CSS

Podaci sa servera prema klijentu se serijalizuju pomoću HTML -a. HTML je skraćenica od HyperText Markup Language[45], što znači da se sadržaj stranica pakuje pomoću tagova. Za razliku od html-a koji služi za dizajn stranice, CSS je zadužen za izgled stranice. Pomoću naredbi u CSS-u se definiše izgled elemenata koji su kreirani naredbama HTML -a.

3.3.2 Javascript

Domen, kada počinje da deluje klijentska skripta je od trenutka kada se učita stranica sa servera pa dok se ne zatvori stranica[5]. Javascript ima veoma sličnu sintaksu programskog jezika Java, podržava neke Javine izraze, ima istu kontrolu toka, kompaktan je i objektno orijentisan skriptni jezik za razvoj Internet aplikacije.

3.3.3 DOM

Svi dokumenti koji se nalaze na strani, jesu elementi (objekti) tog dokumenta i spadaju u DOM model (Document Object Model)[66]. Svaki HTML document kada se učita u browser formira hijerarhijski objektni model. Window bude na vrhu, document ispod, a svi elementi budu unutar document objekta.

3.3.4 Ajax

Termin Ajax izvorno predstavlja skraćenicu za Asinhroni JavaScript i XML. Ajax služi kao client-side tehnologija za prenos podataka između Internet pretraživača i servera asinhrono u pozadini, da bi se obezbedio bolji odziv. Jezik koji rukuje AJAX-om je JavaScript[5].

3.4 Serverske tehnologije

Serverske tehnologije se koriste za obradu na strani servera. U radnu aplikaciju koristi se PHP serverski jezik za Wampserver2.4-x86 server. Kada se koristi Java server, onda se obrada vrši u Java jeziku.

3.4.1 PHP

Najčešće korišćeni skriptni jezik na strani servera je PHP. PHP se zove skriptni jezik jer se piše u vidu skripti i namenski je napravljen za upotrebu na webu, može se pisati u posebne fajlove, a može se umetnuti unutar HTML-a. PHP procesor na web serveru interpretira PHP kod, i na izlazu emituje HTML ili drugu vrstu podataka koju na strani klijenta Internet pretraživač može da razume[58]. Kopija HTML stranice se direktno šalje sa servera na klijentski računar, dok se PHP kod ne šalje direktno, već se pre toga prevede u oblik koji će umeti da interpretira klijentski računar.

3.4.2 MySQL

MySQL je sistem za upravljanje relacionim bazama podataka otvorenog koda. Ovaj sistem omogućava višekorisnički rad sa velikim brojem baza podataka istovremeno. Zbog svoje pouzdanosti, stabilnosti i brzine je čest izbor kod veb aplikacija koje rade sa bazama podataka. MySQL je najpopularniji system za upravljanje relacionim bazama podataka otvorenog koda[5,70]. Neki od primera upotrebe MySQL su Google, Facebook, Twitter. Glavni konkurenti su mu Microsoft SQL Server i Oracle. MySQL radi na većini serverskih operativnih sistema.

3.5 RIA

RIA aplikacije imaju bogatiji korisnički grafički interfejs, rasprostranjena je primena u praksi[1,3,4]. Radna aplikacija je RIA platforma, sa različitim tehnologijama prenosa podataka između servera i klijenta.

3.5.1 Adobe Flex platforma

Web korisnički interfejs zaostajao je tokom razvoja za desktop aplikacijama. Komponente, poput rešetkaste strukture, liste, boksova, menija i slično koje su bile sastavni deo desktop aplikacija, na početku nisu bile dostupne na web-u. Korisnički interfejs kod web aplikacija je bio ograničen propusnim opsegom, te su zbog toga morali biti vrlo jednostavni. Adobe Flex je postojeća tehnika koja treba da obezbedi nešto što se zove RIA aplikacija, odnosno, bogata internet aplikacija. Praktično gledano, Flex daje mogućnost da se napravi grafički korisnički intefejs, nešto što HTML nema[1].

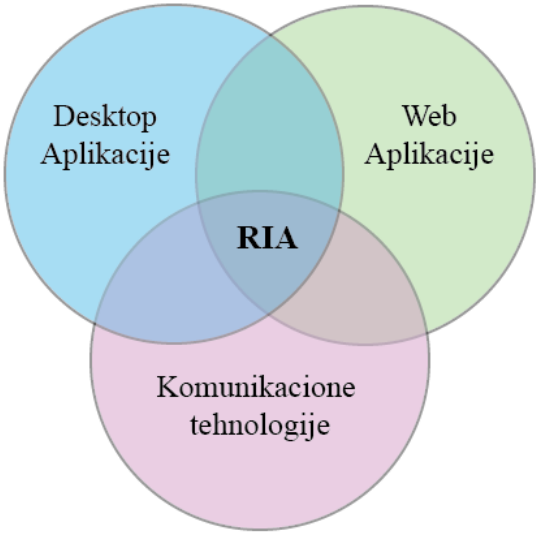
Sada postoji i četvrta mogućnost za kreiranje RIA aplikacija koristeći HTML5 i CSS. Međutim ozbiljan problem kod aplikacija koje su kreirane pomoću HTML5 i CSS jeste, da nije identičan prikaz u svim internet pretraživačima[5]. Drugi problem je sa komunikacijom. Kod HTML5 mora se koristiti JavaScript kod, za komunikaciju, dok je komunikacija u Flex mnogo bolja. Što se tiče okruženja kod HTML5, radi se sve pešice a kod Flex je dosta toga automatizovano, što opet Flex čini atraktivnijim za izbor. Dakle, potrebno je dobro grafičko razvojno okruženje a to je Flex Builder, koji nije besplatan i to je problem. Biblioteka klasa SDK za Flex okruženje je besplatna.

Flex je dobar što se oslanja na Java okruženju, a ono što napravimo u Javi, radi u svim platformama, bez izmene koda. Java poseduje JVMna svakoj mašini radi instalacije i sporije se izvršava zato što se prvo od Java koda, kreira bajt kod, koji se posle izvršava, dok se C#, C++ brže izvršavaju i ako nisu u potpunosti nezavisni od platforme. Postoji problem što se mora instalirati Flesh player jer u suprotnom neće raditi aplikacija kreirana u Flex-u, ali je dobra stvar što Flex aplikacija identifikuje da nema Flesh player-a u internet pretraživaču, i na taj način se sugeriše instalacija ovog pomoćnog programa. Flesh player brine da aplikacija izgleda kako smo zamislili za bilo koji internet pretraživač i zbog toga je on dobar.

Stoga, sada se koriste Web aplikacije koje su slične desktop aplikacijama po interfejsu i broju komponenti koji se koristi. RIA aplikacije po sebi teže funkcionalnosti desktop

aplikacijama, omogućavaju slično iskustvo, mogu sadržavati multimedijske elemente (video i audio sadržaj), omogućavaju međusobnu komunikaciju korisnika[1].

RIA aplikacije objedinjuju u jednu celinu prednosti web aplikacije, napredne desktop aplikacije i komunikacione tehnologije što je grafički predstavljeno sledećom slikom.



Slika 3.5.1. Rich Internet Aplikacija

U literature se često koriste termini RIA, Web 2.0 ili period posle web2.0, za aplikacije bogate sadržajem. Dakle, RIA i Web 2.0 su sinonimi, odnose se na istovetnu stvar. RIA aplikacije se izvršavaju u potpunosti u internet pretraživaču, bez obzira na namenu aplikacije, što je u potpunosti u skladu sa konceptom Web 2.0 aplikacija.

Tehničke specifikacije Web 2.0-a se nisu promenile u odnosu na Web, zapravo, to su brzina prenosa, konekcija(ADSL, kablovska ili dajl up, bežični prenos). Web 2.0 dodaje i socijalnu komponentu, što znači da nije standardna aplikacija, već mogu da se dele neka dešavanja kroz tu aplikaciju npr. fejsbuk, porodica, politička partija. Osnovne karakteristike Web 2.0 standarda su:

- Internet kao patforma, što, znači, sve je i dalje zasnovano na internet strukturi, konekcija, kablovi, HTTP servis, web servis, dakle ono što je i ranije bio internet.
- Korisnici kreiraju sadržaj i nad njima imaju kontrolu, sadržaj pripada korisnicima. Na primer na facebook korisnik određuje sadržaj.
- Društvena komponenta. Fejsbuk korisnik određuje kako će izgledati. Formira novo društvo, socijalnu grupu.

- Unapređeni grafički interfejsi, sada postoje bogatije grafičke komponente, rešetkasta struktura, boksove, liste, menije, panele, prozore, slajdere, kombo boksove, layout kontejnere, form elemente, itd, sve što imaju i desktop aplikacije.

Naime, Flex je open source framework koji pomaže programerima da brzo naprave Flex pod svojim okriljem sadrži:

- Dva jezika, MXML i ActionScript3, jedan okvir koji ih povezuje[1].
- Bogatu biblioteku gotovih komponenti, s tim što se funkcionalnost svake komponente može promeniti, ili se napraviti nova komponenta od nule, i na taj način programmer je u stanju da funkcionalnost Flex komponenti unapredi.
- Kompajler i dibager.
- Alat komandne linije za kompajliranje i otklanjanje grešaka u Flex aplikaciji

Flex poseduje sve ulazno izlazne komponente, layout kontejnere, form elemente.
Koncept arhitekture RIA aplikacije se oslanja na sistem gde se deo obrade izvršava unutar internet pretraživača što čini efikasnijim process obrade, jer se rasterećuje server a pružaju i veće mogućnosti dizajneru aplikacije, zato što odlučuje na koji način se vrši obrada na strani klijenta. Klijentski program koji se koriste za obradu su ActionScript i JavaScript[5]. Još jedna veoma bitna osobina ove arhitekture je asinhrona komunikacija izmedu RIA aplikacije i servera što omogućava brže izvršavanje aplikacije. Kod ovakve vrste komunikacije, šalje se mala količina podataka na serveru, i menja se samo deo stranice, tako da se ne vrši ponovno učitavanje cele stranice kao što je slučaj kod klijent-server arhitekture. Zahvaljujući objektno orijentisanoj notaciji, klijent se posmatra kao skup objekata i podobjekata, gde je objekat na najvišem nivou prozor browser-a, a drugi nivo je document unutar prozora koji se učitava, kao i objekti ovog dokumenta.

3.6.1 XML i HTTP protokol

XML - sintaksa

Struktura XML dokumenta se definiše korišćenjem tagova, koji sadrže labelu i vrednost za svaki element. Jedan tag predstavlja reč ili skup karaktera u okviru znakova <>.XML predstavlja jezik za kreiranje elektronskih dokumenata. Jedan XML dokument predstavlja

hijerarhiju XML elemenata. Svaki element reprezentuje deo informacija koje su sadržane u dokumentu [21].

HTTP protokol

HTTP je aplikacijski protokol kojim se razmenjuju sve vrste fajlova na World Wide Web, bez obzira da li su oni HTML fajlovi, slike, upiti u bazu podataka, ili nešto drugo. HTTP protokol je jedan od najpopularnijih aplikacijskih protokola, gde se razmena podataka između klijenta i servera ostvaruje na asimetričan način i to po modelu zahtev-odgovor. HTTP je bezstatusni protokol, što znači da trenutni zahtev ne zna šta je urađeno u prethodnom zahtevu. Sistem HTTP ne zavisi od tipa podataka koji se prenose između severa i kljijenta.

3.6.2 JSON

JSON ili od JavaScript Object Notation je veoma jednostavan tekstualni fromat za razmenu podataka između servera i klijenta. Jednostavan je za parsiranje i nezavisan od bilo kog programskog jezika[11]. Parsiranje JSON formata se može izvršiti i pomoću ugrađene JavaScript funkcije. U odnosu na xml format brži je, kraći, nema rezervisanih reči.

3.6.3 AMF klase

AMF je PHP biblioteka koja zna kako da serijalizuje i deserijalizuje AMF protokol, i tako omogućava da se izlože PHP klase za Flex aplikacije. Pošto klijent može da poziva metode PHP klasa i ove metode vraćaju PHP objekte, ne mora se modifikovati postojeći kod za izlazni JSON ili XML. Kada se koristi XML ili JSON za remoting, potrebni su dodatni koraci u Flex-u, za obradu podataka, da bi se prikazali ili skladištili. AmfPHP se koristi u projektima koji pokrivaju širok spektar, od igrica do poslovnih aplikacija.

S obzirom da se aplikacije ne prave samo za desktop računare, već i za smart telefone, tablet računare, ovo je povećalo kompleksnost koda, da bi aplikacija nesmetano radila na različitim uređajima. AmfPHP je najbolja solucija za kreiranje dostupnih servisa na svim terminalima. Programer se može fokusirati na jedinstvene karakteristike za svoje projekte, bez obzira na komunikacije između klijenta i servera.

Web servisi

Web srvisi su pristupne tačke na internetu, koje omogućavaju pristup određenim informacijama i dokumentima.Web servisi su prihvaćeni i podržani od strane svih glavnih proizvođača softvera što govori dovoljno o njihovom značaju[27]

3.7 Poređenje načina komunikacije klijentskog i serverskog dela aplikacije

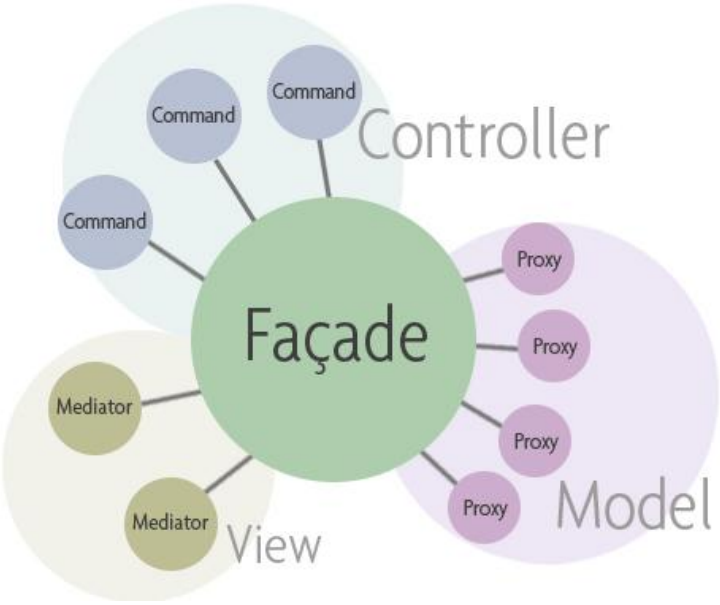
U ovom radu analizirana je klasična web aplikacija koja koristi php i mysql za crud (create, read, update, delete) operacije, sa html i css korisničkim interfejsom. Ovaj rad se bavi i analizom RIA (web2) aplikacije koja koristi php i mysql na serverskoj strani, sa FLEX korisničkim interfejsom, koja isto izvršava CRUD operacije nad bazom podataka, sa sledećim varijacijama:

1. XML REST servisi za komunikaciju klijent-server
2. JSON REST servisi za komunikaciju klijent –server
3. AMF RPC za komunikaciju klijent –server
4. XML Web servise za komunikaciju klijent –server

U modelu se rade RIA aplikacije (Rich internet aplication), koje rade na desktopu, imaju potpunu web komunikaciju sa serverima, služe u poslovne svrhe.
Ovde je najbitnije da se ispitaju različite tehnologije a to je Adobe tehnologija na klijentskoj strani i PHP free tehnologija na serverskoj strani, kako se ponašaju u primeni JSON-a, XML-a i AMF-a.U radnoj aplikaciji se analiziraju karakteristike jave sa svojim serverom, dok se klijentska strana ne menja, za tehnolgije prenosa JSON i XML. Ovde se upoređuju brzine izvršenja u php-u naspram izvršenja u Javi. Json, xml, amf su protokoli koji se koriste za komunikaciju između klijenta i servera. Između njih postoji razlika u brzini pakovanju podataka, načinu formatiranja podataka, složenosti kreiranja aplikacije, veličine fajla koji se transportuje, kao i kompleksnosti u pogledu parsiranja podataka iz jednog formata u drugi format.
RIA aplikacije su kreirane u MVC modelu. MVC model je troslojni, čine ga slojevi model, view i controller. MVC ima veliku primenu u kreiranju aplikacija, jer je pogodan za nadogradnju aplikacije, preglednija je aplikacija, znatno je brža, što je velika prednost za programera a i korisnika ovakvih aplikacija. Ova tri elementa u MVC-u povezuje komponenta

koja se zove Façade. Façade je tipa IFacade. Ako se desi neki događaj u jednom sloju, mogu ga obraditi listeneri u istom sloju ili njegova deca. Dok klase u drugom sloju nisu u poziciji da slušaju datu notifikaciju. Da bi se izbegao ovaj problem, MVC poseduje zajednički mehanizam u MVC strukturi, a to je Facada. Elementi koji egzistiraju u ovakvoj strukturi su Medijator, Proxy i Kontroler [3].

Prema tome, modeli, kontroleri i proksiji povezuju se sa jednom zajedničkom promenljivom façade, tipa Ifacade. Ova promenjiva se nalazi u svim klasama u okviru strukture MVC-a[3], i u klasi medijator i u klasi proksi i u klasi Command. Na slici 3.7.1., je grafički predstavljen veza između objekta Façade i elemenata Controller, View i Model..



Slika 3.7.1. Struktura modela, troslojni model aplikacije.

U Controleru se nalaze komande, View čine medijatori, dok model čine Proxy-ji. Sve klase u MVC strukturi su bazirane na singleton klasama, što znači da se objekat sa istim imenom ne može instancirati više od jednom. Ako se ima jedan objekat koji je identičan za sve delove MVC-a, onda je to veza direktna za njih[3]. Tako izgleda struktura MVC-a i šta ih povezuje.

Medijatori služe prvenstveno da prihvate interakciju korisnika i da mu prikažu rezultate posle obrade. A obrada rezultata obično ide u proksiju. Deo obrade može da se izvrši u medijatoru, kao i što se radi u autorskoj aplikaciji. Ovakva podela posla je uobičajena, zato što se ne može sve uraditi u proksiju. U proksiju se mogu odraditi for petlje, pripreme podataka, arhiviranje podataka u Array, kreiranje filtara, ali na kraju kada treba prikazati rezultate korisniku, onda neke posebne dorade se vrše unutar medijatora. Medijator se pokreće onog trenutka kada se registruje.

Proksi obrađuje podatke, tako što poziva server, prihvata rezultat sa servera, vrati odgovor u medijator a medijator je onda zadužen da informiše korisnika, odnosno informiše grafički interfejs da se nešto desilo kao posledica onog što je korisnik tražio. Ako se učitavaju novi podaci, promeniće se izgled tabele, znači učitaće se novi podaci. Ako se snima novi podatak, korisnik se može informisati porukom, da je dodat uspešno novi podatak, ako je operacija brisanja, kao rezultat obrade obrisaće red u rešeci i u bazi.Dakle, medijatori upravljaju korisničkim interfejsom, prihvataju informaciju od korisnika, t.j. interakciju i na kraju prikazuju korisniku rezultate njegove interakcije. Proksiji služe samo da prihvate podatke od medijatora ili od komandi, obrađuju ih i vraćaju rezultate. U RIA autorskoj aplikaciji, iz komande je pokrenut proksi.Komande služe da startuju određene delove aplikacije, ili izdaju neke određene naredbe po određenim notifikacijama, kao na primer:

```
studentProxy.getAllStudentiFlex(1);
```

Dakle, ovde se traže svi studenti. Sve komande su SimpleCommand-e, a to znači da izvršavaju samo jednu komandu, t.j. metodu. U ovoj metodi može se izvršiti više komandi, na osnovu jedne informacije, može da se pokrene 200 lančanih operacija. Ali CommandSimple je zato što reaguje samo na jednu informaciju a to je ta notifikacija.

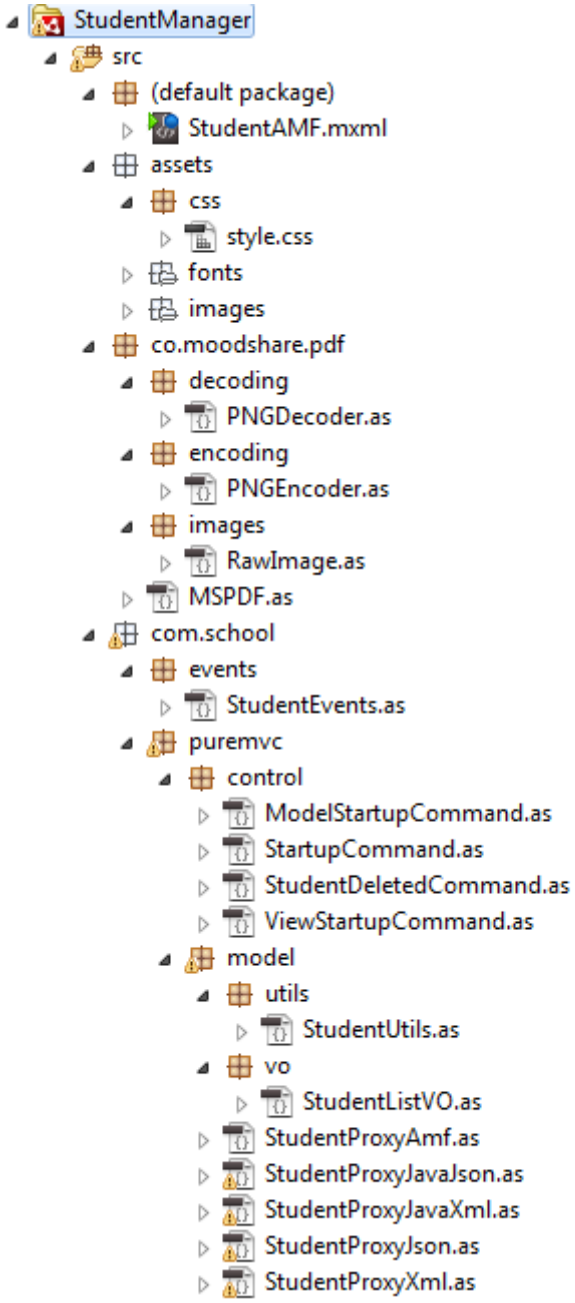
Naime, medijator prima i šalje notifikacije. Komande imaju i šalju notifikacije a proksi samo šalje notifikacije[3]. Kontroleri reaguju, oni se pale samo na notifikacije. U kontroleru pored toga što se izvrši može i da se pošalje notifikacija, znači jedno i drugo.

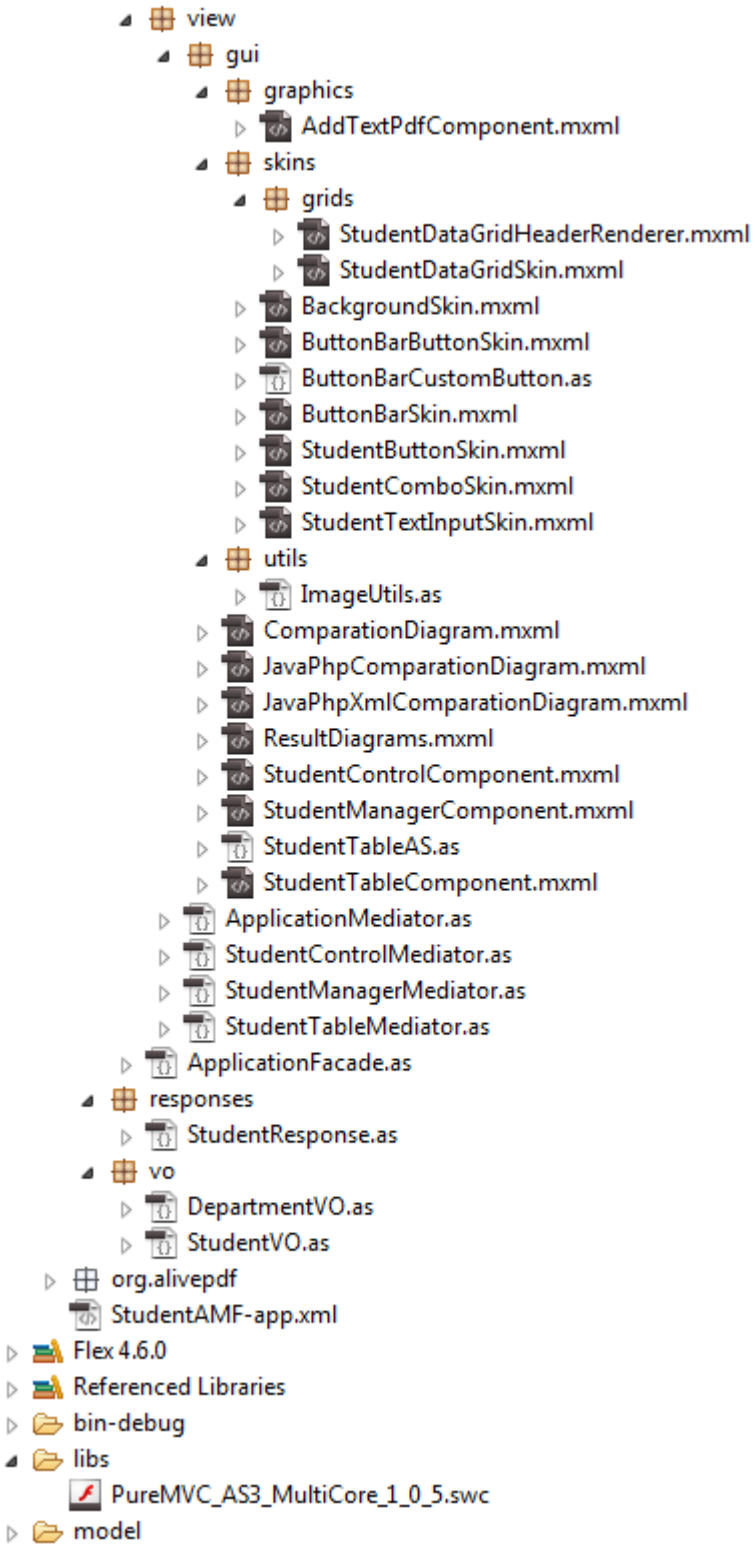
U ovom radu klase su prema logičkoj pripadnosti grupisane u paketima. Osobine klase odnosno objekta mogu biti opisne i procedurne. Klasa Stek ima za opisne osobine broj elemenata steka, kao i same elemente. Procedurne osobine pokazuju šta se sa stekom može uraditi: očitati, upisati, obrisati, itd.

Klasa (objekata) je model klase entiteta čiji su primerci objekti - modeli entiteta iz te klase entiteta. Objekti - primerci date klase imaju istu strukturu i jednako se ponašaju[52].

Paketi su logičke celine, koji mogu da sadrže druge pakete i klase. Oni se koriste radi grupisanja elemenata modela u veće jedinice. U Flexu postoje dve vrste paketa mx i spark. Spark je novija notacija i počinje da se primenjuje od flex 4. Dakle, Flex definiše dva seta komponenti: MX i Spark. MX komponente egzistirale su u prethodnim verzijama Flex-a i definisane su u mx. paketima. Spark komponente definisane su u Flex 4 i u spark.* paketima. Spark komponente koriste novu arhitekturu skinovanja i imaju druge prednosti u odnosu na MX komponente[1].

U modelu struktura paketa i klasa, data je u Package Exploreru, kao na slici3.7.2.


StudentManager
src
(default package)
StudentAMF.mxml
assets
css
style.css
fonts
images
co.moodshare.pdf
decoding
PNGDecoder.as
encoding
PNGEncoder.as
images
RawImage.as
MSPDF.as
com.school
events
StudentEvents.as
puremvc
control
ModelStartupCommand.as
StartupCommand.as
StudentDeletedCommand.as
ViewStartupCommand.as
model
utils
StudentUtils.as
vo
StudentListVO.as
StudentProxyAmf.as
StudentProxyJavaJson.as
StudentProxyJavaXml.as
StudentProxyJson.as
StudentProxyXml.as

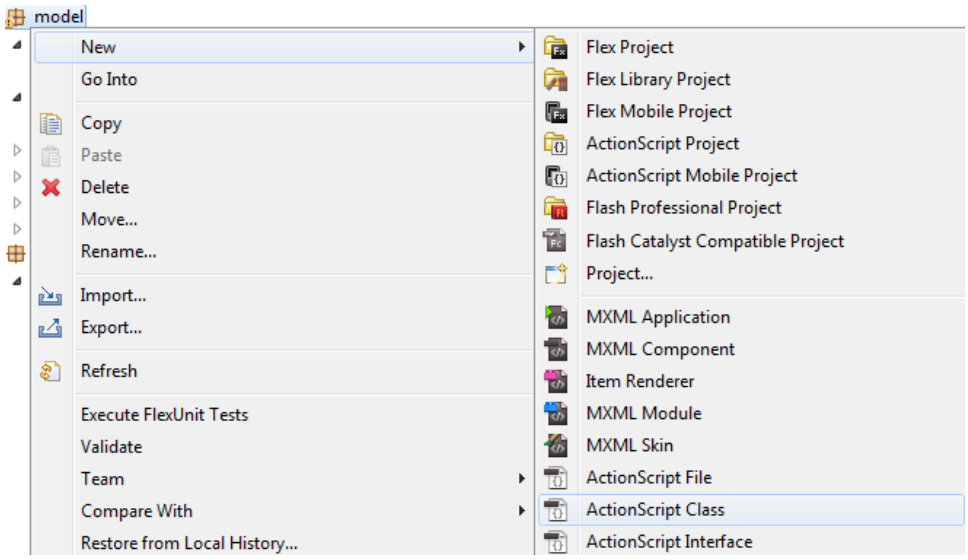


Slika 3.7.2. Struktura derveta, u Flex-u, koji prikazuje pakete i klase u aplikaciji.

Da bi se mvc struktura koristila, neophodno je postaviti PureMVC_AS3_MultiCore_1_0_5.swc fajl u folderu libs. RIA aplikacija, sadrži kontrolere u paketu control, proksije u modelu, i medijatore u view paket. Model poseduje pet proksija:

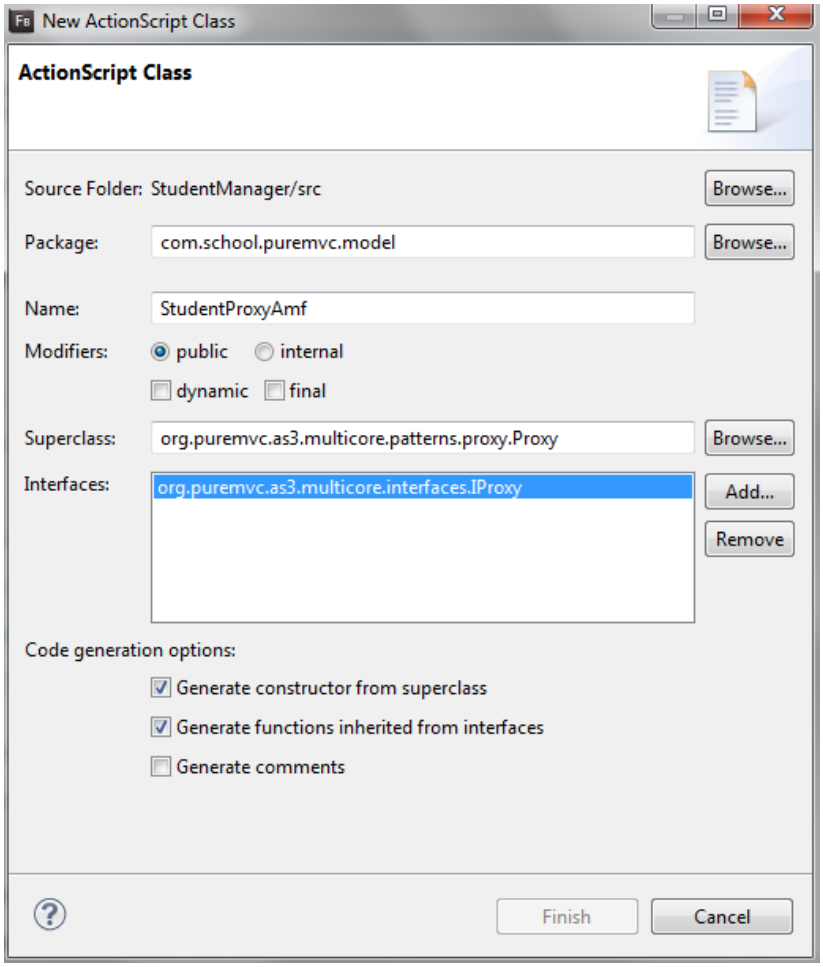
- *StudentProxyAmf.as*
- *StudentProxyJson.as*
- *StudentProxyXml.as*
- *StudentProxyJavaJson*
- *StudentProxyJavaXml*

Klasa StudentProxyAmf.as zadužena je za komunikaciju sa serverom. Ova klasa nasleđuje sadržaj ugrađene klase Proxy, i interfejs IProxy. StudentProxyAmf.as mora da implementira sve metode iz interfejsa IProxy, uz mogućnost dodavanja podataka članova i novih metoda. Naime, svaka klasa koja implementira neki interfejs mora realizovati sve metode koje se nalaze u interfejsu a može dodat nove članove. Ova klasa se kreira u paketu model, pritiskom na desni taster miša, na paket, a onda se odabere stavka ActionScript Class[1], što je pokazano na slici 3.7.3:



Slika 3.7.3. Meni za kreiranje ActionScript klase

Kada se odabere, selektovana stavka sa slike, dobija se dijalog prozor, u kome se definiše naziv klase, ime paketa u koju se smešta klasa i šta nasleđuje [3], što se vidi na slici3.7.4.



Slika 3.7.4. Dijalog za definisanje ActionScript klase StudentProxyAmf

Proxy klasa StudentProxyAmf.as sadrži pet podataka članova, jednu konstantu i dvadeset metoda. Podaci članovi su:

```
private var _myConnection:RemoteObject;
private var time:Number;
private var responseTime:Number;
public var steper:int=1;
public var average:Number = 0;
```

Objekat član _myConnection tipa klase RemoteObject, i podaci članovi time, responseTime, steper vide se u sve metode.

Statička konstanta NAME sadrži ime proksija **StudentProxyAmf.**

```
public static const NAME:String = "StudentProxyAmf";
```

Metode klase su:

- *Konstruktor metoda StudentProxyAmf*
- *onRegister()*
- *getAllStudentiFlex*
- *getAllStudentiResultHanlder*
- *getAllStudentiFaultHandler*
- *changeStudentiDataFlex*
- *changeStudentiDataFlexResultHanlder*
- *changeStudentiDataFlexFaultHandler*
- *deleteStudentiFlex*
- *deleteStudentiFlexResultHandler*
- *deleteStudentiFlexFaultHandler*
- *addStudentiFlex*
- *addStudentiFlexResultHandler*
- *addStudentiFlexFaultHandler*
- *addListeners*
- *removeListeners*
- *filterStudents*
- *filterStudentsResultHanlder*
- *getDepartments*
- *getAllDepartmentResultHanlder*

Konstruktor metoda StudentProxyAmf se poziva pri instanciranju objekta StudentProxyAmf. Ova metoda poziva konstruktor funkciju osnovne klase, prosleđuje joj jedan parameter, a to je naziv objekta, u ovom slučaju to je vrednost statičke konstante NAME.

```
public function StudentProxyAmf()
{
      super(NAME);
}
```

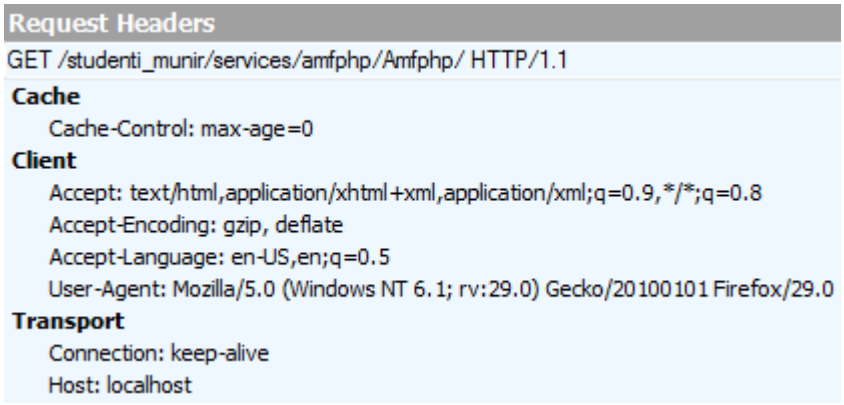
Metoda getAllStudentiFlex, ima jedan parameter start tipa int, koji govori od kojeg podatka u bazi podataka da se počne učitavanje podataka u autorskoj aplikaciji.

```
public function getAllStudentiFlex(start:int):void
{
```

```
        addListeners(getAllStudentiResultHanlder,getAllStudentiFaultHandler,"
        StudentiAPI","StudentiAPI");

        _myConnection.getAllStudenti(start,StudentListVO.requestNumber);
        time = new Date().time;//Ovo je vreme, kada se šalje zahtev bazi
podataka
}
```

U prvom redu metode getAllStudentiFlex, se poziva korisnički definisana metoda addListeners, prosleđujući joj četiri formalna argument. Prva dva argumenta su naziv metode, treći predstavlja destination a četvrti argument je source.

Pomoću objekta _myConnection, se poziva metoda getAllStudenti na serveru, prosleđujući joj podatak odakle da počne učitavanje iz baze i broj elemenata koji treba da učita. Instanca _myConnection se instancira u metodi addListeners. Header zahteva, je oblika:



Slika 3.7.5. Zaglavlje zahteva kod AMF tehnologije

Metoda addListeners uspostavlja vezu sa serverom, definišući klasu na serveru s kojom se komunicira. U metodi addListeners prosleđuju se dva parametra tipa Function.

```
private function addListeners(result:Function, fault:Function,
_destination:String, _source:String):void
{
     _myConnection = new RemoteObject;
     _myConnection.addEventListener(ResultEvent.RESULT, result);
     _myConnection.addEventListener(FaultEvent.FAULT, fault);
     _myConnection.destination = _destination;
     _myConnection.endpoint =
"http://localhost/studenti_munir/services/amfphp/Amfphp/";
    _myConnection.source = _source;
}
```

U telu ove metode instancira se objekat klase RemoteObject. RemoteObject je univerzalna klasa za povezivanje klijentskog računara sa serverom, ne zavisi od programskog okruženja na klijetskoj, ni na serverskoj strani. Objekat ove klase _myConnection, koristi više parametara pri povezivanju. Osobina endpoint definiše adresu servera ili njegovog dela, sa kojim će klijentski računar da komunicira, jer postoji na milione servera, tako da se na ovaj način locira tačno određeni server, dakle sadrži adresu servera sa kim se komunicira. Osobina resource sadrži servis ili klasu na serverskoj strani sa kojim klijent razmenjuje poruke. Polje destination ne sadrži ništa, kada klijentski računar koristi Flex a serverski PHP, već se navodi prazan string, ili se može upisati bilo šta. U primerima koji se koriste u radu postavljen je isti sadržaj kao i u source. Kada se kreiraju klase u PHP-u, naziv klase je naziv fajla[5,6], tako da se piše jedna klasa po fajlu. U dotnet-u, je drugačije, dakle kod C# u jednom fajlu može da ima više klasa, i u destination se postavlja ime klase iz tog fajla sa kojom klijentski računar komunicira, što znači, osobina destination preciznije određuje sa kim na serveru klijent trenutno komunicira.Naredbom

```
_myConnection.addEventListener(ResultEvent.RESULT,result);
```

dodeljuje se listener objektu _myConnection, u slučaju uspešnog prijema podataka sa servera. U tom trenutku aktivira se metoda result.

U delu koda _myConnection.addEventListener(FaultEvent.FAULT, fault); postavlja se osluščkivač na prijem podataka, ako se desi greška, pri zahtevu podataka sa servera, što se proverava u osobini ResultEvent. FAULT, aktivira se funkcija fault.

U izrazu _myConnection.destination=_destination; vrši se dodela vrednosti _destination polju destination, koja se dobija kao parameter metode. U Flex-u, destination je prazan string, pošto očekuje string ili bilo šta može da se napiše, dakle, nije potreban, ali se mora napisati nešto, da bi se zadovoljio princip.

```
_myConnection.endpoint="http://localhost/ordinacija_munir/amfphp/Amfphp/";
```

U endpoint se postavlja adresa servera sa kojim klijent uspostavlja vezu. Dakle, ovde se obraća fajlu index.php. U ovom fajlu je definisan način komunikacije u AMFPHP verziji, kako se serijalizuju i deserijalizuju podaci sa jednog kraja na drugi kraj. Naredbom _myConnection.source=_source; U source se postavlja veb servis(klasa) sa kojim komunicira klijent na serveru.

Metod addListeners instancira objekat klase na strani server sa kojim klijent u datom trenutku komunicira. Kada treba osloboditi osluškivače, onda se poziva metoda removeListeners koja to radi.

```
private function removeListeners(result:Function, fault:Function):void
{
      if (_myConnection){
      _myConnection.removeEventListener(ResultEvent.RESULT, result);
      _myConnection.removeEventListener(FaultEvent.FAULT, fault);
      _myConnection = null;
      }
}
```

U telu removeListeners-a, naredbom **if**(_myConnection) se proverava da li postoji instanca _myConnection klase RemoteObject. Ako postoji, onda se pristupa uklanjanju veza prema ovoj klasi. Naredbom _myConnection.removeEventListener(ResultEvent.RESULT, result); oslobađa se osluškivanje prijema podataka, i raskida se veza sa metodom, čiji naziv se nalazi u result. U delu koda _myConnection.removeEventListener(FaultEvent.FAULT, fault); raskida se veza sa metodom fault, koja se aktivira u slučaju neuspešnog prijema podataka. I na kraju ove metode referenca_myConnection se setuje na nulu, t.j. ne pokazuje ni našta.

```
_myConnection =null;
```

Skupljač smeća, neće ukloniti objekat, dok se ne uklone sve veze ka drugim elementima[1], tako da je ova metoda jako bitna. U trenutku uspešnog prijema podataka sa servera na klijentu aktivira se metoda getAllStudentiResultHanlder. Ako su podaci manji, onda se metoda izvršava brže, a ako ima puno podataka da se očita u bazi onda se kasnije izvršava. Metod ima jedan parameter tipa ResultEvent.

```
private function getAllStudentiResultHanlder(event:ResultEvent):void
 {
//event.result je array StudentiVO klasa dobijenih iz serverskog dela
//aplikacije. is zanci provera tipa objekta, as znaci posmatraj objekat
//kao neki tip.
var response:StudentResponse = event.result as StudentResponse;
if (response.result != null)
   StudentListVO.StudentiArray = new ArrayList(response.result);
      //trace(StudentListVO.StudentiArray.getItemAt(0).creationDate)
   removeListeners(getAllStudentiResultHanlder,getAllStudentiFaultHandler);
   StudentListVO.totalResults = response.totalNumber;
```

```
            //responseTime - je vreme odziva
    responseTime = new Date().time - time;//Ovo je vreme kada stigne response

    if (StudentListVO.amfDataProvider == null)
            StudentListVO.amfDataProvider = new ArrayCollection();
            var obj:Object = new Object();
            obj.callNumber = steper;
            obj.timeValue = responseTime / 1000;
            obj.average = null;
            StudentListVO.amfDataProvider.addItem(obj);
            average += obj.timeValue;
            steper++;

  if (steper == 52){
      for (var i:int=0;i<StudentListVO.amfDataProvider.length;i++){
         StudentListVO.amfDataProvider.getItemAt(i).average = average/50;
            }
            if (StudentListVO.amfAvarage == null)
                  StudentListVO.amfAvarage = new ArrayCollection();

StudentListVO.amfAvarage.addItem({data:StudentListVO.requestNumber,average:
average/ 50});
      }
      sendNotification(ApplicationFacade.STUDENT_LOADED);
}
```

Izgled učitanih podataka u AMF-u, je:

```
HTTP/1.1 200 OK
Date: Mon, 16 Jun 2014 14:51:50 GMT
Server: Apache/2.4.4 (Win32) PHP/5.4.16
X-Powered-By: PHP/5.4.16
Content-Length: 2507
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<br />
<font size='1'><table class='xdebug-error xe-fatal-error' dir='ltr' l
<tr><th align='left' bgcolor='#f57900' colspan="5"><span style='back
<tr><th align='left' bgcolor='#e9b96e' colspan='5'>Call Stack</th></
<tr><th align='center' bgcolor='#eeeeec'>#</th><th align='left' bgco
<tr><td bgcolor='#eeeeec' align='center'>1</td><td bgcolor='#eeeeec'
<tr><td bgcolor='#eeeeec' align='center'>2</td><td bgcolor='#eeeeec'
<tr><td bgcolor='#eeeeec' align='center'>3</td><td bgcolor='#eeeeec'
<tr><td bgcolor='#eeeeec' align='center'>4</td><td bgcolor='#eeeeec'
(  )</td><td title='C:\wamp\www\studenti_munir\services\amfphp\Amfph
<tr><td bgcolor='#eeeeec' align='center'>5</td><td bgcolor='#eeeeec'
</table></font>
```

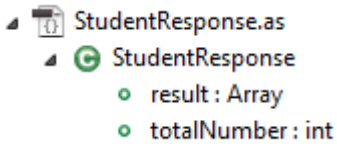
Slika 3.7.6. Struktura podataka, u AMF formatu.

Podaci u AMF format nisu čitljivi, čime se ostvaruje zaštita podataka. U telu metode getAllStudentiResultHanlder se deklariše lokalna promenljiva response tipa

StudentResponse. Klasa StudentResponse ima dva podatka člana tipa Array i int, što se vidi u kodu:

```
package com.school.responses
{
	[RemoteClass(alias="com.school.responses.StudentResponse")]
	[Bindable]
	public class StudentResponse
	{
		public var result:Array;
		public var totalNumber:int;
	}
}
```

U polje result se smeštaju podaci sa servera, dok totalNumber čuva ukupan broj podataka u bazi. Klasa StudentResponse se grafički može predstaviti:



Slika 3.7.7. Prikaz osobina i metoda za klasu StudentResponse

Promenljiva result, se inicijalizuje, pokazujući na objekat, koji treba posmatrati da je tipa klase StudentResponse.

```
var response:StudentResponse = event.result as StudentResponse;
```

event.result sadrži niz objekata tipa klase StudentiVO, dobijenih iz serverskog dela aplikacije. Operator is zanči provera tipa objekta, dok operator as znači posmatraj objekat kao neki tip.
U narednom delu koda, prvo se proverava da li postoje podaci u polju result, instance response. Ako postoje, pristupa se instanciranju objekta klase ArrayList, u njega se smeste podaci iz polja response.result, a onda se ti podaci kopiraju u StudentiArray. StudentiArray tipa ArrayList, pripada klasi StudentListVO.

```
if (response.result != null)
    StudentListVO.StudentiArray = new ArrayList(response.result);
```

Kada su prihvaćeni podaci sa servera, onda se pristupa uklanjanju listenera za metode, u slučaju uspešnog ili neuspešnog prijema podataka:

```
removeListeners(getAllStudentiResultHanlder,getAllStudentiFaultHandler);
```

Ukupan broj pristiglih podataka čuva se u statičkom podatku članu totalResults, klase StudentListVO.

```
StudentListVO.totalResults = response.totalNumber;
```

Vreme odziva u milisekundama, se beleži u promenljivu responseTime.

```
responseTime=new Date().time - time;
```

Za potrebe crtanja grafikona, instancira se generički objekat obj, tipa Object. Ovaj objekat poseduje tri podatka člana, callNumber u kojem se čuva broj pokušaja učitavanja podataka, timeValue koji arhivira vreme potrebno da se podaci učitaju u RIA aplikaciju i average u kojoj se čuva prosečno vreme svih učitanih podataka iz baze podataka.

```
var obj:Object=new Object();
    obj.callNumber=steper;
    obj.timeValue=responseTime/1000;
    obj.average = null;
```

Posle svakog učitavanja, podatak o rednom broju učitavanja, i vremenu potrebnom da se podaci učitaju, čuva se u statičku promenljivu amfDataProvider, koja je tipa ArrayList.

```
StudentListVO.amfDataProvider.addItem(obj);
```

Prosečno vreme se računa naredbom: `average += obj.timeValue;` Na kraju, se uveća broj pokušaja upisa podataka, naredbom: steper++;
U narednom bloku koda, se proverava da li je steper dostigao vrednost 52, ako jeste, onda se pristupa svakom članu niza amfDataProvider, tipa Object. Objekt sadrži tri polja. U svako polje average unutar objekta, koji predstavlja član niza upisuje se prosečna vrednost.

```
if (steper == 52){
for (var i:int=0;i<StudentListVO.amfDataProvider.length;i++){
      StudentListVO.amfDataProvider.getItemAt(i).average = average / 50;
}
      if (StudentListVO.amfAvarage == null)
            StudentListVO.amfAvarage = new ArrayCollection();
```

```
StudentListVO.amfAvarage.addItem({data:StudentListVO.requestNumber,average:
average/ 50});
}
```

Naredbom:

```
if (StudentListVO.amfAvarage == null)
      StudentListVO.amfAvarage = new ArrayCollection();
```

Proverava se da li je za niz StudentListVO.amfAvarage alocirana memorija, ako nije, alocira se. Niz StudentListVO.amfAvarage čuva podatke o odabranom broju učitanih podataka iz ComboBox-a i prosečnom vremenu za ukupan broj učitavanja. Ovaj niz se koristi za kreiranje uporednog grafika, za JSON, AMF i XML tehnologiju. Dakle, u niz amfAvarage unosi se ukupan broj unetih podataka, i prosečno vreme tokom 50 učitavanja.

```
StudentListVO.amfAvarage.addItem({data:StudentListVO.requestNumber,average:
average/ 50});
```

Posle učitavanja podataka, šalje se notifikacija, obaveštavajući da su podaci uspešno učitani.

```
sendNotification(ApplicationFacade.STUDENT_LOADED);
```

Obrada poslate notifikacije ApplicationFacade.STUDENT_LOADED, vrši se u klasi AplicationMediator.as. U slučaju neuspešnog učitavanja podataka, aktivira se metoda:

```
private function getAllStudentiFaultHandler(event:FaultEvent):void
{
      removeListeners(getAllStudentiFaultHandler,changeStudentiDataFlex);
}
```

Ova metoda poziva metodu removeListeners, koja sadtrži dva parametra tipa Function. Metoda removeListeners ukida listenere za objekat _myConnection, u slučaju uspešnog ili neuspešnog učitavanja podataka. Metoda koja se bavi editovanjem podataka u aplikaciji, je oblika:

```
public function changeStudentiDataFlex(item:StudentVO):void
{
      addListeners(changeStudentiDataFlexResultHanlder,changeStudentiDataFl
      exFaultHandler, "StudentiAPI","StudentiAPI");
```

```
        _myConnection.changeStudentiData(item);
}
```

U ovoj metodi poziva se druga metoda addListeners, koja postavlja listenere na kreirani objekat klase RemoteObject, za uspešno ili neuspešno učitavanje podataka od strane servera. Naredbom _myConnection.changeStudentiData(item); aplikacija se obraća serverskoj funkciji changeStudentiData, prosleđujući joj objekat tipa StudentVO, koji sadrži sva potrebna polja. U slučaju uspešne izmene podataka u bazi podataka, na strani servera, aktivira se metoda changeStudentiDataFlexResultHanlder.

```
private function changeStudentiDataFlexResultHanlder(event:ResultEvent):void
{
removeListeners(changeStudentiDataFlexResultHanlder,changeStudentiDataFlexF
aultHandler);
}
```

U ovoj metodi, raskidaju se veze objekta _myConnection, sa metodama za obradu pri uspešnoj izmeni podataka ili u slučaju da dođe do greške pri izmeni podataka u bazi podataka. Naredna metoda, upravlja događajem, kada aplikacija neuspešno obavi editovanje u bazu podataka.

```
private function changeStudentiDataFlexFaultHandler(event:FaultEvent):void
{
removeListeners(changeStudentiDataFlexResultHanlder,changeStudentiDataFlexF
aultHandler);
}
```

Metoda za brisanje podataka u aplikaciji ima oblik.

```
public function deleteStudentiFlex(StudentiId:int):void
{
        addListeners(deleteStudentiFlexResultHandler,deleteStudentiFlexFaultH
        andler,"StudentiAPI", "StudentiAPI");
        _myConnection.deleteStudent(StudentiId);
}
```

U slučaju uspešnog brisanja podatka iz rešetke, aktivira se metoda za obradu takvog događaja.

```
private function deleteStudentiFlexResultHandler(event:ResultEvent):void{
removeListeners(deleteStudentiFlexResultHandler,deleteStudentiFlexFaultHandler);
sendNotification(ApplicationFacade.STUDENT_DELETED,null,"student");
}
```

Naredbom

```
sendNotification(ApplicationFacade.STUDENT_DELETED,null,"student");
```

šalje se notifikacija, da je podatak obrisan. Pored naziva notifikacije, u notifikaciji se šalje prazno telo, a tip notifikacije je student. Naredna metoda, je aktivna, ako se ne obriše podatak u bazi podataka.

```
private function deleteStudentiFlexFaultHandler(event:FaultEvent):void{
removeListeners(deleteStudentiFlexResultHandler,deleteStudentiFlexFaultHandler);
}
```

Metoda koja dodaje novi podatak u aplikaciji, obraća se serveru, gde se novi podaci upisuju u bazi podataka.

```
public function addStudentiFlex(Studenti:StudentVO):void{
addListeners(addStudentiFlexResultHandler,addStudentiFlexFaultHandler,"Stud
entiAPI","StudentiAPI");
_myConnection.addNewStudent(Studenti);
}
```

Za događaj, da je uspešno dodat novi podatak, poziva se metod addStudentiFlexResultHandler, koja obrađuje ovaj događaj.

```
private function addStudentiFlexResultHandler(event:ResultEvent):void
{
removeListeners(addStudentiFlexResultHandler,addStudentiFlexFaultHandler);
      if (event.result == false){
            Alert.show("greska unosa");
      }
      else {
            trace("dodat")
      sendNotification(ApplicationFacade.STUDENT_ADDED,event.result);
      }
}
```

U ovom slučaju event.result će imati vrednost id iz poslednjeg unosa u bazu podataka. Na kraju tela metode, šalje se notifikacija, prosleđujući ime ApplicationFacade.STUDENT_ADDED, i telo notifikacije event.result. Metod koja upravlja događajem, neuspešnog dodavanja podataka u bazi je:

```
private function addStudentiFlexFaultHandler(event:FaultEvent):void
{
removeListeners(addStudentiFlexResultHandler,addStudentiFlexFaultHandler);
}
```

Aplikacija pri učitavanju ima izgled:

Slika 3.7.8. Početna stranica aplikacije-modela.

U aplikaciji podaci se mogu filtrirati, i to pomoću metode:

```
public function filterStudents(obj:Object):void
{
addListeners(filterStudentsResultHanlder,addStudentiFlexFaultHandler,"StudentiAPI","StudentiAPI");
_myConnection.filterStudents(obj.fname,obj.address,obj.email,obj.start,obj.requested,obj.department);
}
```

U prvom redu ove metode, poziva se funkcija addListeners, koja uspostavlja vezu sa određenom PHP klasom na strani servera, instancirajući objekat na serveru, koji dalje prihvata zahteve klijenta. U narednom redu, naredbom:

```
_myConnection.filterStudents(obj.fname,obj.address,obj.email,obj.start,obj.requested,obj.department);
```

poziva se metod na serveru filterStudents, prosleđujući joj vrednosti iz TextInput polja, za ime ili prezime, adresu, email, kao i vrednost od kojeg sloga u bazi da započne pretragu i koliko slogova da pretraži, kao i naziv smera. Pretraga se vrši po bilo kojem zahtevu odavde.

Naime, kada se klikne na dugme filter, aplikacija dobija izgled:

Slika 3.7.9. Prikaz dela aplikacije za pretragu u bazi podataka.

Podaci koji se dostave sa servera na klijentsku aplikaciju, obrađuju se u metodi:

```
private function filterStudentsResultHanlder(event:ResultEvent):void
{
var response:StudentResponse = event.result as StudentResponse;
if (response.result != null)
   StudentListVO.StudentiArray = new ArrayList(response.result);
   removeListeners(getAllStudentiResultHanlder,getAllStudentiFaultHandler);
   StudentListVO.totalResults = response.totalNumber;
}
```

event.result je array StudentiVO klasa dobijenih iz serverskog dela aplikacije, operator is, zanči provera tipa objekta, dok, operator as znači posmatraj objekata kao neki tip. Naredbom if (response.result != null) proverava se da li su podaci sa servera učitani na klijentskoj strani. Ako jesu, onda se alocira memorijski prostor za smeštaj podataka u niz StudentListVO.StudentiArray, prosleđujući mu kao parameter pristigle podatke. Na kraju ove metode, vrši se uklanjanje osluškivača za događaj filtriranja. U delu koda: StudentListVO.totalResults = response.totalNumber; učitava se u promenljivu totalResults, ukupan broj ovakoučitanih podataka. Ako se javi greška pri učitavanju filtriranih podataka, onda se obrada vrši metodom addStudentiFlexFaultHandler:

```
private function addStudentiFlexFaultHandler(event:FaultEvent):void
{
 removeListeners(addStudentiFlexResultHandler,addStudentiFlexFaultHandler);
      trace("stream error "+event.fault)
}
```

Podaci iz tabele department se učitavaju metodom getDepartments. Funkcija getDepartments poziva bazu podataka.

```
public function getDepartments():void {
```

```
addListeners(getAllDepartmentResultHanlder,addStudentiFlexFaultHandler,"StudentiAPI","StudentiAPI");
_myConnection.getDepartments();
}
```

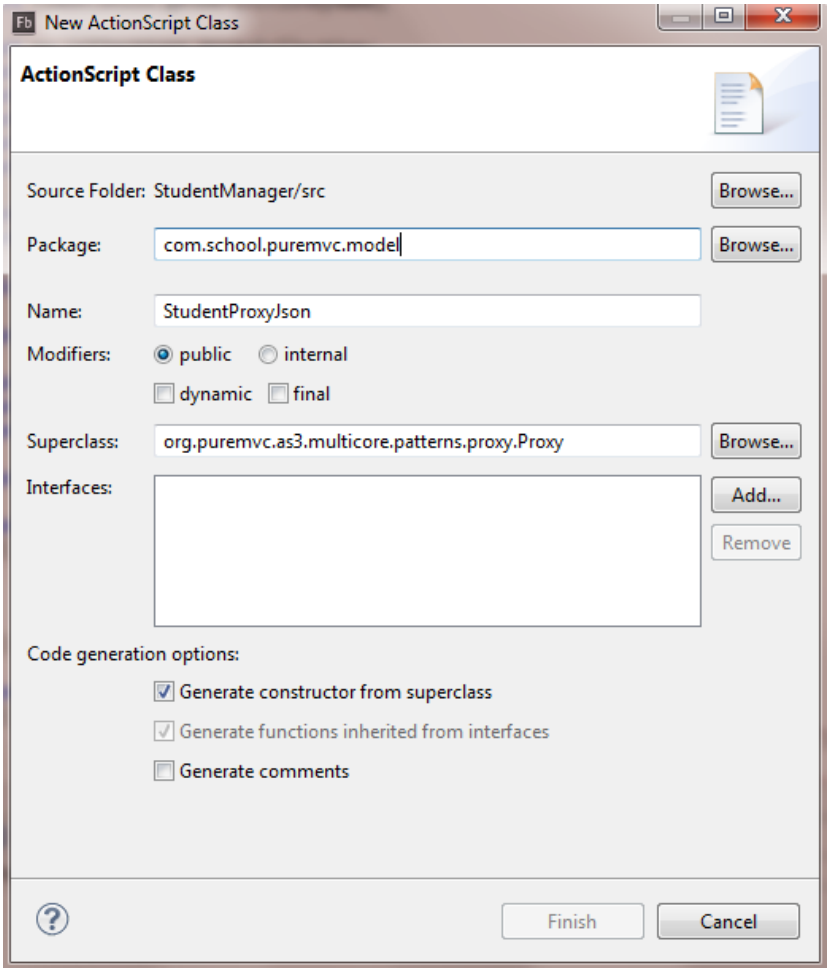
Ovde se poziva funkcija getDepartments:

```
_myConnection.getDepartments();
```

Ovo je funkcija koja se nalazi na serveru u okviru klase StudentiAPI, u bazi podataka.

3.7.1 Klasa StudentProxyJson

Ova klasa koristeći JSON format za formatiranje podataka, uspostavlja vezu sa serverom. Klasa se nalazi u paketu model, kreira se pritiskom na desni taster u prostoru Package explorer-a, a onda odabirom stavke ActionScript, dobija se dijalog prozor.



Slika 3.7.10.Dijalog za definisanje imena klase

U prozoru se definiše naziv klase, kao i paket u kome je smeštena klasa, t.j. proksi. Ovaj proksi nasleđuje klasu Proxy i implementira interfejs IProxy. Klasa StudentProxyJson.as sadrži dve statičke javne konstante NAME i serverName, dve statičke javne osobine totalResults i

requestNumber, dve javne osobine steper i average, tri private osobine loader, time i responseTime, i 14 metoda. Jedna javna metoda je nivoa klase, jedna je javna i pet privatnih metoda. Globalne osobine i konstante su:

```
private var loader:URLLoader;
 [Bindable]
public static var totalResults:int;
public static var requestNumber:int = 500;
private var time:Number;
private var responseTime:Number;
public var steper:int=1;
public var average:Number = 0;
public static const serverName:String= "http://localhost/studenti_munir
/services/json/";
public static const NAME:String = "StudentProxyJson";
```

Objekat član loader, tipa URLLOader je zadužen za uspostavljanje konekcije između RIA aplikacije i servera. Podatak član totalResults sadrži broj podataka koje treba učitati. U polje time se skladišti vreme slanja zahteva serveru, a responseTime čuva ukupno vreme od trenutka slanja zahteva, do dostavljanja podataka iz servera. U podatku članu steper, registruje se redni broj zahteva za podacima na serveru. Osobina average je zadužena za skladištenje prosečnog vremena za ukupan broj učitavanja. Statička javna konstanta serverName čuva stazu do foldera na serveru, gde su smešteni servisi zaduženi za izvršavanje zahteva, koji se uputi sa strane RIA aplikacije. Konstanta NAME sadrži naziv objekta, koji se instancira na osnovu klase koja je zadužena za komunikaciju sa serverom, a pri tome se podaci formatiraju u json format. Klasa *StudentProxyJson* sadrži metode:

- *metodu konstruktor StudentProxyJson*
- *getAllStudentiFlex*
- *on_getAllStudentiFlexHandler*
- *addStudentiFlex*
- *addStudentiFlexResultHandler*
- *deleteStudentiFlex*
- *deleteStudentiFlexResultHandler*
- *changeStudentiDataFlex*
- *changeStudentiDataFlexFaultHandler*

- *filterStudents*
- *filterStudentsResultHanlder*
- *addListeners*
- *on_getAllStudentiFlexErrorHandler*
- *removeListeners*

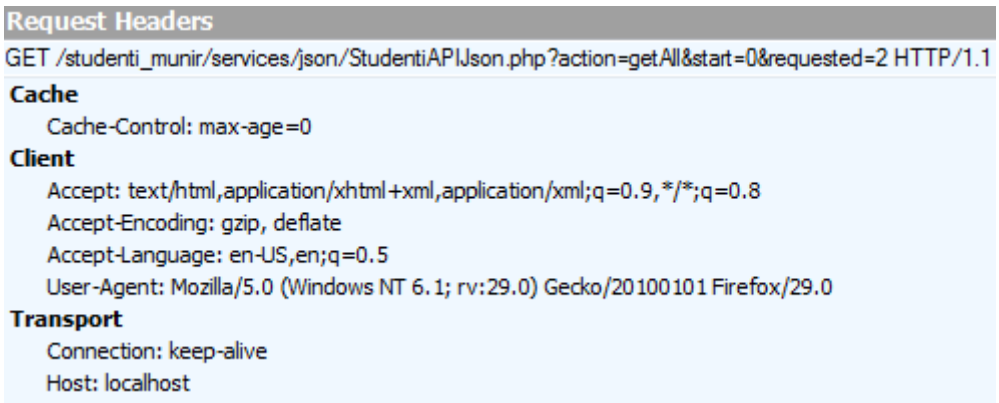
Pri instanciranju objekta klase StudentProxyJson, poziva se konstruktor metoda StudentProxyJson.

```
public function StudentProxyJson(){ super(NAME);}
```

Naredbom super(NAME); poziva se konstruktor bazne klase Proxy, i prosleđuje mu se parametar NAME. U RIA aplikaciji se učitavaju podaci metodom getAllStudentiFlex. Ova metoda poziva metodu addListeners, koja instancira objekat klase na serveru, instancira objekat koji uspostavlja vezu sa serverom.

```
public function getAllStudentiFlex(start:int):void{
  addListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHandl
  er, "StudentiAPIJson.php?action=" +StudentUtils.GET_ALL+"&start="+start
  +"&requested="+StudentListVO.requestNumber,URLRequestMethod.GET);
  time = new Date().time;
}
```

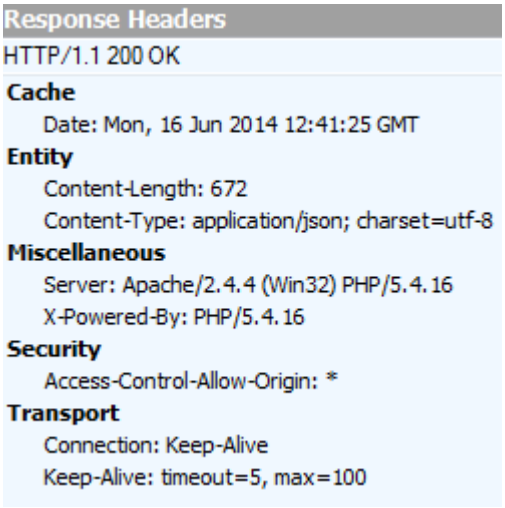
Struktura HTTP header zahteva, serveru je oblika:



Slika 3.7.11. Struktura HTTP zaglavlja u zahtevu

Metodi addListeners, prosleđuju se argumenti, dve metode koje se aktiviraju pri uspešnom ili neusmešnom učitavanju podataka u aplikaciju, naziv serverske klase, sa kojom se uspostavlja veza, kao i parametari koji definišu naziv akcije, početak učitavanja i zahtevani

broj podataka, metod koji se koristi. Podaci učitani sa servera se obrađuju metodom on_getAllStudentiFlexHandler. Naime, dobijeni respons ima oblik zaglavlja:



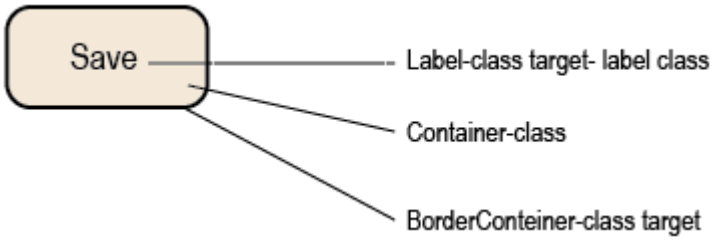
Slika 3.7.12. Struktura HTTP zaglavlja odziva

Metodon_getAllStudentiFlexHandler obrađuje uspešno učitane podatke u klijentsku aplikaciju.

```
private function on_getAllStudentiFlexHandler(event:Event):void{
      try {
            var json_var = JSON.parse(event.target.data.toString());
            var students:Array = json_var.result as Array;
            StudentListVO.StudentiArray = new ArrayList();
            for (var i:int=0;i<students.length;i++){
                  var std:StudentVO = new StudentVO();
                  std.address = students[i].address;
                  std.city = students[i].city;
                  std.country = students[i].country;
                  std.created_at = students[i].created_at;
                  std.department = students[i].depid;
                  std.email = students[i].email;
                  std.first_name = students[i].first_name;
                  std.id = students[i].id;
                  std.index_num = students[i].index_num;
                  std.last_name = students[i].last_name;
                  std.nickname = students[i].nickname;
                  std.password = students[i].region;
                  std.region = students[i].address;
                  std.updated_at = students[i].updated_at;
                  StudentListVO.StudentiArray.addItem(std);
            }
```

```
StudentListVO.totalResults = json_var.totalNumber;
removeListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHandler);
}
      catch(err:Error){
            Alert.show(err.message);
      }
      responseTime = new Date().time - time;

      if (StudentListVO.jsonDataProvider == null)
            StudentListVO.jsonDataProvider = new ArrayCollection();

      StudentListVO.jsonDataProvider.addItem(obj);
      average += obj.timeValue;
      steper++;
```

Target je elemenat nad kojim se desio događaj, a *currentTarget* je objekat koji je zadužen da sluša događaj[3]. U liniji koda ButtonClass.addEventListener("click"), ButtonClass predstavlja currentTarget



Slika 3.7.13. Button dugme

```
Butt=new Button(); Butt.mouseChildren=false,
```

Na ovaj način su diseblovane sve klase koje sadrži klasa Button i sada klasa Button je target a i currentTarget u isto vreme. Bilo gde da se klikne u njega, on će reagovati(Button) a neće tražiti da reaguju sadržane klase. U narednom delu koda se proverava da li je steper jednak 52. Vrednost promenljive steper počinje od 1 i 51 je poslednji prolaz, 50 puta se okreće, ali pošto se kasnije inkrementira steper, neće biti 50 nego 51. Što znači kada bude 51, to se podigne na 52, to je praktično 51 prolaz i tek onda primenjujemo ovaj metod:

```
if (steper == 52){
for (var i:int=0;i<StudentListVO.jsonDataProvider.length;i++){
      StudentListVO.jsonDataProvider.getItemAt(i).average = average / 50;
      }
```

Za svaki element niza jsonDataProvider, računa se i popunjava vrednost average. Ovde se prvo odredi dužina niza i uradi se iteracija za svaki element niza jsonDataProvider, znači uhvati se *getItemAt(i)* element u nizu, uzme se taj average, to je obj.average = null, koji je ubačen, i sad mu se dodeljuje vrednost average podeljen sa 50.

```
if (StudentListVO.jsonAvarage == null)
      StudentListVO.jsonAvarage = new ArrayCollection();
```

U ovom delu koda, prvo se proverava da li je jsonAvarage null, ako je null, napravi se array naredbom *StudentListVO.jsonAvarage = new ArrayCollection();* a onda u njega se ubacuje vrednosti requestNumber. Polje requestNumber je broj rezultata koji se traži, a to je 100, 200, 300...i ubacuje se average rezultat, a to je average / 50. average / 50 je ukupna prosečna vrednost. *jsonAvarage* se popunjava svaki put kada se završi jedan ciklus(50 prelaza se izvrši). Dakle jsonAvarage sadrži jednu vrednost(tipa niz) i to broj zahteva requestNumber i prosečno vreme po tom broju zahteva average / 50.

```
StudentListVO.jsonAvarage.addItem({data:StudentListVO.requestNumber,average
:average / 50});
```

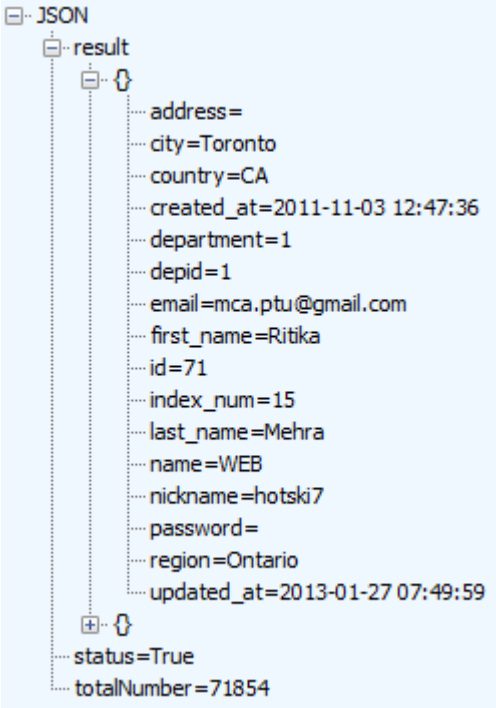
Ovde se pravi jedan array, gde je x osa broj upita requestNumber, a y osa je prosečno vreme izvršenja average / 50. Naime, average/50 je ukupno vreme(average) kroz 50 po jednom zahtevu. Jedan zahtev je kada se klikne u kombo box na npr 200 i pritisne se na dugme Make test. Po zahtevu za 100, po zahtevu za 200 ili po zahtevu za 300. jsonDataProvider uvek se setuje na nula, kada se završi 50 upita.

```
   sendNotification(ApplicationFacade.STUDENT_LOADED);
```

Ova notifikacija se registruje u metod listNotificationInterests, medijatora ApplicationMediator.as. U metod event je objekat Event klase, pojavljuje se u trenutku kad je load funkcija uspešno izvršena. event.target je objekat koji je izvršio load podataka, u našem slučaju to je (loader). Instanca loader je tipa URLLoader i ima osobinu data, koja služi za skladištenje podataka dobijenih lodovanjem dokumenta. U try bloku se vrši učitavanje podataka sa servera. U narednom redu koda se vrši parsiranje podataka koji su u obliku stringa, json formata, poslati sa servera RIA aplikaciji.

```
var json_var = JSON.parse(event.target.data.toString());
```

U data osobinu, objekta Loader, smeštaju se podaci sa servera, u obliku JSON stringa. Polje data je tipa *, što znači, data polje, može prihvatiti vrednost bilo kojeg tipa. Naredbom data.toString(), vrednost koja je stigla u data se pretvara u string. Funkcija JSON.parse, kao parametar funkcije zahteva string. String u data, sa funkcijom parse se pretvara u objekat. json_var objekat ima osobinu status t.j json_var.status. Dakle, ovde se string smešten u osobinu data, objekta loader, u našem slučaju to je event.target, pretvara u json objekat pomoću funkcije parse. Naime, parse se koristi kada se pretvara iz stringa u object. Funkcija parse očekuje string parameter, i parse prolazi kroz svaki deo koda tekstualnog fajla, bilo čega što se parsira. Kada se parsira tekst, u tom tekstu se traže poznate vrednosti i pretvaraju u objekte sa kojim se dalje radi. JSON struktura je oblika [{"id":"3", "first_name":"Kalman"}, {"id":"4", "first_name":"Dejan"}]. Prema tome, metoda parse, proverava ovakvu strukturu i pokušava da pretvori u JSON objekat a ako ne može da obradi izbacuje grešku, obavestiće da dati string nije JSON objekat, t.j. nije u strukturi JSON objekta. Ovde će od [{"id":"3"}, {"first_name":"Kalman"}] strukture, napraviti novi objekat. Object poseduje dva polja id i first_name, a vrednosti tih polja su "3" i "Kalman" onda formira drugi Object i na kraju se sve ubacuje u Array, pošto je na osnovu početne zagrade [i kranje] zaključeno da se radi o array. Ali izbaciće grešku ako postoji početna zagrada [a ne zatvara se sa]. Dakle, prvo se utvrde zagrade [....] i to je za program niz, pa proverava unutar ovih zagrada {id":"4", "first_name":"Dejan"} strukturu i to je za njega objekat, onda se vidi zarez, zarez mu govori da sledi drugi član niza i tako redom. data je tipa *. Nardbom data.ToString()- pretvara object u string. json_var je objekat koji ima strukturu result i status. Struktura učitanih podataka, t.j. responsa, kada se u aplikaciji učitavaju podaci za dva korisnika, izgleda:



Slika 3.7.14. Struktura fajla u json format

Sa slike se vidi, da je JSON tip rezultata, result je ime array. Niz result ima dva objekta, koji su oivičeni vitičastim zagradama. JSON struktura sadrži tri dela, result, status i totalNumber. U narednom bloku koda, promenljiva students se referencira na niz, koji zauzima jedno polje u objektu json_var, a onda se instancira objekat tipa ArrayList. Na ovaj objekat referencira se statička promenljiva StudentiArray klase StudentListVO.

```
var students:Array = json_var.result as Array;
      StudentListVO.StudentiArray = new ArrayList();
```

U for petlji se svi elementi iz svakog člana niza students kopiraju u polja objekta std, klase StudentVO. Na kraju se svaki svaki objekat std dodaje u listu StudentListVO.StudentiArray, na sledeći način:

```
StudentListVO.StudentiArray.addItem(std);
```

Ukupan broj elemenata se dodaje u statičo polje, naredbom:

```
StudentListVO.totalResults = json_var.totalNumber;
```

U funkciji, naredbom responseTime=**new** Date().time - time; se računa vreme odziva. U narednom delu koda se proverava da li je snabdevač podataka za grafikon, json formata

instanciran, ako nije instancira se. Dakle, Kada se prvi put učitavaju podaci sa servera, on je prazan :

```
if(StudentListVO.jsonDataProvider==null)
      StudentListVO.jsonDataProvider=new ArrayCollection();
```

Sada se kreira jedan generički, privremeni objekat obj tipa Object, koji ima tri polja. Privremeni, zato što se nalazi unutar funkcije, i živi unutar funkcije.

```
var obj:Object = new Object();
obj.callNumber = steper;
obj.timeValue = responseTime / 1000;
obj.average = null;
```

U polje obj.callNumber se skladišti broj poziva servera, dok obj.timeValue registruje vreme odziva. obj.average se čuva prosečno vreme od svih učitavanja sa servera. Privremenom objektu obj se setuje osobina callNumber na vrednost steper. Object je dinamička klasa. Dinamičkoj klasi u realno vreme ili u vreme izvršavanja, mogu se dodavati elementi klase. Generička je osnovna klasa. Object jeste generička i dinamička klasa. U svojoj definiciji je prazna potpuno, ima samo mehanizme za kreiranje objekata. Kada se kaže obj.callNumber, objekat callNumber ne postoji, već se kreira trenutno, callNumber-u se dodeljuje vrednost steper. Steper je trenutno 1, jer je pri deklaraciji, izvršena i inicijalizacija na vrednost jedan:
public var steper:int = 1;
To znači da će to biti prvi element u grafikonu. Zatim pravi se timeValue, timeValue objekat će da bude vreme odgovora, podeljeno sa hiljadu, obj.timeValue = responseTime / 1000;
Ovde se deli sa 1000 da bi se dobile sekunde, pošto je vreme u milisekundama. Objekat average je postavljen na nula, jer se average kasnije izračunava.

```
obj.callNumber = steper;
obj.timeValue = responseTime / 1000;
obj.average = null;
```

Izrazom obj.average = null; polje average se kreira i dodeljena mu je vrednost null. Vrednost koja je dodeljena polju obj.average, ovde nije bitna, jer se prepisuje nova vrednost kasnije.
Ovakav objekat obj, koji je privremeni, dodaje se u array jsonDataProvider, kao element niza.

```
StudentListVO.jsonDataProvider.addItem(obj);
```

Pošto se broj pokušaja učitavanja uveća za jedan naredbom steper++; sledi slanje notifikacije, kojom se obaveštava listener da su podaci uspešno učitani:

```
sendNotification(ApplicationFacade.STUDENT_LOADED);
```

Pomoćne metode addListener i remuveListener su zajedničke za sve metode. Metod addListener dodaje slušaoce događaja za load podataka sa servera u obliku fajla, kreira se request sa parametrima, slektuje se metod requesta (POST ili GET).

```
private function
addListeners(onResult:Function,onError:Function,url:String,method:String,
variables: URLVariables = null):void{
loader = new URLLoader();
loader.addEventListener(Event.COMPLETE, onResult);
loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
var request:URLRequest = new URLRequest(StudentProxyJson.serverName + url);
request.method = method;
      if (variables)
            request.data = variables;
            loader.load(request);
      }
```

U ovoj metodi se najpre instancira objekat loader, URLLoader klase. Zatim, sledi dodavanje eventListenera za svaki http upit posebno. Prvi dodati listener se izvršava uvek kada postoji odgovor sa servera (status 200), bez obzira na rezultat izvršenja zahteva. Drugi listener, se aktivira u slučaju neuspešne izvršene akcije (status 404, 500 i dr.).
Narednim redom, pravi se request sa parametrima (url adresa sa GET parametrima).
***var** request:URLRequest = **new** URLRequest(StudentProxyJson.serverName + url);*
A onda se selektuje POST ili GET method. Ako je POST method postoje parametri u obliku URLVariables klase:

```
if (variables)
      request.data = variables;
```

Na kraju se izvršava poziv ka url adresi, naredbom loader.load(request);
Uvek posle izvršene akcije učitavanja, editovanja, brisanja ili dodavanja vrši se ukidanje slušaoca događaja i setovanje URLLoader na vrednost null,t.j. vrši se destrukcija objekta.

```
private function removeListeners(onResult:Function,onError:Function):void{
```

```
      if (loader){
            loader.removeEventListener(Event.COMPLETE,onResult);
            loader.removeEventListener(IOErrorEvent.IO_ERROR,onError);
            loader = null;
      }
}
```

Metoda koja dodaje novi slog podataka u bazi, ima oblik:

```
public function addStudentiFlex(item:StudentVO):void{
      var vars:URLVariables = new URLVariables();
      vars.studentFlex = JSON.stringify(item);
      vars.action = StudentUtils.ADD;
addListeners(addStudentiFlexResultHandler,on_getAllStudentiFlexErrorHandler
,"StudentiAPIJson.php",URLRequestMethod.POST,vars);
}
```

U delu koda *vars.studentFlex = JSON.stringify(item);* ceo objekat item klase StudentVO pretvoren je u string, i potom se taj string šalje u bazu. Dakle, funkcija stringify pretvara objekat u string json oblika. Podaci koji se šalju serveru, i naziv akcije smešta se u generički objekat vars. S obzirom da se šalje veliki broj podataka, odabrana je POST metoda, jer GET metoda je ograničena brojem karaktera koji mogu da se smeste u upitu. Naime, HTTP request je ograničen brojem karaktera koji mogu da stanu u jedan objekat. Kod JSON-a, zadatak je da se ne šalje svaka vrednost pojedinačno, već se sve spakuje u objekat. Naime, Get metoda ne može da se koristi, zbog toga kada ceo objekat bude prebačen u string, biće mnogo velik. To zavisi od veličine objekta, ali može biti veći nego što je dozvoljeno. Iz ovog razloga se u metodi add koristi metod post. Za događaj, da se podatak smestio u bazi na serveru, obrađuje metoda addStudentiFlexResultHandler.

```
private function addStudentiFlexResultHandler(event:Event):void
{
try {
var json_var = JSON.parse(event.target.data.toString());
      if (json_var.status == true){
         sendNotification(ApplicationFacade.STUDENT_ADDED,json_var.result);
            }
      else {
                  //obaveštenje o grešci prilikom mySql upita
```

```
                    Alert.show(json_var.error);
                    }
            }
                    catch(err:Error){
                    //obaveštenje o grešci prilikom parsiranja Json file-a
                          Alert.show(err.message);
                    }
removeListeners(addStudentiFlexResultHandler,on_getAllStudentiFlexErrorHand
ler);
}
```

Posle konvertovanja stringa u objekat json_var, proverava se da li je polje status postavljeno na true. Ako jeste, šalje se notifikacija. Notifikacija se obrađuje u medijatoru StudentManagerMediator. Metoda, koja briše slog iz baze podataka je oblika:

```
public function deleteStudentiFlex(StudentiId:int):void
{
var url:String =
StudentiAPIJson.php?action="+StudentUtils.DELETE+"&studentId="+StudentiId;
addListeners(deleteStudentiFlexResultHandler,on_getAllStudentiFlexErrorHand
ler,url,URLRequestMethod.GET);
}
public function filterStudents(obj:Object):void
{
      var vars:URLVariables = new URLVariables();
      vars.name = obj.fname;
      vars.address = obj.address;
      vars.email = obj.email;
      vars.start = obj.start;
      vars.requested = obj.requested;
      vars.department = obj.department;
      vars.action = StudentUtils.FILTER_STUDENTS;
addListeners(filterStudentsResultHanlder,on_getAllStudentiFlexErrorHandler,
"StudentiAPIJson.php",URLRequestMethod.POST,vars);
}
```

Metoda filterStudents kao parameter poseduje promenljivu obj tipa Object, koji sadrži podatke iz TextInput polja, koji služe za pretragu u bazi podataka. Deo aplikacije, kojim se filtriraju podaci prikazan je na slici 3.7.9. U telu metode filterStudents deklariše se promenljiva vars tipa

URLVariables. Istovremeno sa deklaracijom, pristupa se i alociranju memorije za objekat klase URLVariables. Alociranom objektu, putem reference vars, dodeljuju se vrednosti iz objekta obj, koji je tipa Object. Objekat klase URLVariables je dinamičkog tipa, dodeljuje mu se naziv operacije, putem statičke konstante FILTER_STUDENTS. Na kraju se poziva funkcija addListeners, koja uspostavlja vezu sa serverom. Metoda addListeners kao formalne argument poseduje naziv funkcije, koja obrađuje učitane podatke sa servera, funkciju koja obrađuje grešku u slučaju neuspešne konekcije sa serverom, naziv klase na serveru, koja se instancira i odgovara na zahteve klijenta, naziv metode i podatke koji se prosleđuju serveru sa klijentske strane. Podaci koji se dostave sa servera na klijentsku aplikaciju, obrađuju se u metodi filterStudentsResultHanlder. Prva linija koda u try bloku konvertuje arhivirane podatke u polje data, u string, a potom JSON string se metodom parse transformiše u JSON objekat. JSON objekat json_vars poseduje tri polja result, totalNumber i status. Osobina result, tipa Array, sadrži podatke iz tabele u bazi podataka.

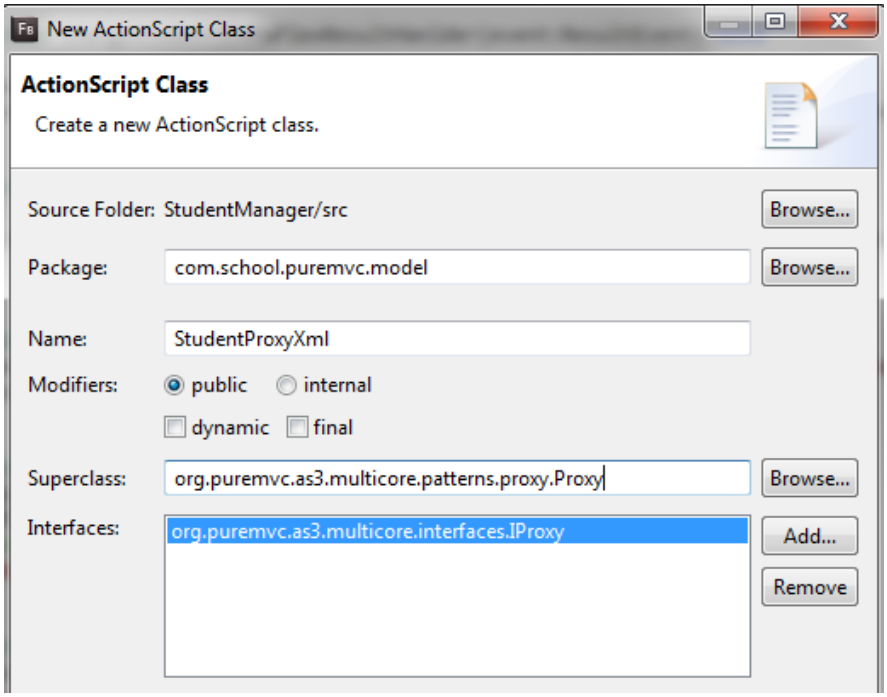
```
private function filterStudentsResultHanlder(event:Event):void
{
	try {
		var json_var = JSON.parse(event.target.data.toString());
		var students:Array = json_var.result as Array;
		StudentListVO.StudentiArray = new ArrayList();
		for (var i:int=0;i<students.length;i++){
			var std:StudentVO = new StudentVO();
			std.address = students[i].address;
			std.city = students[i].city;
			std.country = students[i].country;
			std.created_at = students[i].created_at;
			std.department = students[i].depid;
			std.email = students[i].email;
			std.first_name = students[i].first_name;
			std.id = students[i].id;
			std.index_num = students[i].index_num;
			std.last_name = students[i].last_name;
			std.nickname = students[i].nickname;
			std.password = students[i].region;
			std.region = students[i].address;
			std.updated_at = students[i].updated_at;
			StudentListVO.StudentiArray.addItem(std);
		}
```

```
StudentListVO.totalResults = json_var.totalNumber;
removeListeners(filterStudentsResultHanlder,on_getAllStudentiFlexErrorHandler);
	}
	catch(err:Error){
		Alert.show(err.message);
	}
}
```

U delu koda var students:Array = json_var.result as Array; izvlači se sadržaj niza, koji nosi naziv result, a sastavni deo je objekta json_var, i smešta se u students, tipa Array. Onda se kreira prazan statički niz StudentListVO.StudentiArray. Pomoću for petlje vrši se iteracija kroz sve elemente niza students. Pristupa se svakom delu elementa niza, i kopira se u objekat std tipa StudentVO. Polja klase StudentVO, nose isti naziv kao i naziv kolona u tabeli users na strani server. Svaki red iz tabele se smešta u jedan objekat std. Kada se napuni objekat std vrednostima, dodaje se u niz StudentListVO.StudentiArray. Na kraju se posle izvršenja svih iteracija, učitava ukupan broj učitanih podataka u statičko polje StudentListVO.totalResults.

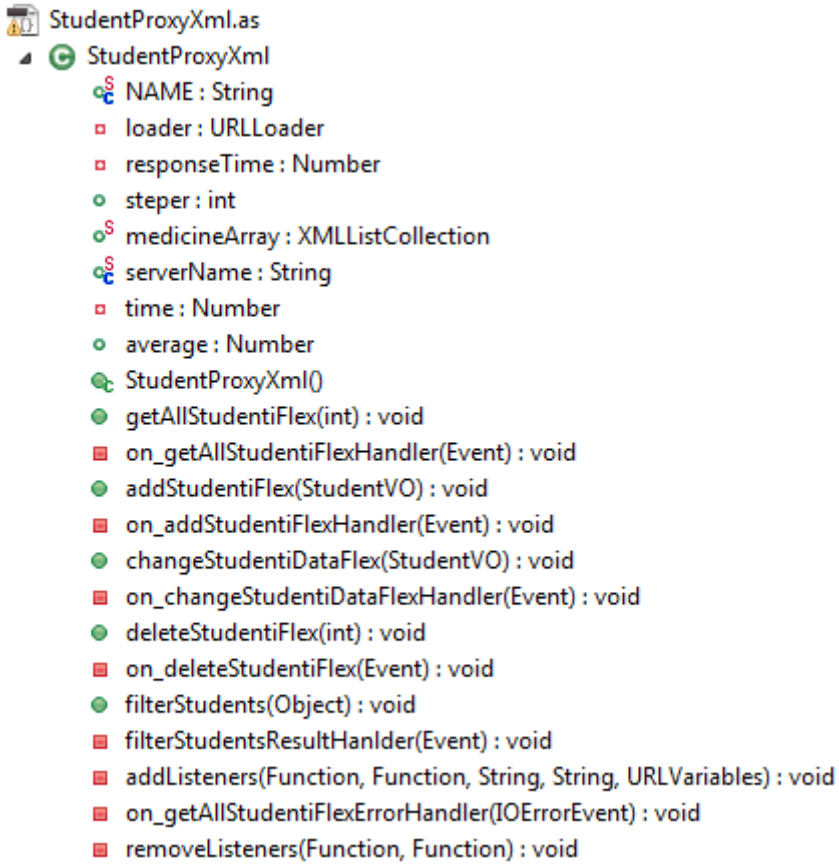
3.7.2 Klasa StudentProxyXml

Klasa *StudentProxyXml*.as koristeći XML format za pakovanje podataka, uspostavlja vezu sa serverom. Što se tiče obrade podataka, Xml je jako sličan Json-u. Xml je sporiji zbog strukture podataka. Struktura podataka Xml-a je veći dokument. Ima otvorene i zatvorene tagove a između su podaci. Same klase za obradu kod Xml-a su glomaznije nego kod Json-a. zato je malo sporije. Klasa StudentProxyXml se nalazi u paketu model, kreira se pritiskom na desni taster u prostoru Package explorer-a(slika 3.7.3)[1], a onda odabirom stavke ActionScript, dobija se dijalog prozor.



Slika 3.7.15. Dijalog prozor za kreiranje klase u Action Script-u.

U dijalog prozoru navedeno je ime klase, naziv roditeljske klase[52] i naziv interfejsa. Pritiskom na dugme Finish, kreira se klasa StudentProxyXml.as. Klasa StudentProxyXml.as u svojoj strukturi poseduje dve javne statičke konstante NAME i serverName, jednu javnu statičku osobinu medicineArray, dve javne osobinu steper i average, tri private osobine loader, responseTime i time, jednu metodu nivoa klase[52] StudentProxyXml, pet javnih metoda i osam privatnih metoda. Podaci članovi, objekti članovi i metode su prikazani na slici3.7.16.



Slika 3.7.16. Prikaz osobina i metoda za klasu StudentProxyXml

Objekat član loader, tipa URLLOader uspostavlja vezu između RIA aplikacije i servera [1]. U polje time se čuva vreme slanja zahteva serveru, a responseTime arhivira ukupno vreme od trenutka slanja zahteva, do dostavljanja podataka iz servera. U podatku članu steper, registruje se redni broj zahteva za podacima na serveru. Osobina average je zadužena za skladištenje prosečnog vremena za ukupan broj učitavanja.

Statička javna konstanta serverName čuva stazu do foldera na serveru, gde su smešteni servisi zaduženi za izvršavanje zahteva, koji se uputi sa strane RIA aplikacije. Konstanta NAME sadrži naziv objekta, koji se instancira na osnovu klase koja je zadužena za komunikaciju sa serverom, a pri tome se podaci formatiraju u xml format.

Pri kreiranju objekta klase StudentProxyXml, poziva se konstruktor metoda StudentProxyXml.

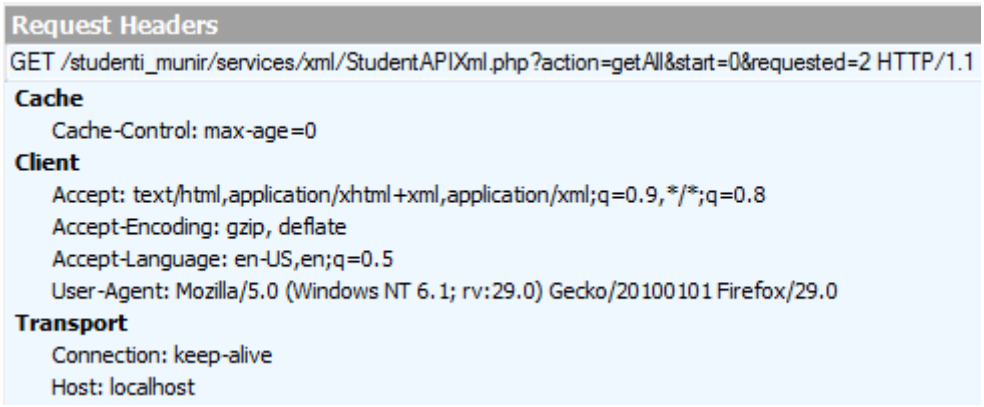
```
public function StudentProxyXml()
{
	super(NAME);
}
```

Naredbom super(NAME); poziva se konstruktor bazne klase Proxy, i prosleđuje mu se parametar NAME.

U RIA aplikaciji se učitavaju podaci metodom getAllStudentiFlex. Ova metoda uspostavlja vezu sa serverom, pozivom metod addListeners, koja instancira objekat klase na serveru.

```
public function getAllStudentiFlex(start:int):void {
addListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHandler
,"StudentAPIXml.php?action=" + StudentUtils.GET_ALL+"&start="
+start+"&requested=" +StudentListVO.requestNumber, URLRequestMethod.GET);
   time = new Date().time;
}
```

Naime, kada se učitavaju podaci u klijentskoj aplikaciji, koristeći XML tehnologiju za formatiranje podataka, poziva se metoda proksija getAllStudentiFlex, prosleđujući joj kao parametar vrednost odakle počinje učitavanje podataka u bazi. U telu metode getAllStudentiFlex, poziva se privatna metoda addListeners, koja uspostavlja konekciju sa serverom. Struktura header-a zahteva prema serveru ima oblik:



Slika 3.7.17. Zaglavlje XML zahteva hedera.

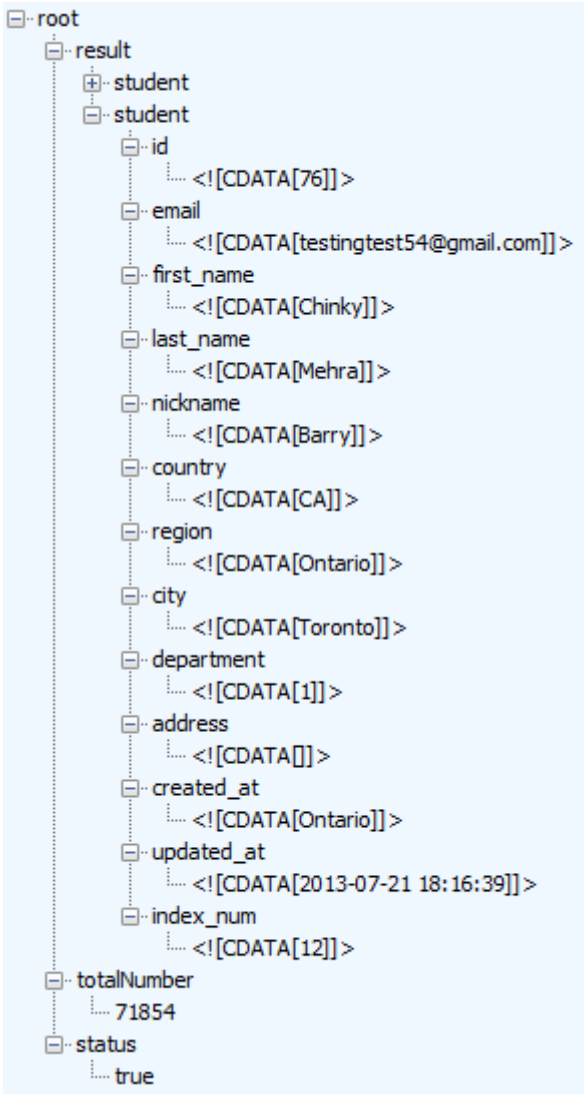
Metoda addListeners, kao formalne argument poseduje naziv funkcije, koja obrađuje primljene podatke sa strane servera, funkciju koja obrađuje slučaj, kada se podaci sa servera ne mogu učitati, naziv klase na serveru, koja će prihvatiti zahtev sa klijentske strane, šaljući joj ime zahteva, mesto početka pretrage u bazi, zahtevani broj podataka iz baze, i naziv metode kojim se podaci prenose od servera do klijenta. Ako bi klijentska aplikacija tražila dva sloga u bazi podataka, struktura traženih podataka izgleda kao na slici3.7.18. Ovakva struktura podataka se smešta u osobinu data, URLLOader objekta. Iz klijentske aplikacije šalje se upit serveru sledeće strukture:

```
http://localhost/studenti_munir/services/xml/StudentAPIXml.php?action=getAl
l&start=0&requested=2
```

Slika 3.7.18. Struktura xml fajla, struktura tagova u fajlu

Ovakva struktura se dobije u data, t.j. event.target.data. Struktura xml-a, ima root, result, i u tom result-u su deset rezultata od student-a. Postoji još dva elementa, totalNumber i status. Podaci student imaju svoje elemente, *id, email, first_name, last_name, nickname, country, region, city, department, address, created_at, updated_at i index_num,* što je prikazano na slici

3.7.18. Naredbom time = **new** Date().time; definiše se vreme kada je upućen zahtev serveru da pripremi podatke. Metod koja obrađuje podatke, koji su uspešno učitani sa strane servera na klijentu, nosi naziv getAllStudentiFlexHandler. U try bloku, kreira se XML objekat koji kopira podatke xml formata, smeštene u osobunu data, loader objekta. U data može da se smesti bilo koji tip podatka, u ovom slučaju smešta se struktura xml-a, u obliku stringa. Struktura xml-a u obliku stringa, ovde se pretvara u xml objekat. Pretvaranje se vrši radi dalje obrade, sada objekat xml tipa XML, poseduje metode za manipulaciju podacima u xml-u. Objekat xml sadrži tri polja result, totalNumber i status. U delu koda **if** (xml.status == **true**) proverava se da li su podaci formatirani ispravno. Ako je uslov zadovoljen, formira se prazan niz StudentListVO.StudentiArray, deklariše se promenljiva xmlList tipa XMLList, i popunjava se tagovima student, koji sadrže podatke o studentima. Dakle, struktura response, koji je smešten u osobinu data, loader objekta, je:



Slika 3.7.19. Struktura xml fajla, result niz ima dva elementa

Na slici 3.7.19., se vidi, da koreni tag u Xml strukturi nosi naziv root. Podtagovi su result, totalNumber i status. Tag result, ovde predstavlja niz objekata student. Struktura objekta student je prikazana na slici, gde se vidi da su svi podaci smešteni po tagovima, oivičeni sa CDATA, i time se omogućava da sadržaj taga može sadržati i specijalne znakove. U for petlji određuje se broj tagova u nizu xmlList, pristupa se svakom polju jednog elementa niza i dodeljuje se njegov sadržaj u polja privremenog objekta std, tipa StudentVO. Na kraju se svaki objekat std dodaje u listu StudentListVO.StudentiArray, na sledeći način:

```
StudentListVO.StudentiArray.addItem(std);
```

Kada se popuni niz, poziva se metoda remove, koja vrši ukidanje slusaoca dogadjaja i setovanje URLLoader na vrednost null, t.j. destrukcija se izvrši.

```
private function on_getAllStudentiFlexHandler(event:Event):void {
	try {
		var xml:XML = new XML(event.target.data);
}
```

Ovde je pretvorena xml struktura sa strane servera, u XML objekat. On sadrži sve tagove, ali sadrži i metode koje omogućavaju pristup tim tagovima. Dalje, mogu se pretvoriti svi elementi u XMLList ili samo oni tagovi koji nas interesuju.

```
	catch(err:Error){
		trace("error " + err.message);
		return;
	}
	if (xml.status == true){
		StudentListVO.StudentiArray = new ArrayList();
		var xmlList:XMLList = xml..student;
		for (var i:int=0;i < xmlList.length();i++){
			var std:StudentVO = new StudentVO();
			std.address = xmlList[i].address;
			std.city = xmlList[i].city;
			std.country = xmlList[i].country;
			std.created_at = xmlList[i].created_at;
			std.department = xmlList[i].department;
			std.email = xmlList[i].email;
			std.first_name = xmlList[i].first_name;
			std.id = xmlList[i].id;
```

```
                    std.index_num = xmlList[i].index_num;
                    std.last_name = xmlList[i].last_name;
                    std.nickname = xmlList[i].nickname;
                    std.password = xmlList[i].region;
                    std.region = xmlList[i].address;
                    std.updated_at = xmlList[i].updated_at;
                    StudentListVO.StudentiArray.addItem(std);
              }
              StudentListVO.totalResults = xml..totalNumber;
       }
removeListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHand
ler);
       responseTime = new Date().time - time;
       if (StudentListVO.xmlDataProvider == null)
              StudentListVO.xmlDataProvider = new ArrayCollection();
              var obj:Object = new Object();
              obj.callNumber = steper;
              obj.timeValue = responseTime / 1000;
              obj.average = null;

              StudentListVO.xmlDataProvider.addItem(obj);
              average += obj.timeValue;
              steper++;
              if (steper == 52){
for (var i:int=0;i<StudentListVO.xmlDataProvider.length;i++){
       StudentListVO.xmlDataProvider.getItemAt(i).average = average / 50;
}
              if (StudentListVO.xmlAvarage == null)
                    StudentListVO.xmlAvarage = new ArrayCollection();
StudentListVO.xmlAvarage.addItem({data:StudentListVO.requestNumber,average:
average / 50});
       }
       sendNotification(ApplicationFacade.STUDENT_LOADED);
}

public function addStudentiFlex(student:StudentVO):void {
       var vars:URLVariables = new URLVariables();
       vars.address = student.address;
       vars.city = student.city;
       vars.country = student.country;
       vars.department = student.department;
```

```
		vars.email = student.email;
		vars.first_name = student.first_name;
		vars.index_num = student.index_num;
		vars.last_name = student.last_name;
		vars.nickname = student.nickname;
		vars.password = student.password;
		vars.region = student.region;
		vars.action = StudentUtils.ADD;
addListeners(on_addStudentiFlexHandler,on_getAllStudentiFlexErrorHandler,"S
tudentAPIXml.php",URLRequestMethod.POST,vars);
}

private function on_addStudentiFlexHandler(event:Event):void{
		try {
			var xml:XML = new XML(event.target.data);
		}
		catch(err:Error){
			return;
		}
		if (xml.status == true){

		sendNotification(ApplicationFacade.STUDENT_ADDED,xml..result);
		}
}
```

U delu koda **if** (xml.status == **true**) proverava se da li je osobina status, xml objekta postavljena na true, ako jeste, to znači da su podaci uspešno učitani iz baze podataka na strani servera. Naredbom:

```
var xmlList:XMLList = xml..student;
```

pristupa se xml strukturi, u xml objektu, objektima student, i smeštaju se u niz xmlList, koji je tipa XMLList. Naime, xml..student, je skraćeni zapis for petje, na ovaj način pristupa se svim elementima student u xml strukturi, koja se nalazi u objektu xml. Dakle, XMLList je klasa. xmlList je objekat klase XMLList. Klasa XMLList pravi listu,naime, pravi praktično array xml objekata, po nekom zadatom nodu xml..student, iz xml strukture. xml..student prolazi kroz čitav xml i traži nodove student. Sa dve tačke može se pristupiti bilo čemu, znači u čitavoj

strukturi sa dve tačke traži se student, i kada nađe student smešta se sve u jedan array. Prvo se student pretvori u jedan objekat, pa se smesti u array XMLList. XMLList je na nivou array.

xmlList[0]		xmlList[1]			xmlList[n]	
Student		***student***			***student***	
id	*71*	*Id*	*76*		*id*	*76*
email	*mca.ptu@gmail.com*	*Email*	*ftnns@gmail.com*		*email*	*ftn@gmail.com*
first_name	*Ritika*	*first_name*	*Chinky*		*first_name*	*Chinky*
last_name	*Mehra*	*last_name*	*Mehra*		*last_name*	*Mehra*
nickname	*hotski7*	*nickname*	*Barry*		*nickname*	*Barry*
country	*CA*	*country*	*CA*		*country*	*CA*
region	*Ontario*	*region*	*Ontario*		*region*	*Ontario*
city	*Toronto*	*City*	*Toronto*	...	*city*	*Toronto*
department	*1*	*department*	*2*		*department*	*2*
address		*address*			*address*	
created_at	*2013-01-27 07:49:59*	*created_at*	*2013-07-21 18:16:39*		*created_at*	*2013-07-21 18:16:39*
updated_at	*2013-01-27 07:49:59*	*updated_at*	*2013-09-27 07:49:59*		*updated_at*	*2013-09-27 07:49:59*
index_num	*15*	*index_num*	*15*		*index_num*	*15*

Slika3.7.20. Struktura xmlList niza

```
StudentiArray je ArrayList t.j.
      [Bindable]
      public static var StudentiArray:ArrayList;
```

Dakle, event.target.data je xml struktura, ali AS3 jeziku ona ne znači ništa. Za AS3 je to neki string koji prati neku strukturu. Da bi mogao da se napravi objekat, sa kojim može da se radi u AS3 jeziku, a zna se da je prihvaćen xml u event.target.data, koristi se klasa XML. Klasa XML kreira novi xml objekat. Mora se koristiti konstruktor za kreiranje xml objekta, pre nego što se pozove bilo koja metoda xml klase [1]. XML klasa uzima reprezent event.target.data, koji predstavlja xml i kreira objekat sa kojim može da se radi, i to smesti u objekat xml. Objekat xml, sadrži xml tagove. Xml je u obliku AS3 objekta, gde semože pristupiti direktno pojedinim elementima strukture xml-a. U delu koda xml.status, pristupa se tagu status.

U objektu xml, xml ima istu strukturu kao što je i struktura xml-a. XML je helper klasa, koja omogućava manipulaciju xml tagovima ili nodovima. Objekat xml klase XML, pored čitanja, omogućava i dodavanje elemenata u postojeću strukturu xml-a. Jedan od razloga pretvaranja stringa sastavljenog od strukture xml-a u objekat xml, jeste da bi se došlo do tih xml-a, koristeći dve tačke xml..student. Dve tačke su kraći zapis for petlje. Ovo znači pogledaj xml strukturu i nađi sve tagove imena student, i kada se nađu, izdvajaju se, i smeštaju se u xmlList:XMLList = xml..student;

Na ovaj način, se pređe kroz celu strukturu, pokupe se samo student i smeste se u XMLList, dakle smesti ih u jedan array, i dodeli im numeričke indekse 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Kada im se dodele numerički indeksi, sada postoji jedan array, koji se zove xmlList. xmlList je objekat, pošto je AS3 objektno orijentisan jezik. Objekat xmlList pristupa svojim osobinama pomoću specifikatora pristupa tačka, npr. std.address = xmlList[i].address, ako ne pronađe osobinu, izbaci će poruku da data osobina nije definisana, ne postoji a neće izbaciti grešku.

U for petlji se svi elementi iz svakog člana niza students kopiraju u polja objekta std, klase StudentVO. Na kraju se svaki svaki objekat std dodaje u listu StudentListVO.StudentiArray, na sledeći način:

```
StudentListVO.StudentiArray.addItem(std);
```

Ukupan broj elemenata se dodaje u statičko polje, naredbom:

```
StudentListVO.totalResults = xml..totalNumber;
```

U funkciji, naredbom responseTime=**new** Date().time - time; se računa vreme odziva. U narednom delu koda se proverava da li je snabdevač podataka za grafikon, xml formata instanciran, ako nije instancira se. Dakle, Kada se prvi put učitavaju podaci sa servera, on je prazan :

```
if(StudentListVO.xmlDataProvider ==null)
      StudentListVO.xmlDataProvider =new ArrayCollection();
```

Sada se kreira jedan generički, privremeni objekat obj tipa Object, koji ima tri polja. Privremeni, zato što se nalazi unutar funkcije, i živi unutar funkcije.

```
var obj:Object = new Object();
obj.callNumber = steper;
obj.timeValue = responseTime / 1000;
obj.average = null;
```

U polje obj.callNumber se skladišti broj poziva servera, dok se u polje obj.timeValue registruje vreme odziva. U polje obj.average čuva se prosečno vreme od svih učitavanja sa servera. Privremenom objektu obj se setuje osobina callNumber na vrednost steper. Object klasa je dinamička klasa. Dinamička klasa znači da u realno vreme ili u vreme izvršavanja, može da se dodaju elementi klase. Generička je osnovna klasa, početna klasa. Object jeste generička i dinamička klasa. U svojoj definiciji je prazna potpuno, ima samo mehanizme za kreiranje objekata. Kada se kaže obj.callNumber, ovaj objekat callNumber ne postoji, ali postoji u ovom momentu obj.callNumber, kada se deklariše. Ovde se koristi dinamička osobina ove klase i napravljen je callNumber i odmah je dodeljena vrednost steper. Ova vrednost steper je trenutno 1, zato što je na početku pri deklaraciji, izvršena inicijalizacija na vrednost jedan:

public var steper:int = 1;

To znači da će to biti prvi element u grafikonu. Zatim pravi se timeValue, timeValue objekat će da bude vreme odgovora, podeljeno sa hiljadu, obj.timeValue = responseTime / 1000;

Ovde se deli sa 1000 da bi se dobile sekunde, pošto je vreme u milisekundama. Zatim imamo objekat average, on je postavljen na nula, jer se taj average kasnije izračunava.

```
obj.callNumber = steper;
obj.timeValue = responseTime / 1000;
obj.average = null;
```

-obj.average = null - na ovaj način je polje average kreirano i dodeljena mu je vrednost null. Vrednost koja je dodeljena polju obj.average, ovde nije bitno, jer se prepisuje nova vrednost kasnije.

Ovakav objekat obj, koji je privremeni, dodaje se u array xmlDataProvider, kao element niza.

```
StudentListVO.xmlDataProvider.addItem(obj);
```

Pošto se broj pokušaja učitavanja uveća za jedan naredbom steper++; sledi slanje notifikacije, kojom se obaveštava listener da su podaci uspešno učitani:

```
sendNotification(ApplicationFacade.STUDENT_LOADED);
```

Metoda koja obrađuje promenu podataka u redu rešetke je changeStudentiDataFlex. Ovoj metodi kao parameter se prosleđuje objekat tipa StudentVO. U metodi se instancira privremeni, dinamički objekat vars, tipa URLVariables. U objekat vars se kreiraju i popunjavaju polja iz objekta student. Na kraju se dodaje naziv operacije u polje action. Naziv operacije je smešten u konstantu StudentUtils.CHANGE. Na kraju se poziva privatna metoda addListeners, koja

sadrži pet formalnih argumenata. Prvi je nazivfunkcije, koja obrađuje uspešno učitane podatke sa server na klijentu, drugi argument je funkcija koja obrađuje slučaj, kada dođe do greške pri učitavanu podataka, treći parameter predstavlja naziv klase na server, koja se instancira a potom odgovara na zahteve klijentske strane. Četvrti argument predstavlja naziv metode, kojom se razmenjuju podaci između klijenta i server, a peti argument definiše blok podataka, koji se šaju POST metodom server.

```
public function changeStudentiDataFlex(student:StudentVO):void
{
	var vars:URLVariables = new URLVariables();
	vars.address = student.address;
	vars.city = student.city;
	vars.country = student.country;
	vars.department = student.department;
	vars.email = student.email;
	vars.first_name = student.first_name;
	vars.index_num = student.index_num;
	vars.last_name = student.last_name;
	vars.nickname = student.nickname;
	vars.password = student.password;
	vars.region = student.region;
	vars.id = student.id;
	vars.action = StudentUtils.CHANGE;

addListeners(on_changeStudentiDataFlexHandler,on_getAllStudentiFlexErrorHan
dler, "StudentAPIXml.php",URLRequestMethod.POST,vars);
}
```

Podaci poslati sa servera klijentu, obrađuju se u metodi on_changeStudentiDataFlexHandler. String u xml format smešta se u osobini data, URLoadera objekta. Podaci ovog formata se smeštaju u XML objekat, koji se zove xml. U metodi se proverava da li je ispravno izvršeno editovanje podataka, ispitujući sadržaj promenljive status.

```
private function on_changeStudentiDataFlexHandler(event:Event):void
{
	try {
		var xml:XML = new XML(event.target.data);
	}
	catch(err:Error){
		return;
```

```
	}
	if (xml.status == true){

	}
	else {
		Alert.show("Nije uspelo editovanje");
	}

removeListeners(on_changeStudentiDataFlexHandler,
on_getAllStudentiFlexErrorHandler);
}
```

Metoda za brisanje sadržaja jednog sloga u bazi podataka je deleteStudentiFlex. U telu metode poziva se privatna metoda addListeners, koja uspostavlja konekciju sa serverskom klasom, koja opslužuje klijentske zahteve.

```
public function deleteStudentiFlex(StudentiId:int):void
{
addListeners(on_deleteStudentiFlex,on_getAllStudentiFlexErrorHandler,"Stude
ntAPIXml.php?action="+StudentUtils.DELETE+"&studentId="+StudentiId,
URLRequestMethod.GET);
}
```

Ako se uspešno izvrši brisanje sloga u bazi podataka, aktivira će se metoda on_deleteStudentiFlex. U try bloku pristigli podaci sa servera, smešteni u osobini data, u xml formatu, konvertuju se u xml objekat. Objekat xml ima tri osobine, result, totalNumber i status. Ako je polje status, objekta xml, setovano na true, što se proverava konstrukcijom if (xml.status == true) , to znači da je izvršeno brisanje sloga u bazi podataka i šalje se notifikacija sendNotification(ApplicationFacade.STUDENT_DELETED);. Notifikacija se registruje metodom listNotificationInterests, a obrađuje u metodi handleNotification. Ove metode se nalaze u klasi StudentManagerMediator.as.

```
private function on_deleteStudentiFlex(event:Event):void {
	try {
		var xml:XML = new XML(event.target.data);
	}
	catch(err:Error){
		return;
	}
	if (xml.status == true){
```

```
            sendNotification(ApplicationFacade.STUDENT_DELETED);
      }
      else {
            Alert.show("Nije uspelo brisanje");
      }
}
```

U aplikaciji, gde se primenjuje XML tehnologija, podaci se mogu filtrirati, I to pomoću metode filterStudents:

```
public function filterStudents(obj:Object):void
{
      var vars:URLVariables = new URLVariables();
      vars.name = obj.fname;
      vars.address = obj.address;
      vars.email = obj.email;
      vars.start = obj.start;
      vars.requested = obj.requested;
      vars.department = obj.department;
      vars.action = StudentUtils.FILTER_STUDENTS;
addListeners(filterStudentsResultHanlder,on_getAllStudentiFlexErrorHandler,
"StudentAPIXml.php",URLRequestMethod.POST,vars);
}
```

Metodi filterStudents se kao parametar prosleđuje objekat koji sadrži podatke iz TextInput polja, koji služe za filtriranje (slika 3.7.9.). U telu metode filterStudents deklariše se promenljiva vars tipa URLVariables. Istovremeno sa deklaracijom, pristupa se i alociranju memorije za objekat klase URLVariables. Alociranom objektu, putem reference vars, dodeljuju se vrednosti iz objekta obj, koji je tipa Object. Objekat klase URLVariables je dinamičkog tipa. Na kraju mu se dodeljuje naziv operacije, putem statičke konstante FILTER_STUDENTS. Na kraju se poziva funkcija addListeners, koja uspostavlja vezu sa serverom. Metoda addListeners kao formalne argument poseduje naziv funkcije, koja obrađuje učitane podatke sa servera, funkciju koja obrađuje grešku u slučaju neuspešne konekcije sa serverom, naziv klase na serveru, koja se instancira i odgovara na zahteve klijenta, naziv metode i podatke koji se prosleđuju serveru sa klijentske strane.

Podaci koji se dostave sa servera na klijentsku aplikaciju, obrađuju se u metodi:

```
private function filterStudentsResultHanlder(event:Event):void
{
```

```
	try {
		var xml:XML = new XML(event.target.data);
	}
	catch(err:Error){
		trace("error " + err.message);
		return;
	}
	if (xml.status == true){
		StudentListVO.StudentiArray = new ArrayList();
		var xmlList:XMLList = xml..student;
		for (var i:int=0;i < xmlList.length();i++){
			var std:StudentVO = new StudentVO();
			std.address = xmlList[i].address;
			std.city = xmlList[i].city;
			std.country = xmlList[i].country;
			std.created_at = xmlList[i].created_at;
			std.department = xmlList[i].department;
			std.email = xmlList[i].email;
			std.first_name = xmlList[i].first_name;
			std.id = xmlList[i].id;
			std.index_num = xmlList[i].index_num;
			std.last_name = xmlList[i].last_name;
			std.nickname = xmlList[i].nickname;
			std.password = xmlList[i].region;
			std.region = xmlList[i].address;
			std.updated_at = xmlList[i].updated_at;

			StudentListVO.StudentiArray.addItem(std);
		}
		StudentListVO.totalResults = xml..totalNumber;
	}
removeListeners(filterStudentsResultHanlder,on_getAllStudentiFlexErrorHandler);
}
```

U try bloku, podaci sa servera, u xml formatu, koji su smešteni u osobinu data, kovertuju se u XML objekat. Ako su podaci uspešno učitani, alocira se memorija za statički niz StudentListVO.StudentiArray. U promenljivu xmlList tpa XMLList, smeštaju se tagovi student iz objekta xml. U for petlji prolazi se kroz sve objekte xmlList-e, pristupa se svakom polju unutar objekta, po elementima xmlList-e, i kopira se u polje privremenog objekta std tipa StudentVO. Kada se popune sva polja, onda se privremeni objekat ubacuje u statički niz,kojim se popunjava rešetka naredbom StudentListVO.StudentiArray.addItem(std);. Na kraju, kada se

izvrše sve iteracije u for petlji, u niz StudentiArray ubacuje se ukupan broj pristiglih podataka sa server, naredbom StudentListVO.totalResults = xml..totalNumber;. Na kraju obrade u ovoj metodi, poziva se funkcija removeListeners, koja ukida slušaoce događaja i setuje URLLoader na vrednost null.

```
private function filterStudentsResultHanlder(event:Event):void
{
	try {
		var xml:XML = new XML(event.target.data);
	}
	catch(err:Error){
		trace("error " + err.message);
		return;
	}
	if (xml.status == true){
		StudentListVO.StudentiArray = new ArrayList();
		var xmlList:XMLList = xml..student;
		for (var i:int=0;i < xmlList.length();i++){
			var std:StudentVO = new StudentVO();
			std.address = xmlList[i].address;
			std.city = xmlList[i].city;
			std.country = xmlList[i].country;
			std.created_at = xmlList[i].created_at;
			std.department = xmlList[i].department;
			std.email = xmlList[i].email;
			std.first_name = xmlList[i].first_name;
			std.id = xmlList[i].id;
			std.index_num = xmlList[i].index_num;
			std.last_name = xmlList[i].last_name;
			std.nickname = xmlList[i].nickname;
			std.password = xmlList[i].region;
			std.region = xmlList[i].address;
			std.updated_at = xmlList[i].updated_at;
			StudentListVO.StudentiArray.addItem(std);
		}
		StudentListVO.totalResults = xml..totalNumber;
	}
removeListeners(filterStudentsResultHanlder,on_getAllStudentiFlexErrorHandl
er);
}
```

Metoda addListeners, je zajednička, kreira URLLoader, url request. Koristi se pri učitavanju podataka sa servera, brisanju, izmeni i dodavanju podataka. U ovoj metodi dodaju se slušaoci događaja za load podataka sa servera u obliku fajla i kreira se request sa parametrima, selektuje se method requesta (POST ili GET). U delu koda:

```
loader.addEventListener(Event.COMPLETE, onResult);
```

dodaje se eventListener za svaki http upit, posebno se uvek izvršava, kada postoji odgovor sa servera (status 200), bez obzira na rezultat izvršenja zahteva

```
private functionaddListeners(onResult:Function, onError:Function,
url:String, method:String, variables:URLVariables  = null):void{
loader = new URLLoader();
loader.addEventListener(Event.COMPLETE, onResult);
loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
var request:URLRequest = new URLRequest(StudentProxyXml.serverName + url);
request.method = method;
            if (variables)
                  request.data = variables;
            loader.load(request);
      }
```

Metoda za obradu greške, u slučaju da se ne učitaju ispravno podaci.

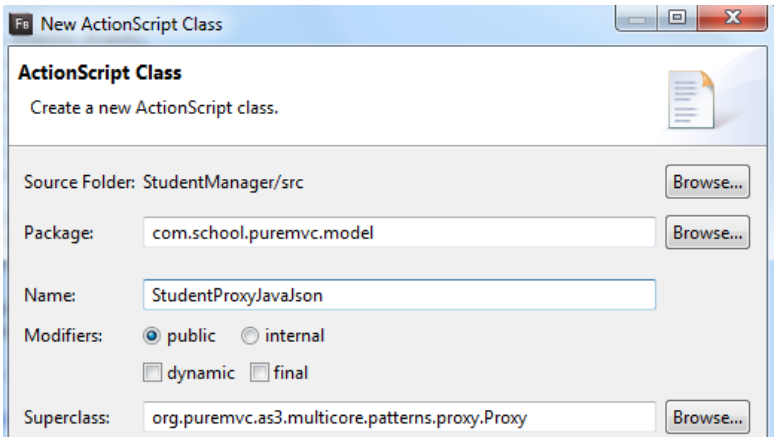
```
private function
on_getAllStudentiFlexErrorHandler(error:IOErrorEvent):void{Alert.show(error
.text);}
```

Metoda *removeListeners* ukida slušaoca događaja i setuje URLLoader na vrednost null, t.j. destruiše.

```
private function removeListeners(onResult:Function,onError:Function):void{
                  if (loader){
      loader.removeEventListener(Event.COMPLETE,onResult);
      loader.removeEventListener(IOErrorEvent.IO_ERROR,onError);
                        loader = null;
                  }
            }
      }
}
```

3.7.3 Proksi klasa StudentProxyJavaJson

ActionScript klasa StudentProxyJavaJson koristi JSON format za formatiranje podataka, uspostavlja vezu sa Java serverom. Klasa se nalazi u paketu model. Paket model se selektuje, klikom na desni taster otvara se meni, u kome se bira opcija ActionScript class (slika 3.7.3.) Odabirom stavke ActionScript class, dobija se dijalog prozor.



Slika 3.7.21. Dijalog za kreiranje ActionScript klase StudentProxyJavaJson

U prozoru se upisuje naziv klase, paket u kome je smeštena klasa, ovde se zove paket model. Ova klasa proksi nasleđuje klasu Proxy i implementira interfejs IProxy. Klasa StudentProxyJavaJson.as sadrži dve statičke javne konstante NAME i serverName, dve statičke javne osobine totalResults i requestNumber, dve javne osobine steper i average, tri private osobine loader, time i responseTime, i dvanaest metoda. Javna metoda konstruktor je nivoa klase, četiri su javne i sedam privatne metode. Globalne osobine i konstante su:

```
private var loader:URLLoader;
public static const NAME:String = "StudentProxyJavaJson";
private var loader:URLLoader;
 [Bindable]
public static var totalResults:int;
public static var requestNumber:int = 500;
private var time:Number;
private var responseTime:Number;
public var steper:int = 1;
public var average:Number = 0;
public static const serverName:String = "http://localhost:9000";
```

Objekat član loader, tipa URLLOader je zadužen za uspostavljanje konekcije između RIA aplikacije i servera. Podatak član totalResults sadrži broj podataka koje treba učitati. U polje time se skladišti vreme slanja zahteva serveru, a responseTime čuva ukupno vreme od trenutka slanja zahteva, do dostavljanja podataka iz servera. U podatku članu steper, registruje se redni broj zahteva za podacima na serveru. Osobina average je zadužena za skladištenje prosečnog vremena za ukupan broj učitavanja.
Statička javna konstanta serverName čuva stazu do foldera na serveru, gde su smešteni servisi zaduženi za izvršavanje zahteva, koji se uputi sa strane RIA aplikacije. Port na kome radi Java server je 9000. Konstanta NAME sadrži naziv objekta, koji se instancira na osnovu klase koja je zadužena za komunikaciju sa serverom, a pri tome se podaci formatiraju u json format.

Klasa *StudentProxyJson* sadrži metode:

- *metodu konstruktor* `StudentProxyJavaJson`
- *getAllStudentiFlex*
- *on_getAllStudentiFlexHandler*
- *addStudentiFlex*
- *addStudentiFlexResultHandler*
- *deleteStudentiFlex*
- *deleteStudentiFlexResultHandler*
- *changeStudentiDataFlex*
- *changeStudentiDataFlexFaultHandler*
- *addListeners*
- *on_getAllStudentiFlexErrorHandler*
- *removeListeners*

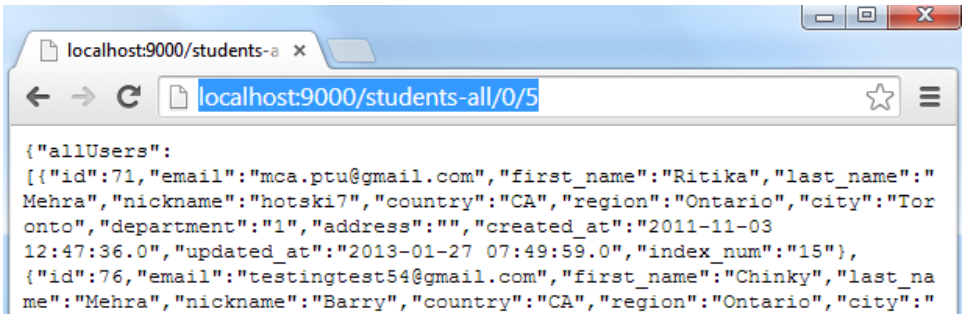
Pri instanciranju objekta klase StudentProxyJavaJson, poziva se konstruktor metoda StudentProxyJavaJson.

```
public function StudentProxyJavaJson (){
      super(NAME);
}
```

Naredbom **super**(NAME); poziva se konstruktor bazne klase Proxy, i prosleđuje mu se parametar NAME. U RIA aplikaciji se učitavaju podaci metodom getAllStudentiFlex. Ova metoda poziva metodu addListeners , u kojoj se instancira objekat URLLoader, koji uspostavlja vezu sa serverom, i koja instancira objekat klase na serveru, sa kojom se uspostavlja veza.

```
public function getAllStudentiFlex(start:int):void{
addListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHandler
,"/students-
all/"+start+"/"+StudentListVO.requestNumber,URLRequestMethod.GET);
time = new Date().time;
}
```

Dakli, metoda getAllStudentiFlex, poziva metodu addListeners, prosleđujući joj pet formalnih argumenata, nazif metode on_getAllStudentiFlexHandler, koja obrađuje uspešno učitane podatke sa strane server, naziv metode on_getAllStudentiFlexErrorHandler, koja se poziva ako klijentska strana ne može da učita podatke sa serverske strane, adresa metode, koja se poziva na strain server, sa vrednostima parametara, zahtevani broj podataka iz baze i naziv metode koja se koristi. Ako je start=0, requested=2, naziv pozivajuće serverske metode students-all, struktura url adrese, kao oblik učitanih podataka na izlazu ima oblik:



Slika 3.7.22. Prikaza json fajla u Exploreru, kreiranog na Java server.

Metod koji obrađuje uspešno učitane podatke sa strane servera je on_getAllStudentiFlexHandler. U try bloku vrši se konverzija stringa Json formata iz osobine data u JSON objekat json_var. Objekat json_var sadrži sve podatke kao I string data, i poseduje metode za manipulaciju podacima. Niz allUsers iz objekta json_var smešta se u nizovnu promenljivu students, onda se alocira memorija za statički niz StudentiArray, klase StudentListVO, koji se korisete za popunjavanje dataProvider-a rešetke. U For petlji se vrši iteracija po svim poljima niza students. U for petlji se kreira lokalni objekat std tipa StudentVO, i u njega se popunjavaju osobine redom iz dvakog elementa niza students. A onda se na kraju privremeni objekat std smesti u niz StudentiArray. Kada se kopiraju svi objekti iz

students u StudentiArray, onda se upisuje ukupan broj elemenata iz baze u promenljivu StudentListVO.totalResults. Naime, event je objekat Event klase, pojavljuje se u trenutku kad je load funkcija uspešno izvršena, event.target je objekat koji je izvršio load podataka, u našem slučaju je to loader. loader je tipa URLLoader i ima osobinu data, koja služi za skladištenje podataka dobijenih lodovanjem dokumenta.

```
private function on_getAllStudentiFlexHandler(event:Event):void{
      try {
      var json_var = JSON.parse(event.target.data.toString());
      var students:Array = json_var.allUsers as Array;
        StudentListVO.StudentiArray = new ArrayList();

      for (var i:int=0;i<students.length;i++){
            var std:StudentVO = new StudentVO();
            std.address = students[i].address;
            std.city = students[i].city;
            std.country = students[i].country;
            std.created_at = students[i].created_at;
            std.department = students[i].department;
            std.email = students[i].email;
            std.first_name = students[i].first_name;
            std.id = students[i].id;
            std.index_num = students[i].index_num;
            std.last_name = students[i].last_name;
            std.nickname = students[i].nickname;
            std.password = students[i].region;
            std.region = students[i].address;
            std.updated_at = students[i].updated_at;

            StudentListVO.StudentiArray.addItem(std);
          }
          StudentListVO.totalResults = json_var.totalNumber;
removeListeners(on_getAllStudentiFlexHandler,
on_getAllStudentiFlexErrorHandler);
      }
            catch(err:Error){ Alert.show(err.message);}
responseTime = new Date().time - time;

if (StudentListVO.jsonJavaDataProvider == null)
          StudentListVO.jsonJavaDataProvider = new ArrayCollection();
```

```
        var obj:Object = new Object();
        obj.callNumber = steper;
        obj.timeValue = responseTime / 1000;
        obj.average = null;
        obj.requested = StudentListVO.requestNumber;
        StudentListVO.jsonJavaDataProvider.addItem(obj);
        average += obj.timeValue;
        steper++;
if (steper == 52){
for (var i:int=0;i<StudentListVO.jsonJavaDataProvider.length;i++){
    StudentListVO.jsonJavaDataProvider.getItemAt(i).average = average / 50;
}
}
        sendNotification(ApplicationFacade.STUDENT_LOADED);
}
```

U iniji koda responseTime = new Date().time - time; određuje se ukupno vreme koje protekne od trenutka slanja zahteva serveru, do trenutka učitavanja podataka sa servera. U narednim linijama koda proverava se da li je niz jsonJavaDataProvider prazan, ako jeste alocira se memorija i popunjava odgovarajućim vrednostima, koji se koristi kao dataProvider za grafikon koji se crta. I na kraju se određuje prosečno vreme za pedeset puta učitavanja podataka.

U metodi addListeners dodaju se slušaoci događaja za load podataka sa servera u obliku fajla. Ovde se kreira request sa parametrima, slektuje se method requesta (POST ili GET). U liniji koda loader = new URLLoader(); vrši se instanciraje URLLoader klase. U drugoj liniji koda loader.addEventListener(Event.COMPLETE, onResult); vrši se dodavanje eventListenera za svaki http upit posebno i izvršava se uvek kada postoji odgovor sa servera (status 200), bez obzira na rezultat izvršenja zahteva.

U trećoj liniji koda

```
loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
```

trigeruje se događaj, t.j. izvršava se u slučajevima kada ne postoji odgovor sa servera (status 404, 500 i dr.).

Naredbom **var** request:URLRequest = **new** URLRequest(StudentProxyJavaJson.serverName + url); realizuje se pravljenje request-a sa parametrima (url adresa sa GET parametrima). Izrazom request.method = method; selektuje se POST ili GET metoda. Ako je POST metod

postoje parametri u obliku URLVariables klase, što se proverafa if konstrukcijom. I na kraju sledi izvršenje poziva ka url adresi loader.load(request);

```
private function
addListeners(onResult:Function,onError:Function,url:String,method:String,va
riables:URLVariables = null):void{
loader = new URLLoader();
loader.addEventListener(Event.COMPLETE, onResult);
loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
var request:URLRequest=new URLRequest(StudentProxyJavaJson.serverName+ url);
      request.method = method;
      if (variables)request.data = variables;
      loader.load(request);
}
```

Metoda koja ispisuje poruku o grešci.

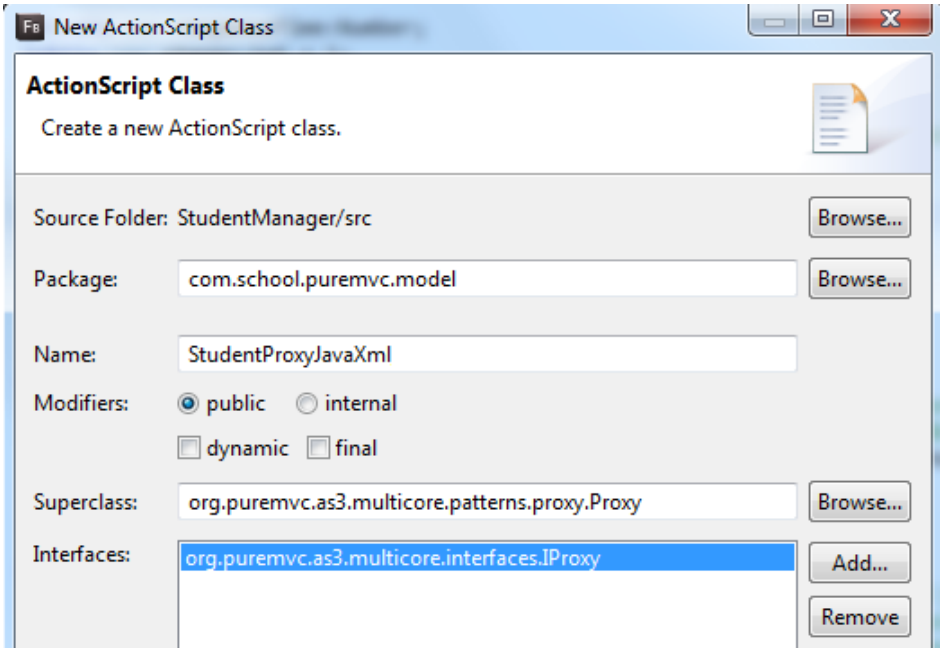
```
private function
on_getAllStudentiFlexErrorHandler(error:IOErrorEvent):void{
      Alert.show(error.text);
}
```

Metod removeListeners vrši ukidanje slušaoca događaja i setovanje URLLoader-a na vrednost null, dakle, vrši se destrukcija URLLoader objekta.

```
private function removeListeners(onResult:Function,onError:Function):void{
  if (loader){
      loader.removeEventListener(Event.COMPLETE,onResult);
      loader.removeEventListener(IOErrorEvent.IO_ERROR,onError);
      loader = null;
    }
}
```

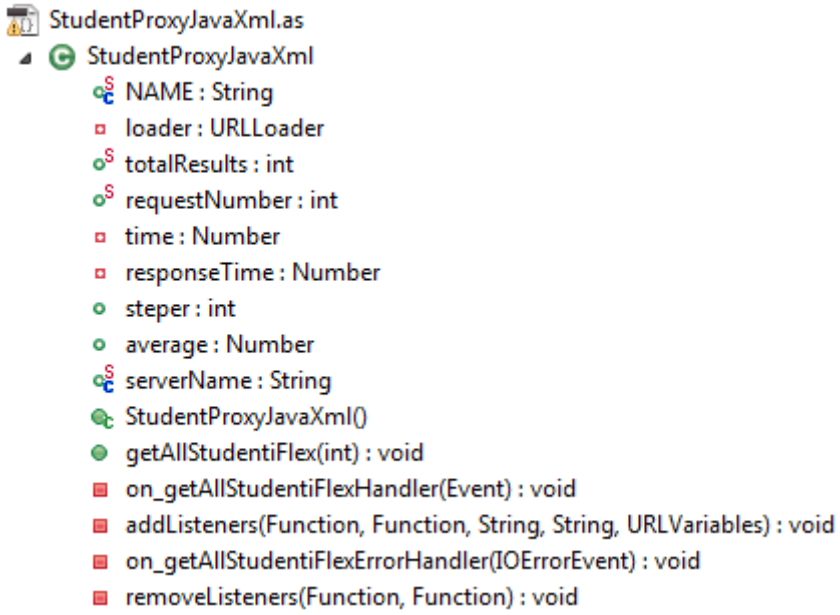
3.7.4 Proksi klasa StudentProxyJavaXml.as

Proksi klasa StudentProxyJavaXml.as koristeći XML format za pakovanje podataka, uspostavlja vezu sa Java serverom. Xml document ima otvorene i zatvorene tagove a između su podaci. Same klase za obradu kod Xml-a su glomaznije nego kod Json-a. zato je xml sporiji. Klasa StudentProxyJavaXml se nalazi u paketu model, pravi se pritiskom na desni taster paketa model, koji se nalazi u prostoru Package explorer-a (slika 3.7.3.), a onda izborom stavke ActionScript Class, dobija se dijalog prozor.



Slika 3.7.23. Dijalog za definisanje ActionScript klase StudentProxyJavaXml

U dijalog prozoru navedeno je ime klase, naziv roditeljske klase[52]i naziv interfejsa. Pritiskom na dugme Finish, kreira se klasa StudentProxyJavaXml.as. Klasa StudentProxyJavaXml.as u svojoj strukturi poseduje dve javne statičke konstante NAME i serverName, dve javne statičke osobine totalResults i requestNumber, dve javne osobinu steper i average, tri private osobine loader, responseTime i time, jednu metodu nivoa klase[52] StudentProxyJavaXml, jednu javnu metodu i četiri privatne metode. Podaci članovi, objekti članovi i metode su prikazani na slici 3.7.24.



Slika 3.7.24. Prikaz osobina i metoda za klasu StudentProxyJavaXml

Osobine klase su:

```
public static const NAME:String = "StudentProxyJavaXml";
private var loader:URLLoader;
[Bindable]
public static var totalResults:int;
public static var requestNumber:int = 500;
private var time:Number;
private var responseTime:Number;
public var steper:int = 1;
public var average:Number = 0;
public static const serverName:String = "http://localhost:9000";
```

Objekat član loader, tipa URLLOader uspostavlja vezu između RIA aplikacije i servera [1]. U polje time se čuva vreme slanja zahteva serveru, a responseTime arhivira ukupno vreme od trenutka slanja zahteva, do dostavljanja podataka iz servera. U podatku članu steper, registruje se redni broj zahteva za podacima na serveru. Osobina average je zadužena za skladištenje prosečnog vremena za ukupan broj učitavanja.
Statička javna konstanta serverName čuva stazu do foldera na serveru, gde su smešteni servisi zaduženi za izvršavanje zahteva, koji se uputi sa strane RIA aplikacije. Konstanta NAME sadrži naziv objekta, koji se instancira na osnovu klase koja je zadužena za komunikaciju sa serverom, a pri tome se podaci formatiraju u xml format. Osobina requestNumber sadrži broj podataka koji se učitava iz baze.Metode koje se koriste u ovoj klasi, date su u listi:

- *StudentProxyJavaXml*
- *getAllStudentiFlex*
- *on_getAllStudentiFlexHandler*
- *addListeners*
- *on_getAllStudentiFlexErrorHandler*
- *removeListeners*

Konstruktor klase StudentProxyJavaXml.

```
public function StudentProxyJavaXml()
{
      super(NAME);
}
```

Metod getAllStudentiFlex, se poziva kada se učitavaju podaci iz baze podataka.

```
public function getAllStudentiFlex(start:int):void{
addListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorH
andler,"/students-all-
xml/"+start+"/"+StudentListVO.requestNumber,URLRequestMethod.GET);
time = new Date().time;
}
```

Metod on_getAllStudentiFlexHandler obrađuje uspešno učitane podatke sa server.

```
private function on_getAllStudentiFlexHandler(event:Event):void{
//trace("ovo je JSON")
try {
      try { var xml:XML = new XML(event.target.data); }
      catch(err:Error){
            trace("error " + err.message);
            return;
      }
      if (xml.status == true){
            StudentListVO.StudentiArray = new ArrayList();
      var xmlList:XMLList = xml..student;
      for (var i:int=0;i < xmlList.length();i++){
      var std:StudentVO = new StudentVO();
      std.address = xmlList[i].address;
      std.city = xmlList[i].city;
      std.country = xmlList[i].country;
      std.created_at = xmlList[i].created_at;
      std.department = xmlList[i].department;
      std.email = xmlList[i].email;
      std.first_name = xmlList[i].first_name;
      std.id = xmlList[i].id;
      std.index_num = xmlList[i].index_num;
      std.last_name = xmlList[i].last_name;
      std.nickname = xmlList[i].nickname;
      std.password = xmlList[i].region;
      std.region = xmlList[i].address;
      std.updated_at = xmlList[i].updated_at;
StudentListVO.StudentiArray.addItem(std);
}
StudentListVO.totalResults = xml..totalNumber;
}
removeListeners(on_getAllStudentiFlexHandler,on_getAllStudentiFlexErrorHandler);
}
```

```
        catch(err:Error){
        Alert.show(err.message);
        }
        responseTime = new Date().time - time;
        if (StudentListVO.xmlJavaDataProvider == null)
              StudentListVO.xmlJavaDataProvider = new ArrayCollection();
        var obj:Object = new Object();
        obj.callNumber = steper;
        obj.timeValue = responseTime / 1000;
        obj.average = null;
        obj.requested = StudentListVO.requestNumber;
        StudentListVO.xmlJavaDataProvider.addItem(obj);
        average += obj.timeValue;
        steper++;
if (steper == 52){
for (var i:int=0;i<StudentListVO.xmlJavaDataProvider.length;i++){
     StudentListVO.xmlJavaDataProvider.getItemAt(i).average = average / 50;
                  }
        }
        sendNotification(ApplicationFacade.STUDENT_LOADED);
}
```

Metod addListeners, vrši instanciranje URLLoader klase, dodaje eventListenera za svaki http upit posebno. Metoda onResult izvršava se uvek kada postoji odgovor sa servera (status 200), bez obzira na rezultat izvršenja zahteva. Metod onError izvršava se u slučajevima kada ne postoji odgovor sa servera (status 404, 500 i dr.). U liniji koda

```
var request:URLRequest=new URLRequest(StudentProxyJavaJson.serverName+
url);
```

pravi se request sa parametrima (url adresa sa GET parametrima). U metodi se vrši selekcija POST ili GET metoda. Ako je POST metod postoje parametri u obliku URLVariables klase i na kraju se izvršava poziv ka url adresi.

```
privatefunction
addListeners(onResult:Function,onError:Function,url:String,method:String,va
riables:URLVariables = null):void{
      loader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, onResult);
      loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
var request:URLRequest=new URLRequest(StudentProxyJavaJson.serverName+
url);
```

```
        request.method = method;
        if (variables)request.data = variables;
        loader.load(request);
}
```

Metod on_getAllStudentiFlexErrorHandler ispisuje poruku o grešći, u slučaju neuspešnog učitavanja podataka.

```
privatefunction on_getAllStudentiFlexErrorHandler(error:IOErrorEvent):void{
        Alert.show(error.text);
}
```

Metod removeListeners, ukida slušaoce događaja i setuje URLLoader na vrednost null.

```
private function removeListeners(onResult:Function,onError:Function):void{
        if (loader){
        loader.removeEventListener(Event.COMPLETE,onResult);
        loader.removeEventListener(IOErrorEvent.IO_ERROR,onError);
        loader = null;
                    }
              }
        }
}
```

3.7.5 Kreiranje pdf stranica i dodavanje teksta

Izgled stranice aplikacije može se predstaviti u pdf formatu, uz mogućnost dodavanja teksta ispod stranice. U glavnoj aplikaciji definisano je pet stanja. Mogućnost dodavanja pdf sadržaja, postoje za četiri stanja i to diagrams, javaPhp, javaPhpXml i comparation. Sadržaj stranice u pdf formatu se arhivira u niz, a na kraju se u stanju comparation kreira pdf fajl na željeno mesto. Metode koje kreiraju pdf format egzistiraju u dve klase AddTextPdfComponent i ApplicationMediator. Dugmad kojim se dodaje tekstualni sadržaj u pdf stranicu, kao i komponenta u kojoj postoji mogućnost pisanja, definisani su u klasi AddTextPdfComponent. Dugme kojim se dodaje sadržaj stranice u pdf format definisano je u klasama ResultDiagrams, JavaPhpComparationDiagram i ComparationDiagram, i ima oblik:

```
<s:Button
     id="addPage"
     label="Add PDF page"
     bottom="10" left="20"
/>
```

Instanca klase Button, se zove addPage. Na dugmetu je definisan naslov u osobini label, i podešene su pozicije dugmeta na stranici osobinama bottom i left. Dugme kojim se kreira pdf stranica na računaru, definisano je u klasi ComparationDiagram, oblika je:

```
<s:Button
    id="createPDF"
    label="Make PDF"
    bottom="10" right="20"
/>
```

U klasi ApplicationMediator definisan je listener na dugme addPage, u metodi onStateChangeHandler. Ova metoda se poziva, pri bilo kakvoj promeni stanja na glavnoj aplikaciji. Osluškivač na promenu stanja u glavnoj aplikaciji se dodaje u metodi onRegfister:

```
override publicfunction onRegister():void
{
  controlMediator = facade.retrieveMediator(StudentControlMediator.NAME) as
StudentControlMediator;
view.addEventListener(FlexEvent.STATE_CHANGE_COMPLETE,onStateChangeHandler);
 view.selectProxyBar.addEventListener(IndexChangeEvent.CHANGE,
onSelectedProxyHandler);
}
private function onStateChangeHandler(event:FlexEvent):void
{
      if (view.currentState == "diagrams"){
                              …
view.diagram.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.diagram.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
}
else if (view.currentState == "javaPhp") {
view.javaPhpComp.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);

view.javaPhpComp.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
}
else if (view.currentState == "comparation"){
```

```
 view.comparationDiagram.createPDF.removeEventListener(MouseEvent.CLICK,
onCreatePDFHandler);
view.comparationDiagram.createPDF.addEventListener(MouseEvent.CLICK,
onCreatePDFHandler,false,0,true);
view.comparationDiagram.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.comparationDiagram.addPage.addEventListener(MouseEvent.CLICK,onAddPage
Handler,false,0,true);
}
else if (view.currentState == "javaPhpXml"){     …
view.javaXmlCompar.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.javaXmlCompar.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
ApplicationFacade.SELECTED_PROXY = StudentProxyJavaXml.NAME;
      }
}
```

U telu metode prvo se proverava koje dugme je u fokusu, t.j. koje stanje je aktivno. Ako je pritisnuto dugme Diagrams, kao na slici 3.3.7.25., onda se pristupa dodale listenera dugmetu addPage.

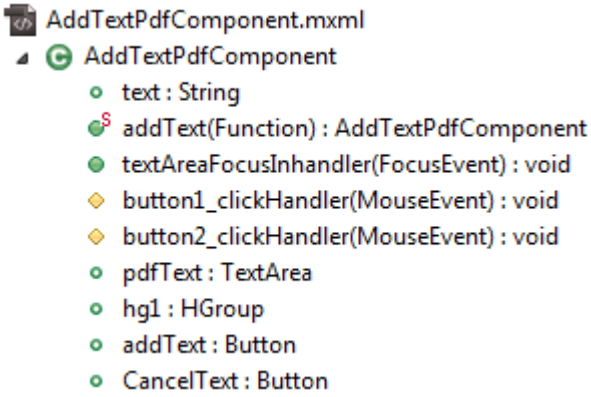
Slika 3.7.25. Prikaz komponente Diagrams.

Dakle, ovde se prvo ukloni listener, a onda se doda listener datom dugmetu. Pri dodavanju listenera na dugme, prvi parameter addEventListener-a je tip događaja, ovde je to klik t.j. MouseEvent.CLICK, i naziv metode koja će da bude aktivirana kada se klikne na dugme. U ovom slučaju to je metoda onAddPageHandler, koja ima oblik:

```
private function onAddPageHandler(evt:MouseEvent):void
{
   AddTextPdfComponent.addText(onAddPdfTextHandler);
}
```

Naime, kada se dodaje pdf, prvo se poziva metod onAddPageHandler, klikom na dugme addPage. U telu ove metode pristupa se komponenti AddTextPdfComponent, poziva se njena statička metoda addText, i kao argument prosledi se naziv metode onAddPdfTextHandler. Klasa AddTextPdfComponent sadrži pet javnih osobina i četiri metode.

Osobine i metode su date na slici 3.7.26.



Slika 3.7.26. Prikaz osobina i metoda klase AddTextPdfComponent

Dakle, kada se dodaje pdf, prvo se poziva metod addText iz AplicationManager-a. Parametar metode closeFunction, tipa Function je funkcijaonAddPdfTextHandler, koja se nalazi u AplicationManager. U telu metode addText, pravi se instanca, koja se zove pdf, klase AddTextPdfComponent. PopUpManager će dodati iskačući novi prostor. Naredbom

```
PopUpManager.addPopUp(pdf,FlexGlobals.topLevelApplication as Sprite,true);
```

PopUpManager koji predstavlja ceo ekran, zatamnjeni deo na slici3.6.44., dodaje jednu komponentu pdf, koja predstavlja objekat klase AddTextPdfComponent, ekranu. Naime, iskačući prozor, sa dugmetom *Add text* i *Cancel* predstavlja objekat klase AddTextPdfComponent. Prema tome, PopUpManager je ceo ekran, zatamnjeni deo. Objekat pdf se sastoji od velikog textArea i dva dugmeta Add text i Cancel:

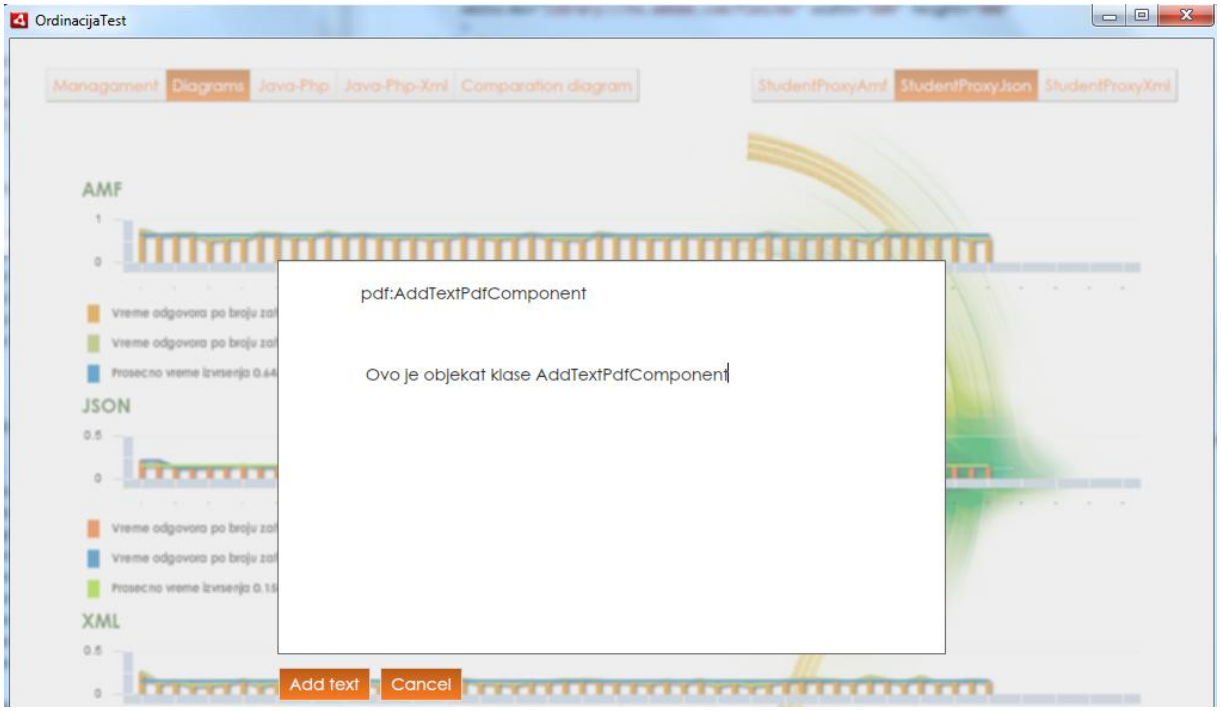
```
<s:TextArea
```

```
      top="20" left="20" right="20" bottom="50"
      id="pdfText" styleName="buttonFonts"
            />
<s:HGroup
      width="100%"
      bottom="10" left="20" right="20"
      >
      <s:Button
            label="Add text"
            click="button2_clickHandler(event)"
            />
      <s:Button
            label="Cancel"
            click="button1_clickHandler(event)"
            />
</s:HGroup>
```

Drugi parameter u konstrukciji:

```
PopUpManager.addPopUppdf,FlexGlobals.topLevelApplication as Sprite,true);
```

je FlexGlobals.topLevelApplication. FlexGlobals.topLevelApplication znači, obraćamo se Fleks globalnoj klasi, to je ovde main aplikacija. Funkcija addPopUp traži da se definiše koji objekat se želi prikazati u popup-u, traži roditelja, gde se želi prikazati taj popup[1]. Ovde se želi prikazati u main aplikaciji, unutar glavne aplikacije. Iz klase AddTextPdfComponent, može se pristupiti glavnoj aplikaciji pomoću FlexGlobals klase, dakle pomoću FlexGlobals.topLevelApplication uhvati će se StudentAMF.mxml klasa. Ovde se prikazuje FlexGlobals.topLevelApplication kao Sprite, jer topLevelApplication je tipa Object. A metoda addPopUp traži da ne bude object, već DisplayObject. To se zna da je DisplayObject, ali se u compile modu nezna. Zato se kaže as Sprite. Sprite je osnovni DisplayObject. Treći parameter true, ukazuje da je modal, što znači da ostatak prozora se zasivi, i van popup prozora, ne dopušta interakciju korisnika van popup menadžera.



Slika3.7.27. Prikaz prompta

```
publicstaticfunction addText(closeFunction:Function):AddTextPdfComponent
{
var pdf:AddTextPdfComponent = new AddTextPdfComponent();

PopUpManager.addPopUp(pdf,FlexGlobals.topLevelApplication as Sprite,true);
PopUpManager.centerPopUp(pdf);
pdf.addEventListener(CloseEvent.CLOSE, closeFunction);
//Dodeli se listener CLOSE, kada se zatvara ova komponenta. Pozvaće
//closeFunction, a closeFunction se prosleđuje svana
pdf.pdfText.scroller.addEventListener(FocusEvent.FOCUS_IN,
pdf.textAreaFocusInhandler,false,1);
return pdf;
}
```

U delu koda pdf.addEventListener(CloseEvent.CLOSE, closeFunction); objektu pdf, dodaje se EventListener CloseEvent. Znači kada se zatvori pozvaće se funkcija closeFunction. closeFunction je prosleđena kao parameter spolja, iz metode onAddPageHandler, koja se nalazi u klasi AplicationMediator. Metoda onAddPageHandler ima oblik:

```
private function onAddPageHandler(evt:MouseEvent):void
{
    AddTextPdfComponent.addText(onAddPdfTextHandler);
}
```

To znači, kada se zatvori prozor, on će da pozove metodu onAddPdfTextHandler.

```
private function onAddPdfTextHandler(event:CloseEvent):void
{
	var pdfObject:Object = new Object();
	pdfObject.text = event.currentTarget.text;
	if (!pages)
		pages = new Array();
	var page:BitmapData;
	if (view.currentState == "diagrams"){
		page =
ImageSnapshot.captureBitmapData(view.diagram.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
else if (view.currentState == "comparation"){
 page =
ImageSnapshot.captureBitmapData(view.comparationDiagram.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	else if (view.currentState == "javaPhp"){
	   page =
ImageSnapshot.captureBitmapData(view.javaPhpComp.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	else if (view.currentState == "javaPhpXml"){
	 page =
ImageSnapshot.captureBitmapData(view.javaXmlCompar.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	   Alert.show("PDF page added!!");
}
```

U telu metode onAddPdfTextHandler, deklariše se i kreira lokalna promenljiva pdfObject. pdfObject je jedan generički i dinamički, privremeni objekat tipa Object, kojim se dodeljuju dva polja. Privremeni, zato što se nalazi unutar funkcije, i živi unutar funkcije. Naredbom pdfObject.text = event.currentTarget.text; vrši se upis podataka iz osobine text, objekta pdf koji je tipa AddTextPdfComponent, u osobinu text objekta pdfObject. Objekat koji je trigerovao

događaj event.currentTarget zove se pdf. event.currentTarget je objekat pdf klase AddTextPdfComponent. Naime, događaj je dodeljen pdf-u, t.j. instanci klase AddTextPdfComponent. U bloku koda:

```
if (!pages)
    pages = new Array();
```

proverava se da li je niz pages prazan, ako jeste, onda se alocira memorija za niz. Zatim se deklariše lokalna promenljiva page tipa BitmapData. BitmapData je jedan objekat koji dopušta da se radi sa pikselima[3], naime, page sadrži podatke koji su bitmapirani, što znači da pampti sliku po pikselima.

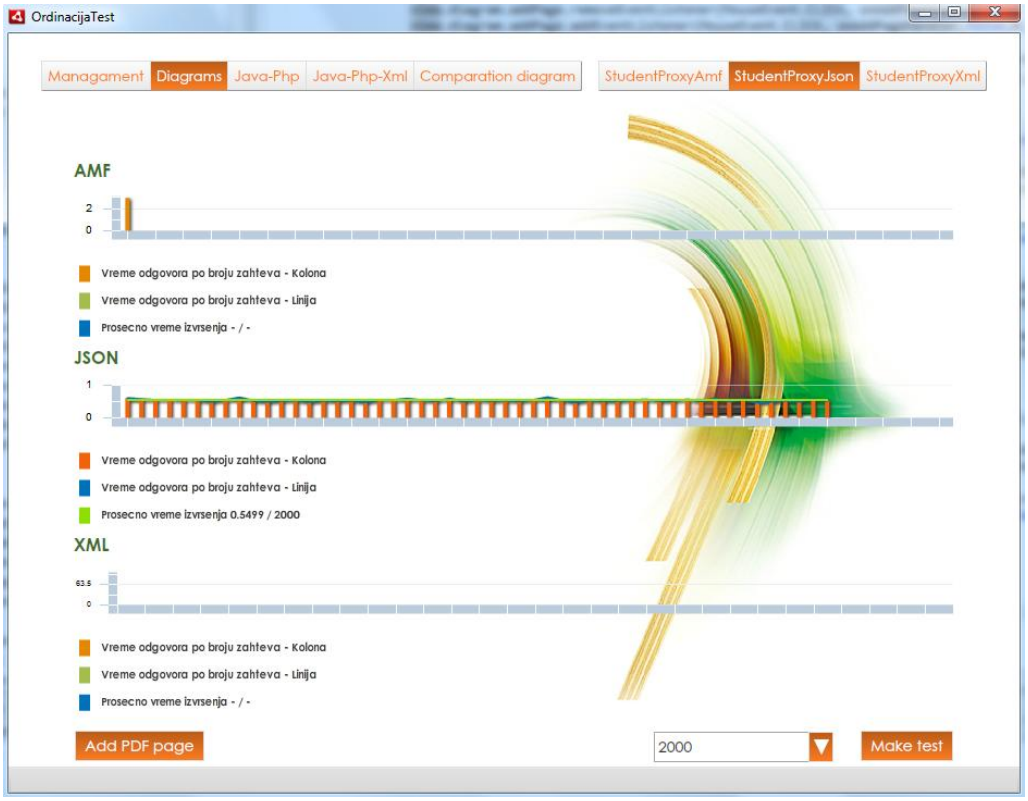
Prema tome, u promenljivoj page svaki piksel se pamti ponaosob. U ovoj metodi se proverava, koji deo aplikacije je aktivan. Naredbom **if** (view.currentState == "**diagrams**") proverava se da li je aplikacija u stanju diagrams, naredna slika. Ako jeste, onda se pristupa slikanju željenog prostora u aplikaciji.

Ovde se slika prostor gde se nalaze tri grafikona, i to se snima u promenljivu page. U narednom bloku koda:

```
page = ImageSnapshot.captureBitmapData(view.diagram.captureArea);
pdfObject.page = page;
pages.push(pdfObject);
```

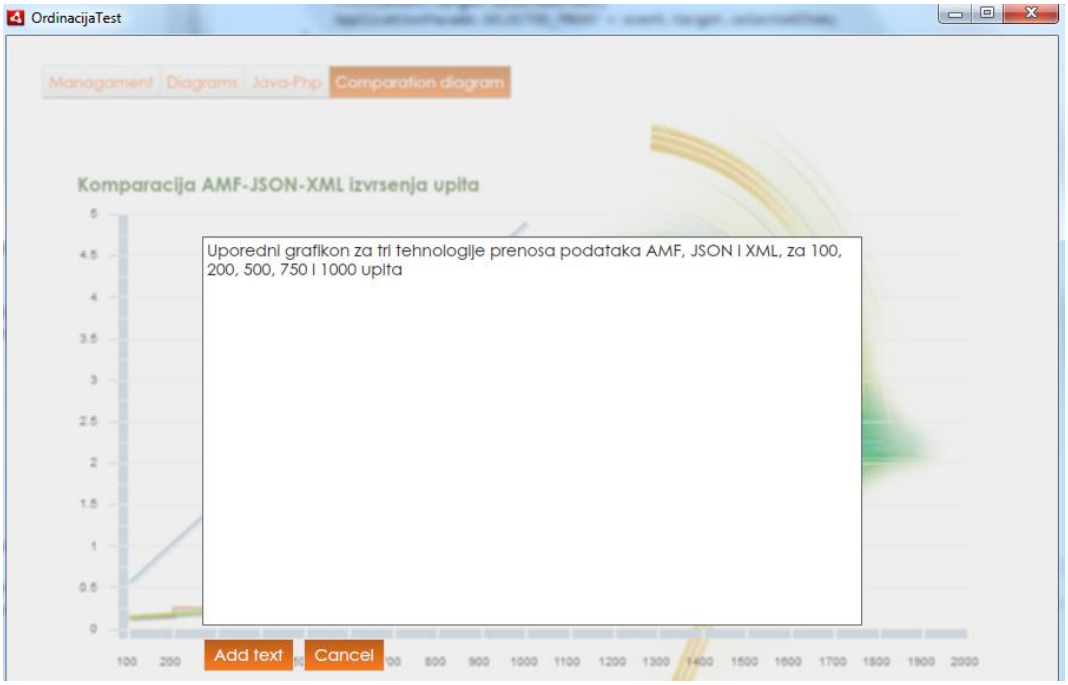
za slikanje prostora koristi se klasa ImageSnapsho. ImageSnapshot je helper klasa, koja uzima trenutnu sliku, bilo koje fleš komponente, koja u sebe implementira interfejs komponente IBitmapDrawable[3].

Znači bilo koja vizuelna grafička komponenta. captureBitmapData je statička metoda koja uzima BitmapData.



Slika3.7.28. Prikaz niza učitavanja za json opciju.

Kao parametar u metodi captureBitmapData se navodi šta tačno se želi slikati, u ovom slučaju to je view.diagram.captureArea. captureArea je VGroup u klasi ResultDiagram, koja ima sve dijagrame. Kada se dobije BitmapData t.j. page, onda u array se smesti page. Kada se dodaju sve stranice, onda postoji niz pages, u koji se nalaze sve stranice u obliku BitmapData.



Slika 3.7.29. Prikaz iskačućeg prozora, koj služi za registrovanje komentara ispod slike.

Kada se klikne na dugme Make pdf, onda se poziva metoda onCreatePDFHandler u klasi AplicationMediator.as. U ovoj metodi pravi se novi objekat pdf, tipa MSPDF. MSPDF je pdf klasa. Kao parametar pri instanciranju objekta klase MSPDF, prosleđuje se orijentacija stranice Orientation.PORTRAIT, jedinice koje se primenjuju na stranici Unit.POINT, dok setovanjem parametra autoPageBreak na true, t.j. autoPageBreak:Boolean=true, znači da se implicitno određuje gde se završava stranica, i format stranice je pageSize=Size.A4. Zatim se dodaje na pdf addEventListener događaj ProcessingEvent.COMPLETE. ProcessingEvent.COMPLETE je ivent koji ima klasa MSPDF. Kada završi kreiranje pdf dokumenta, poziva se funkcija _onCompleteHadler. Funkcija _onCompleteHadler daje obaveštenje da da je pdf kreiran. U delu koda:

```
            if (!pages)
            return;
```

Proverava se da li je u niz pages stavljena bar jedna stranica, ako ne sadrži ni jednu stranicu, onda naredbom return, znači vrati. Ako postoje stranice u pages, onda u for petlji se prolazi kroz sve stranice. Naredbom pdf.addPage(); objekat pdf dodaje ili ubacuje novu stranicu.

```
private function onCreatePDFHandler(evt:MouseEvent):void
{
      pdf = new MSPDF(Orientation.PORTRAIT,Unit.POINT,true,Size.A4);
      pdf.addEventListener(ProcessingEvent.COMPLETE,_onCompleteHadler);
            if (!pages)
                  return;
  for (var i:int=0;i<pages.length;i++){
      pdf.addPage();
pdf.addImageStream( PNGEncoder.encode(pages[i].page),
ColorSpace.DEVICE_RGB, null, PAGE_X, PAGE_Y, PAGE_WIDTH, PAGE_HEIGHT,0,1 );
```

U kodu, pdf.addImageStream je funkcija koja služi da napravi stranicu od BitmapData. Kao prvi parametar je ByteArray. pages su BitmapData. Funkcija encode pretvara BitmapData u ByteArray. Dakle, addImageStream je funkcija koja od ByteArray t.j. od niza bajtova, pravi stranicu. Jedna stranica page je jedan BitmapData. Naredbom PNGEncoder.encode(pages[i].page) koduje se svaka stranica pojedinačno u bajtove. Drugi parametar colorSpace=ColorSpace.DEVICE_RGB određuje u kom formatu se snima, znači RGB mod za ekran se koristi, Treći parametar je resizeMode:Resize. On je null, znači nema resize, pozicije stranice na ekranu x:Number i y:Number, date su kao konstante, koje su

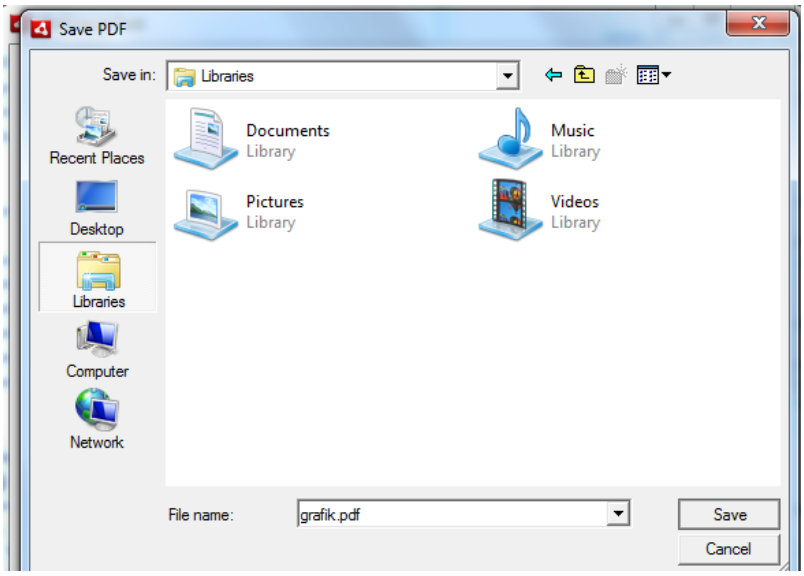
definisane na početku klase, širina i visina stranice width:Number, height:Number, da li je prisutna rotacija rotation:Number, rotacija nula, znači ne može se rotirati pdf stranica, transparentnost alpha:Number=1, što znači da je vidljivost 1, blendMode govori kako da se uklopi sa pozadinom blendMode:String="Normal", link:ILink=null znači, kada smo dodali sve stranice u pdf-u. Metoda setXY, je metod koja setuje x i y u isto vreme. Na mestu gde su postavljene koordinate x i y postavlja se tekst, naredbom pdf.writeText(10,pages[i].text); Razmak između redova je 10.

```
 pdf.setXY(20, 450);

        pdf.writeText(10,pages[i].text);
}
      file = new File();
      file.addEventListener(Event.SELECT, onFileSelectHandler);
      file.browseForSave("Save PDF");
}
```

U narednom delu koda *otvoren je novi fajl u RAM memoriji.*

```
file = new File();
```

File je fleksova klasa za manipulaciju sa fajl sistemom na računaru, dakle, povezuje fajl sistem sa AS3 jezikom[1]. U kodu file.browseForSave("Save PDF"); sa browseForSave funkcijom se dozvoljava korisniku da odabere ime fajla i lokaciju na hard disku[1]. Dakle, otvara se prozor kojim korisnik bira tačnu lokaciju na hard disku, naredna slika.



Slika 3.7.30. Odabir mesta na računaru, za skladištenje pdf fajla

Prozor je helper za odabir lokacije na hard disku. Kada se izabere lokacija na računaru, prozor se zatvara pritiskom na dugme Save, i tada se dispečuje event Event.SELECT. Naime, Event.SELECT je događaj, koji se dispečuje, kada se klikne dugme Save u browser prozoru, onda se poziva funkcija onFileSelectHandler, koja vrši snimanje:

```
private function onFileSelectHandler(event:Event):void
{
	var f:FileStream = new FileStream();

	f.open( file, FileMode.WRITE);

	var bytes:ByteArray = pdf.save( Method.LOCAL);
	f.writeBytes(bytes); //ovde se upisuju bajtovi
	f.close();
}
```

U telu metode deklarisana je lokalna promenljiva f tipa FileStream i alociran je memorijski prostor, naredbom **var** f:FileStream = **new** FileStream();. FileStream je klasa, koja služi za otvaranje putanje do fajla. U delu koda koda f.open(file, FileMode.WRITE); otvara se fajl, daje se ime fajlu koji se otvara na hard disku. Sada fajl ima tačno ime i lokaciju. Fajl se otvara u modu FileMode.WRITE, radi pisanja.

Naredbom **var** bytes:ByteArray = pdf.save(Method.LOCAL); pravi se ByteArray od pdf-a koji je kreran u metodi onCreatePDFHandler. Naime, u metodi onCreatePDFHandler napravljen je novi objekat pdf i u njega su ubačene sve stranice. objekat pdf se nalazi u radnoj memoriji. Instanca pdf, koristi metod save, da snimi podatke na lokalni disk i taj metod pretvara pdf u ByteArray, dakle, u seriju bajtova i onda na kraju se ti bajtovi smeste u FileStream, koji je selektovao fajl i u taj fajl se upisuje serija bajtova. Na kraju kada se sve zapiše, zatvori se fajl stream naredbom f.close();, da bi se oslobodila RAM memorija.

Ovde su bitne stvari pdf i file. Suština je da se sadržaj iz pdf upiše u file. Korisniku se dopušta da odabere lokaciju na računaru I ime fajla, I kada izabere ova dva parametra, onda se jednostavno otvara putanja do fajla na hard disku. Pošto je poznata lokacija fajla, kada se otvori, upisuju se bajtovi.

3.7.6 Mediatori

Mediatori upravljaju grafičkim komponentama. Kada se pokrene aplikacija prvo se registruju svi proksiji, pa onda mediatori. Mediator se pokreće onog trenutka kada se registruje. Registracija mediatora vrši se u kontroler klasi ViewStartupCommand. Aplikacija sadrži četiri mediatora ApplicationMediator, StudentControlMediator, StudentManagerMediator i StudentTableMediator.

3.7.7 Mediator ApplicationMediator.ac

Ovaj mediator upravlja grafičkom komponentom StudentAMF.mxml. Registruje se pri pokretanju aplikacije u klasi ViewStartupCommand, linijom koda:

```
facade.registerMediator(new ApplicationMediator(C));
```

Kada se kaže new ApplicationMediator(app), to znači poziva se konstruktor funkcija unutar klase mediatora.

```
public function ApplicationMediator(_viewComponent:Object)
{
      super(NAME, _viewComponent);
}
```

Konstruktor funkciji prosleđuje se objekat _viewComponent:Object. To je objekat sa kojim se radi, t.j. vizuelni. U konstruktoru se poziva super funkcija, konstruktor osnovne klase kojem se prosleđuje ime instance ovog medijatora **"ApplicationMediator"**, a to je jedinstveno ime instance ovog mediatora i prosleđuje mu se komponenta _viewComponent sa kojom će instanca mediatora da komunicira. Dakle, parametar _viewComponent ima identično ime kao ugrađena promenljiva viewComponent. Ime grafičke promenljive _viewComponent može biti i drugačije.

```
public function ApplicationMediator(_viewComponent:Object)
{
      super(NAME, _viewComponent);
}
```

Geter metoda view, vraća grafičku komponentu StudentAMF.

```
public functionget view():StudentAMF
{
      return viewComponent as StudentAMF;
}
```

Promenljive viewComponent i _viewComponent nisu iste. Ugrađena promenljiva viewComponent, po deklaraciji, i pripadnosti paketu, pripada Mediatoru, što se vidi iz deklaracije *viewComponent:Object- org.puremvc.as3.multicore. patterns.Mediator*. ***Mediator***. To znači da pozivom konstruktor funkcije osnovne klase **super**(NAME, _viewComponent); prosleđuje se komponenta _viewComponent, promenljivoj viewComponent u osnovnoj klasi, dakle, to uradi konstruktor. To znači da komponenta viewComponent ima vrednost objekta _viewComponent. U kodu se koristi **get** view zbog kastovanja, jer je viewComponent tipa Object. Klasa AplicationMediator sadrži sedam osobina, jednu javnu statičku konstantu, šest privatnih konstanti i šesnaest metoda.

Osobine i konstante su:

```
public static const NAME:String = "ApplicationMediator";
private var steper:int = 1;
private var _proxy:*;
/*Po difoltu, boolean promenljiva ima vrednost false*/
private var isTest:Boolean;
private var controlMediator:StudentControlMediator;
private var pdf:MSPDF;
private const PAGE_X:Number = 10;
private const PAGE_Y:Number = 10;
private const PAGE_WIDTH:Number = 450;
private const PAGE_HEIGHT:Number = 400;
private const PAGE_DPI : Number = 150;
private const PAGE_QUALITY : Number = 95;
private var file:File;
private var pages:Array;
```

Metode ove klase su:

- *Konstruktor funkcija ApplicationMediator*
- *Geter osobina **get** view*
- *onRegister*

- *onStateChangeHandler*
- *numOfObjects_changeHandler*
- *onRemove*
- *onSelectedProxyHandler*
- *onTestHandler*
- *onAddPageHandler*
- *onAddPdfTextHandler*
- *_onCompleteHadler*
- *onCreatePDFHandler*
- *onFileSelectHandler*
- *listNotificationInterests*
- *handleNotification*
- ***get** proxy*

Ugrađena metoda onRegister se poziva automatski, pri instanciranju i registraciji objekta mediatora. Prvo se izvrši super funkcija pa tek onda se registruje. Ova funkcija se pokrene na registraciju. Sve što je urađeno u onRegister funkciji, može da se uradi u konstruktoru, ali se to ne preporučuje, onda bi se dobio veći fajl i sporije bi radila aplikacija. Sve se radi u onRegister iz prostog razloga da se ne opterećuje konstruktor previše. Pusti se konstruktor da što pre završi svoj posao. On Register je ugrađen u klasu medijator.

U onRegister registruju se onListener-i, npr next je button. Button-u se može dodeliti event listener na dva načina, prvi način je što će mu se odmah dodeliti funkcija na klik(u mxml-u, kada se kreira dugme), a drugi način je da se dodeli iventListener kroz ActionScript kod. Ovde se koristi drugi slučaj, dakle, daje se iventListener kroz ActinScript kod. Zašto mu dajemo kroz Action Script kod? Prva i osnovna stvar je kada se nekoj komponenti daje iventListener direktno iz mxml-a taj listener nikad više ne može da se ukine. Ako mu se dade kroz ActionScript, onda može da se ugasi, to je prva stvar, druga stvar je ako bi se dao ivent listener u definiciji dugmeta:

```
<s:Button
      id="next"
      label="Next"
      />
```

onda bi se funkcija nalazila u ovoj klasi, koja se zove id="next", next je naziv klase, kada se ovaj blok koda pretvori u AS3. A onda bi moralo da se pristupa toj klasi iz medijatora kroz view. Ovde je izabran drugi način, celokupna logika je napravljena u mediatoru a kroz mediator je dodeljen samo ivent listenere. Na ovaj način je čistiji kod.
mxml komponente su po pravilu mnogo sporije nego mxml klase, zato što se deo koda u mxml-u:

```
<s:Button
      id="prev"
      label="Previous"
      />
```

mora prevesti u AS3, pa se posle izvršava. Ove komponente (button i druge) treba da budu što lakše. A sama reč MVC znači da se razbacuje po slojevima logika aplikacije. Logika view definiše izgled komponente. Dakle, logika view prikazuje korisniku izgled aplikacije i prihvata interakciju korisnika, znači kada se klikne view prihvati to i prikaže.
Logika proračuna ili manipulacije nečega treba da se nalazi u drugom sloju(leru) i zato mediator služi. Baratanje sa podacima ide u proksi, naime, pozivanje http requesta, pozivanje baze. Komplikovani proračuni idu u proksi , kao i priprema podataka, a onda se vrate u mediator i to je najbolja logika podele MVC-a. Kontroleri treba da budu okidači ili trigeri komandi. Svaki kontroler je samo jedna komanda. Dakle, mediator prima i šalje notifikacije. Komande imaju i šalju notifikacije a proksi samo šalje notifikacije. Kontroleri reaguju, oni se pale samo na notifikacije. U kontroleru pored što se izvrši može i da se pošalje notifikacija, naime, znači jedno i drugo. Medijator radi to isto, može da pošalje i primi notifikaciju. Ovo je lista notifikacija sa kojim će medijator raditi:

```
override public function listNotificationInterests():Array
{
      return [
            ApplicationFacade.STUDENT_LOADED
      ]
}
```

handleNotification je override funkcija koja služi za obradu tih notifikacija:

```
override public function
handleNotification(notification:INotification):void
{
      switch(notification.getName()){
```

```
            case ApplicationFacade.STUDENT_LOADED:
                  if (isTest == false) break;
                  if (steper <= 50){
                        steper++;

proxy.getAllStudentiFlex(controlMediator.currentPage);
}
                  . . .
}
```

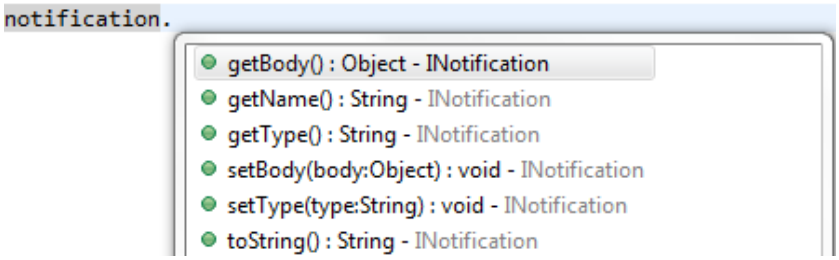
Dakle, override metod listNotificationInterests prihvata notifikacije, override handleNotification ih posle obrađuje. Stoga, mediatori mogu i da šalju notifikacije, što se vidi u narednom kodu:

```
protected function medicineGrid_clickHandler(event:MouseEvent):void
{
if (event.target is DefaultGridItemRenderer){
      view.selectedStudent =  view.medicineGrid.selectedItem as StudentVO;
sendNotification(ApplicationFacade.STUDENT_SELECTED,view.selectedStudent);
      }
}
```

Kod proksija je sasvim drugačija situacija, proksi nemaju ugrađenu funkciju listNotification, ne mogu da primaju notifikaciju, već samo šalju, što je i opravdano, jer proksiju se obraća direktno preko proksija, zove se njegov metod. On kada obradi neki podatak, on ga pošalje, znači vratio ga kroz samu notification. Nema potrebe da prihvati notifikaciju.
Svaki medijator ima ugrađenu funkciju koja se zove listNotificationInterests. Ova funkcija ne radi ništa drugo, već vraća jedan Array. Ovde se registruje lista notifikacija za koje je ovaj mediator zadužen. Dakle, u ovoj funkciji se daje lista notifikacija kroz jedan Array. Praktično, registrovan je ovaj mediator da sluša ovaj događaj.

```
override public function listNotificationInterests():Array
{
      return [
            ApplicationFacade.STUDENT_LOADED
      ]
}
```

Kada se desi ova notifikacija, onda se se pokreće funkcija, koja se zove handleNotification**.** Jedna i druga funkcija je override, što **z**nači da su ugrađene, i ovde se prepišu, a algoritam tela funkcije prilagodi se potrebama, dakle, piše se drugi algoritam, koji definiše ono čime hoćemo da se služimo. Svaka notifikacija je tipa INotification, znači klasa notification primenjuje interfejs INotification[1,3]. Svaka notifikacija kao objekat poseduje tri bitne informacije Body, Name i Type.



Slika 3.7.31. Meni za izbor getBody() kod notifikacije.

Tip notifikacije može programmer da definiše sam kao ime notifikacije i kao Body notifikacije. Dakle, ostavljeno je da se tip notifikacije može definisati posebno. Naime, programer može, naročito kada radi sa multicore sistemom, t.j. kada radi sa više notifikacija i kada ima više aplikacija sa kojim može da se radi odjednom, može se raditi sa istom notifikacijom ali drugog tipa i onda se određuje u zavisnosti od tipa, šta treba uraditi.
Tip se često koristi kod slučaja, kada se pošalje ista notifikaciju na više mesta, a po različitosti tipa se odreaguje na određenim mestima.

Ako se uzme event STUDENT_DELETED, koji je registrovan za slušanje na dva mesta, u studentTableMedijator i sudentManagerMedijator, I pri tome se pošalje ovaj događaj STUDENT_DELETED, u nekim slučajevima je bitno da jedan medijator ne odreaguje. STUDENT_DELETED notifikacija može se slati više puta sa više mesta. U nekim slučajevima nije potrebna ova notifikacija. Definišući tip posebno, definiše se I da li datu notifikaciju treba obraditi ili ne.
U liniji koda sendNotification(ApplicationFacade.STUDENT_DELETED); u sendNotification, na prvo mesto je ime, na drugo mesto je object, telo, a na treće mesto je tip. Ovde je dato samo ime notifikacije, događaja. Sada se može definisati i tip. Neka se tip zove "manager". U narednoj liniji koda, notifikacija poseduje, ime, telo i tip.

```
sendNotification(ApplicationFacade.STUDENT_DELETED,null,"manager");
```

Notifikacija se obrađuje u metodi handleNotification, klase studentManager:

```
override public function
handleNotification(notification:INotification):void {
switch(notification.getName()){
      case ApplicationFacade.STUDENT_SELECTED:
      view.selectedStudent = notification.getBody() as StudentVO;
            break;
      case ApplicationFacade.STUDENT_DELETED:
            if(notification.getType()=="manager")
                  trace("ovo je manager");
                              break;
      case ApplicationFacade.STUDENT_ADDED:
            if (newStudent){
            newStudent.id = notification.getBody() as int;
                        StudentListVO.StudentiArray.addItem(newStudent);
                  reset();
                              }
            break;
                  }
            }
```

Kada pokrenemo aplikaciju, posle brisanja sloga ispisaće poruku nadno:*Ovo je manager*
Ako je izvršena promena u proksiju, i sadržaj tipa se promeni, nije manager, već nešto drugo i ponovi se akcija, ispisaće.

```
private function deleteStudentiFlexResultHandler(event:ResultEvent):void{
removeListeners(deleteStudentiFlexResultHandler,deleteStudentiFlexFaultHand
ler);
      sendNotification(ApplicationFacade.STUDENT_DELETED,null,"brisi");

      trace("obrisan student")
}
```

Ne piše sada:

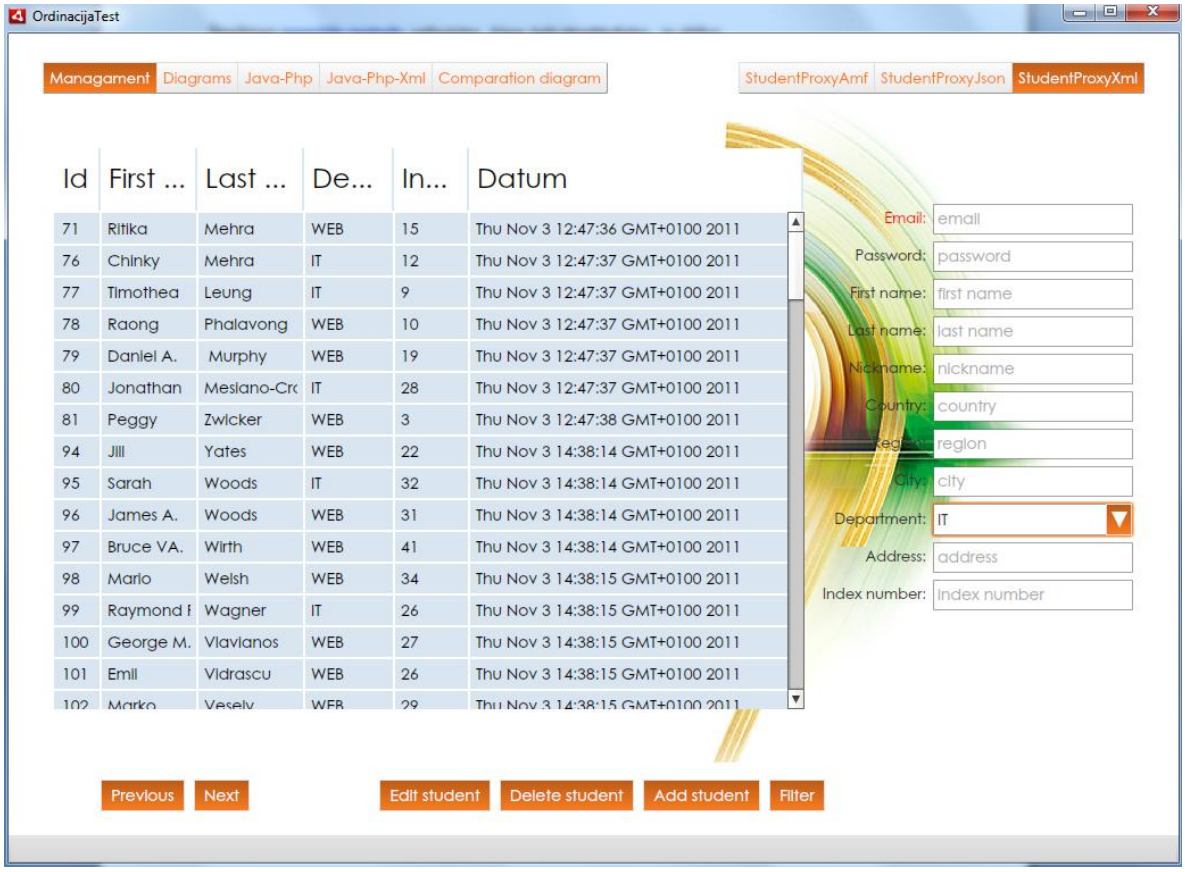
ovo je manager.

Struktura override metodeonRegister, klase AplicationMediator, je oblika:

```
override public function onRegister():void
{
```

```
controlMediator = facade.retrieveMediator(StudentControlMediator.NAME) as
StudentControlMediator;
view.addEventListener(FlexEvent.STATE_CHANGE_COMPLETE,onStateChangeHandler);
view.selectProxyBar.addEventListener(IndexChangeEvent.CHANGE,
onSelectedProxyHandler);
}
```

Ova funkcija se odmah poziva pri instanciranju i registraciji ovog medijatora. U prvoj liniji koda, unutar tela ove funkcije pronalazi se medijator iz liste sa imenom StudentControlMediator.NAME, i sada controlMediator ukazuje na taj medijator. U drugoj liniji koda dodaje se eventListener na grafičku komponentu view t.j. na objekat klase SudentAMF.mxml. Aplikacija je prikzana na slici 3.7.32.



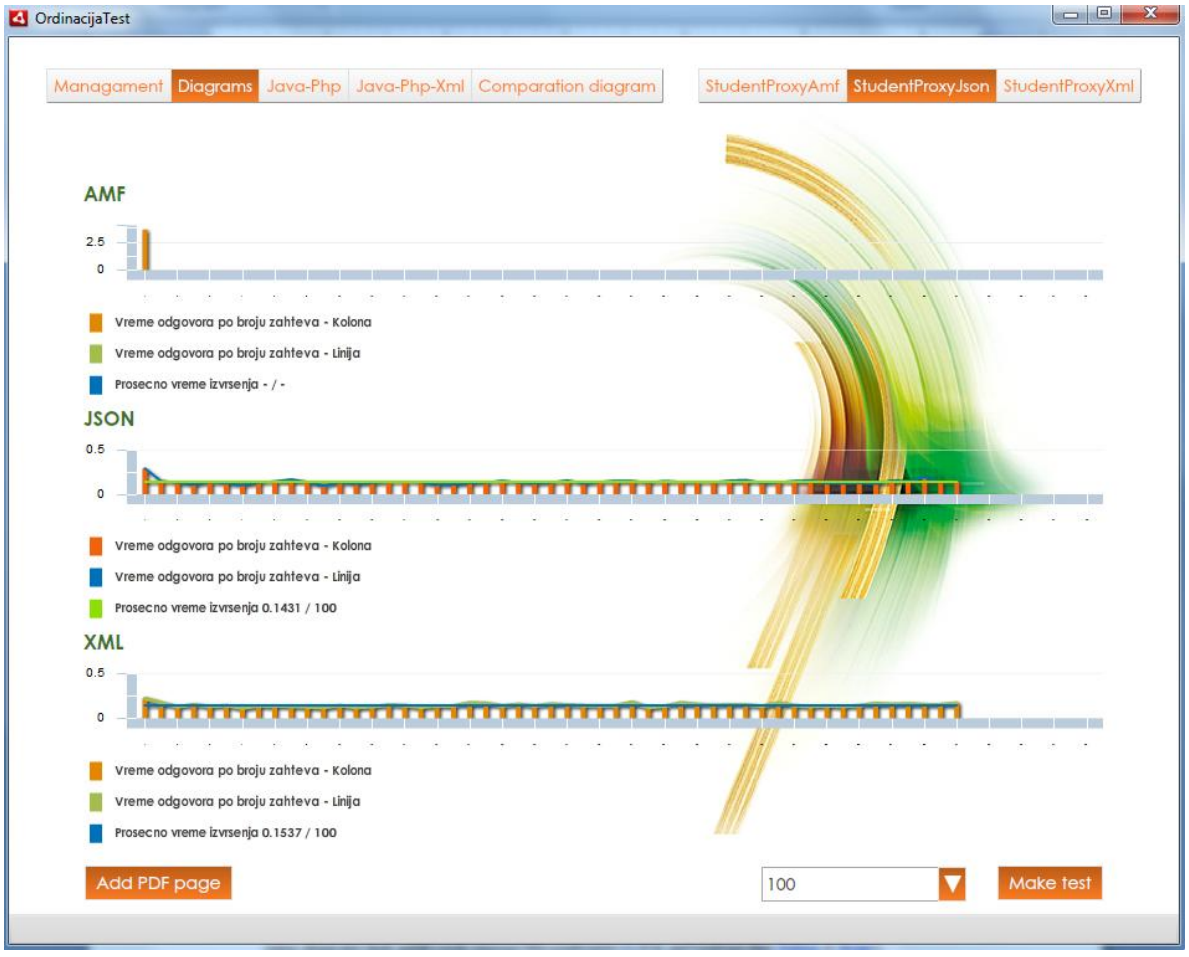
Slika 3.7.32. Početna strana aplikacije, kada su učitani podaci

Događaj FlexEvent.STATE_CHANGE_COMPLETE registruje bilo kakvu promenu na aplikaciji, i kada se završi promena onda se aktivira metoda onStateChangeHandler. U narednoj liniji koda dodaje se eventListener na SelectProxyBar koji predstavlja ButtonBar dugme, naredna slika:

Slika 3.7.33.ButtonBar dugme sa pet stanja

Događaj IndexChangeEvent.CHANGE registruje pritisak na ButtonBar selectProxyBar i kada se završi promena onda se aktivira metoda onSelectedProxyHandler.

Algoritam metode onStateChangeHandler, je opisan narednim linijama koda. Ova metoda se izvršava svaki put kada se izvrši promena stanja na aplikaciji. U if naredbi, poziva se geter view, koji vraća grafičku komponentu, u ovom slučaju to je instance klase StudentAMF. Pristupa se ugrađenoj promenljivoj currentState, i proverava se da li je trenutno stanje u aplikaciji "diagrams", ako jeste, to znači proverava se da li je komponenta diagram null, a null je ako nije aktivna, onda se izvršavaju naredbe unutar tela if bloka. Aplikacija u stanju "diagrams", je kao na slici3.7.34.



Slika 3.7.34. Prikaz tri grafikona za amf, json i xml tehnologiju.

Naredni kod, ukazuje na veze ka drugim objektima iz medijatora.

```
private function onStateChangeHandler(event:FlexEvent):void
{
 if (view.currentState == "diagrams"){
    view.diagram.test.removeEventListener(MouseEvent.CLICK, onTestHandler);
      view.diagram.test.addEventListener(MouseEvent.CLICK,onTestHandler,fal
      se,0,true);
      view.diagram.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
      view.diagram.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
view.diagram.numOfObjects.removeEventListener(IndexChangeEvent.CHANGE,
numOfObjects_changeHandler);
view.diagram.numOfObjects.addEventListener(IndexChangeEvent.CHANGE,
numOfObjects_changeHandler,false,0,true);
}
```

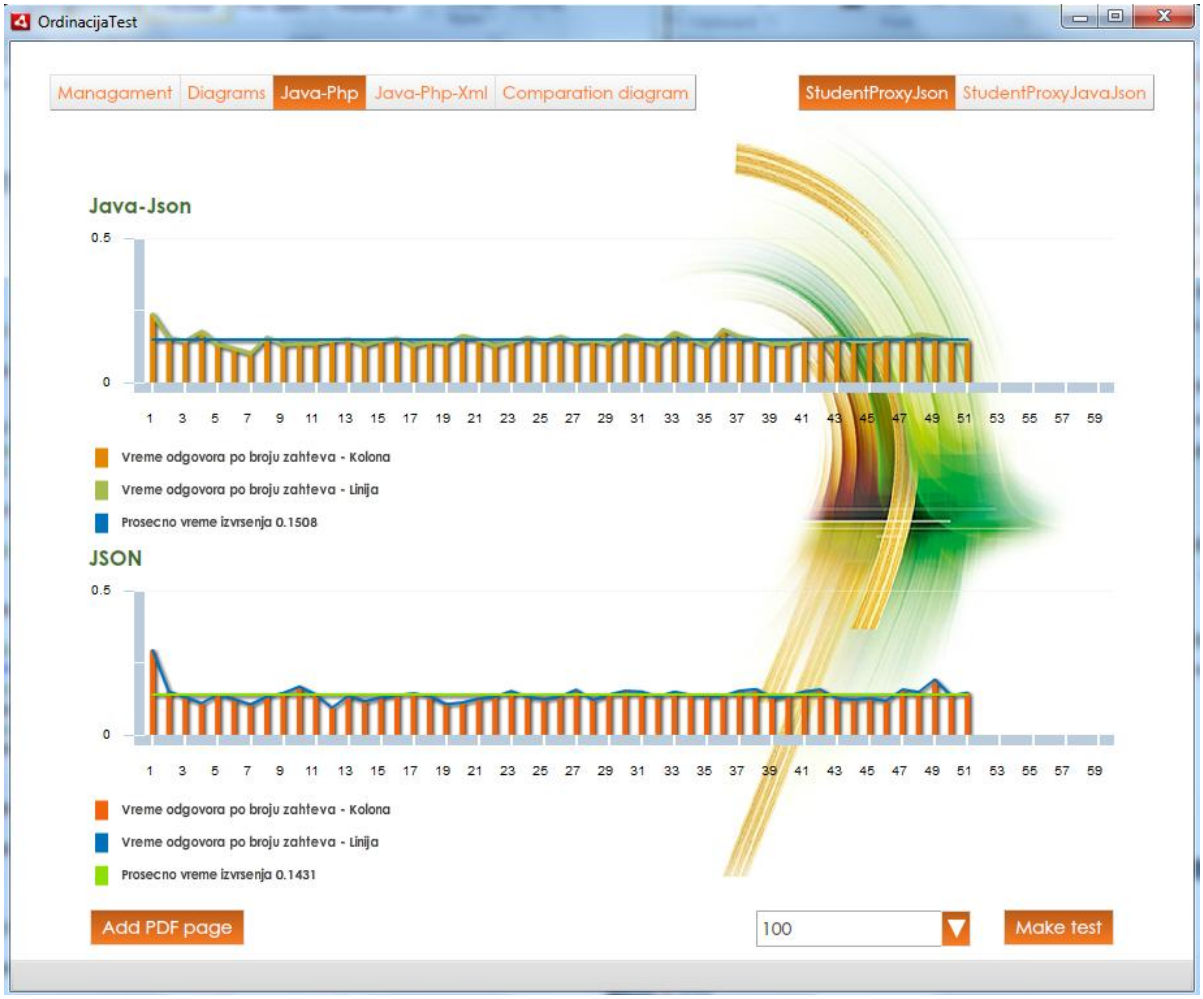
Dakle, ako je view.currentState == "diagrams" onda se pristupa osobinama komponente diagram. Uprvoj liniji koda, unutar if bloka, view.diagram.test osobini komponente diagram se uklanja listener u slučaju da se klikne na njega. U drugoj liniji koda, if bloka, view.diagram.test osobini komponente diagram se dodaje listener u slučaju da se klikne na njega, i svaki put priovom događaju poziva se metoda onTestHandler. U trećoj liniji koda view.diagram.addPage osobini komponente diagram se uklanja listener u slučaju da se klikne na njega, dok se u četvrtoj liniji koda osobini view.diagram.addPage komponente diagram, dodaje listener u slučaju da se klikne na njega, i poziva se metod onAddPageHandler. U petoj liniji koda, kada je diagram komponenta aktivna, prvo se uklanja listener sa njene osobine view.diagram.numOfObjects a onda se u narednom koraku dodaje listener za događaj IndexChangeEvent. Događaj IndexChangeEvent je trigerovan, kada se promeni selekcija u kompo boksu view.diagram.numOfObjects. Tada se poziva funkcija 'numOfObjects_changeHandler' , koja obrađuje ovaj događaj.

U liniji koda if (view.currentState == "javaPhp") proverava se da li je trenutno stanje u glavnoj aplikaciji "javaPhp". Ako nije, onda je komponentajavaPhpComp null, a null je ako nije aktivna. Ako je zadovoljen uslov, onda je komponenta javaPhpComp aktivna i pristupa se osobinama komponente javaPhpComp. U prvoj liniji koda, unutar if bloka, view.javaPhpComp.test osobini komponente javaPhpComp se uklanja listener u slučaju da se klikne na njega. U drugoj liniji koda istoj osobini se dodeljuje eventListener u slučaju da se klikne na njega, i tada se poziva metoda onTestHandler, koja obrađuje ovakav događaj. U trećoj

liniji koda, uklanja se eventListener, u slučaju klika na kombo bix view.javaPhpComp.numOfObjects, a u narednoj liniji koda u slučaju klka mu se dodeljuje listener, i poziva se metoda numOfObjects_changeHandler za obradu događaja. U petoj liniji koda, unutar if bloka, prvo se uklanja a onda dodaje event listener u narednoj liniji koda osobini view.javaPhpComp.addPage u slučaju da se klikne na njega, Ii poziva se metoda onAddPageHandler.

```
else if (view.currentState == "javaPhp"){
view.javaPhpComp.test.removeEventListener(MouseEvent.CLICK, onTestHandler);
view.javaPhpComp.test.addEventListener(MouseEvent.CLICK,
onTestHandler,false,0,true);
view.javaPhpComp.numOfObjects.removeEventListener(IndexChangeEvent.CHANGE,
numOfObjects_changeHandler);
view.javaPhpComp.numOfObjects.addEventListener(IndexChangeEvent.CHANGE,
numOfObjects_changeHandler,false,0,true);
view.javaPhpComp.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.javaPhpComp.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
}
```

Aplikacija je prikazana na slici 3.7.35., kada je aktivna komponenta javaPhpComp.



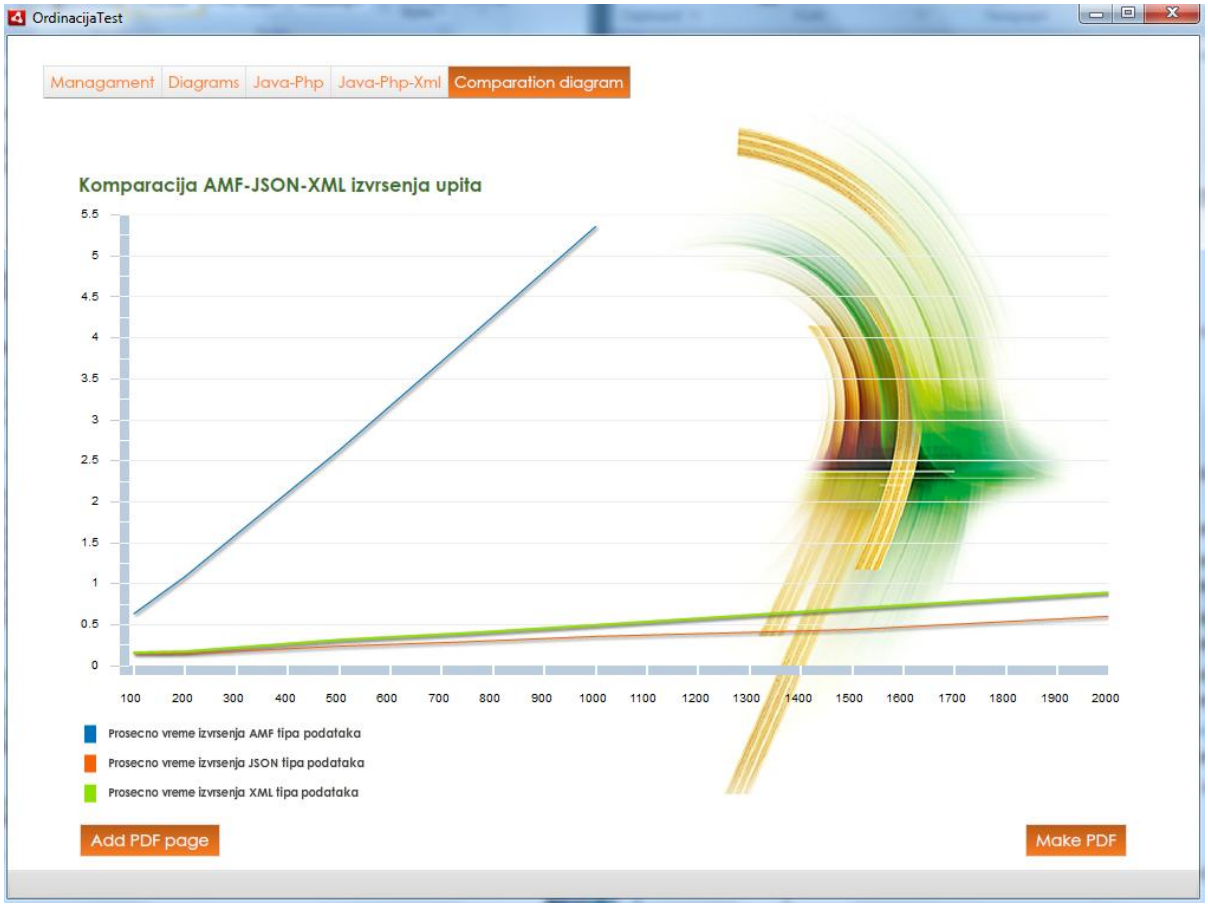
Slika 3.7.35. Prikaz dva grafikona kada sekoristi Java server i Wamp server

Ako je uključeno stanje "comparation" u aplikaciji, onda je komponenta comparationDiagram različita od null, a različita je od null ako je aktivna i pristupa se osobinama komponente comparationDiagram. U if bloku, osobinama view.comparationDiagram.createPDF, view.comparationDiagram.addPage komponente comparationDiagram, prvo se uklanja listener, a potom se dodaje listener u slučaju da se klikne na njih, i poziva se metoda onCreatePDFHandler za prvi sluča, i onAddPageHandler za drugi slučaj.

```
else if (view.currentState == "comparation"){

view.comparationDiagram.createPDF.removeEventListener(MouseEvent.CLICK,
onCreatePDFHandler);
view.comparationDiagram.createPDF.addEventListener(MouseEvent.CLICK,
onCreatePDFHandler,false,0,true);
view.comparationDiagram.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
```

```
view.comparationDiagram.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
}
```

Aplikacija za slučaj, da je komponenta comparationDiagram aktivna, ima izgled:



Slika 3.7.36. Komparacioni grafikon za amf, json i xml tehnologiju.

U if naredbi (view.currentState == "javaPhpXml"), proverava se da li je aktivno stanje "javaPhpXml" u aplikaciji, odnosno da li je aktivna kopmonenta javaXmlCompar. Ako je view.javaXmlCompar različito od null onda se pristupa osobinama komponentejavaXmlCompar. U ovom if bloku, dodaje se listener osobinama:

```
view.javaXmlCompar.test,
view.javaXmlCompar.numOfObjects,
view.javaXmlCompar.addPage
```

komponente javaXmlCompar.

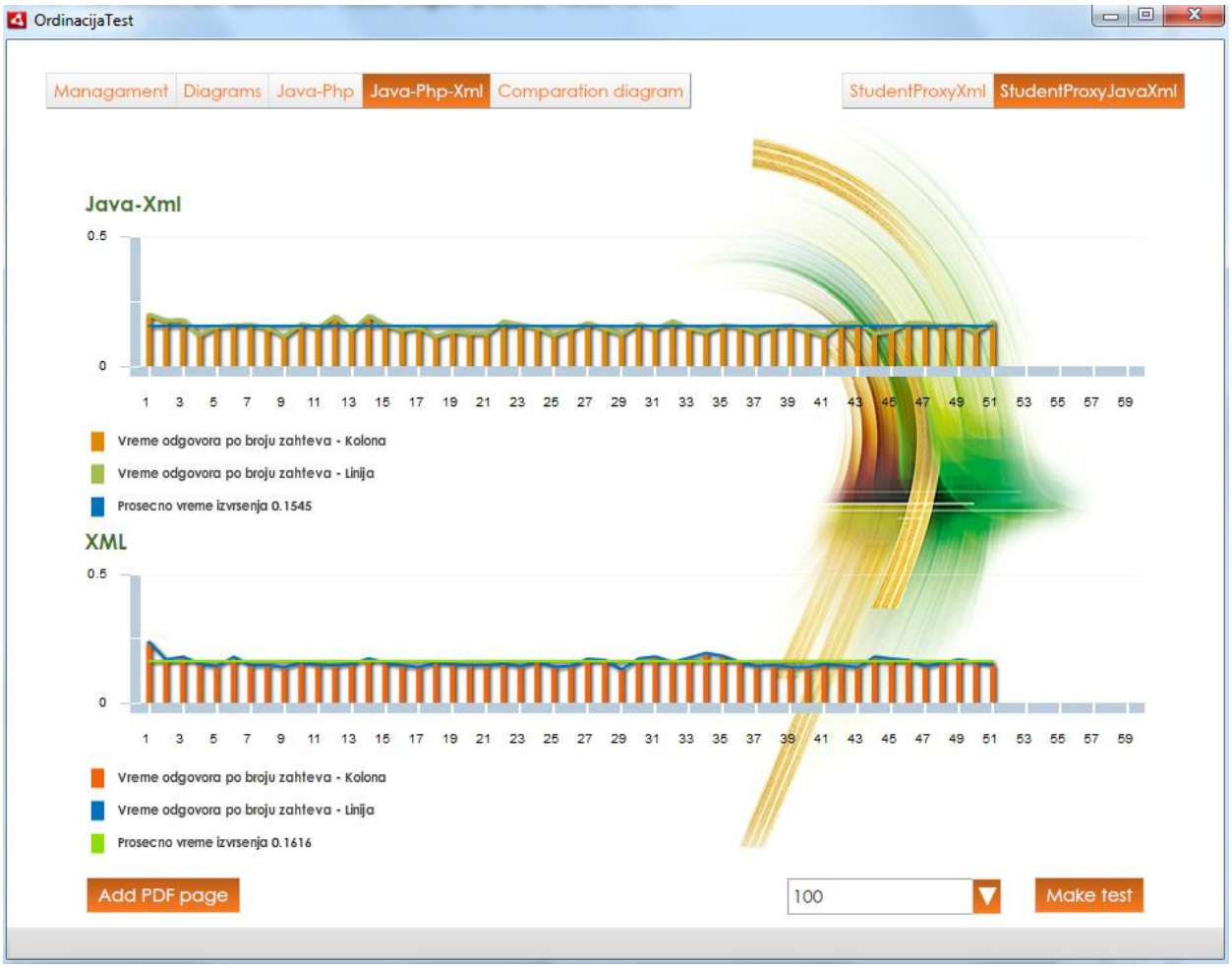
```
else
if (view.currentState == "javaPhpXml"){
```

```
view.javaXmlCompar.test.removeEventListener(MouseEvent.CLICK,
onTestHandler);
view.javaXmlCompar.test.addEventListener(MouseEvent.CLICK,
onTestHandler,false,0,true);

view.javaXmlCompar.numOfObjects.removeEventListener(IndexChangeEvent.CHANGE,
 numOfObjects_changeHandler);
view.javaXmlCompar.numOfObjects.addEventListener(IndexChangeEvent.CHANGE,
numOfObjects_changeHandler,false,0,true);
view.javaXmlCompar.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.javaXmlCompar.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);
```

Kada neko prvi put klikne na stavku javaXmlCompar, selektovani proksi će biti StudentProxyXml.NAME

```
ApplicationFacade.SELECTED_PROXY = StudentProxyXml.NAME;
      }
}
```

Aplikacija, u stanju "javaPhpXml", izgleda kao na slici 3.7.37.



Slika 3.7.37. Prikaz grafikona u kada je serverski jezik Java i Php.

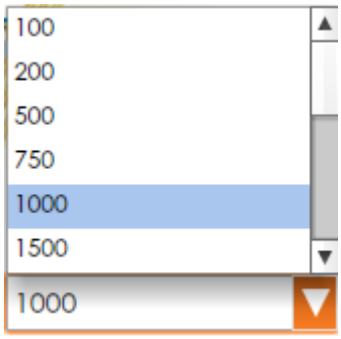
U funkciji *numOfObjects_changeHandler* se kaže, statička varijabla 'StudentListVO.requestNumber' je jednaka ono što je selektovano iz kombo boksa. Dakle, promene se registruju u promenljivu requestNumber, koja se deklariše, linijom koda:

```
[Bindable]
public static var requestNumber:int = 500;
```

pošto je requestNumber, statička varijabla i bindebl, ona se primenjuje svuda gde se poziva, u svim kalkulacijama i svim prikazima.

```
protected function numOfObjects_changeHandler(event:IndexChangeEvent):void
{
	StudentListVO.requestNumber = event.currentTarget.selectedItem;
}
```

Znači, kada se promeni selekcija na osobini numOfObjects, poziva se ova funkcija. Promena selekcije u ComboBox-u je prikazana na slici 3.7.38.



Slika 3.7.38. ComboBox u aplikaciji

Metod addEventListener sadrži pet parametara i ima strukturu:

```
addEventListener (type:String, listener:Function, useCapture:Boolean=false,
priority:int=0, useWeakReference:Boolean=false):void
```

Prvi parameter *type:String* je obavezan, definiše tip eventa, drugi parameter *listener:Function*, je listener koji se poziva, kada se desi događaj, treći parameter *useCapture:Boolean=false*, je opcioni. useCapture je jedan od faza unutar samog događaja. Četvrti parameter *priority:int=0*, nije obavezan. Celobrojni parameter priority definiše prioriteta događaja. Ako se dešavaju u istom trenutku dva događaja prioritet određuje koji se prvo procesuira. Peti parameter *useWeakReference:Boolean=false*, je opcioni, logičkog je tipa, može da uzme dve vrednosti false i true[1,3]. useWeakReferencs po difoltu je false. Ako je useWeakReferencs false, znači koristi jake veze. Ake se želi slaba veza, dodeliće se parametru useWeakReferencs vrednost true. U slučaju da useWeakReferencs koristi slabu vezu, kada se objekat prvi put uništi, veća je verovatnoća da će ga skupljač smeća pokupiti što pre, ili i u slučaju grešaka programera veća je verovatnoća da će što pre on raskinuti veze sa objektima i da će biti odnet. U slučaju da useWeakReferencs koristi slabu vezu, kada se objekat prvi put uništi, veća je verovatnoća da će ga skupljač smeća pokupiti što pre, ili i u slučaju grešaka programera veća je verovatnoća da će što pre on raskinuti veze sa objektima i da će biti odnet. Ako se medijator briše, tada referenca je na null postavljena, međutim, ako nisu obrisane reference do drugih objekata iz medijatora, onda se neće u potpunosti obrisati instance medijatora. Ako se naredni put pokuša alocirati objekat za medijator sa isti imenom, neće se dobiti novi medijator, već će se ukazati na isti, koji nije ukinuo reference ka drugim objektima, a zbog toga što je brisan neće moći ništa da radi, što predstavlja problem.

Metod onRemove uklanja listener sa osobina view.diagram.test i view.selectProxyBar.

```
override public function onRemove():void
{
view.diagram.test.removeEventListener(MouseEvent.CLICK, onTestHandler);
view.selectProxyBar.removeEventListener(IndexChangeEvent.CHANGE,
onSelectedProxyHandler);
}
```

Metod onSelectedProxyHandler se poziva u slučaju da se klikne na osobinu view.selectProxyBar (slika 3.7.35.)

```
private function onSelectedProxyHandler(event:IndexChangeEvent):void
{
      trace(event.target.selectedItem);
      ApplicationFacade.SELECTED_PROXY = event.target.selectedItem;
}
```

U slučaju da se klikne na osobinu view.diagram.test komponente diagram, ili view.javaPhpComp.test komponente javaPhpComp ili na osobinu view.javaXmlCompar.test komponente javaXmlCompar poziva se metod onTestHandler. Na samom početku se postavlja globalna kontrolna promenljiva logičkog tipa isTest na true**.** Naime, kada se pokrene 50 zahteva pritiskom na dugme Make test, setuje se promenljiva isTest na true i to svaki put kada se pokrene test. Dakle, kada se traži prvih 50 loada, on postavlja isTest na true, i onda krene mehanizam da radi. If naredbom proverava se koji je proksi aktivan, a onda se alocira memorija za statiki niz tipa ArrayCollection, koji služi kao snabdevač podataka dataProvidera grafikona.

```
private function onTestHandler(evt:MouseEvent):void
{
      trace("selected proxy == "+ApplicationFacade.SELECTED_PROXY)
      isTest = true;
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyAmf.NAME)
            StudentListVO.amfDataProvider = new ArrayCollection();
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyJson.NAME)
            StudentListVO.jsonDataProvider = new ArrayCollection();
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyXml.NAME)
            StudentListVO.xmlDataProvider = new ArrayCollection();
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaJson.NAME)
            StudentListVO.jsonJavaDataProvider = new ArrayCollection();
```

```
        if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaXml.NAME)
              StudentListVO.xmlJavaDataProvider = new ArrayCollection();
        proxy.steper = 1;
        proxy.average = 0;
        proxy.getAllStudentiFlex(controlMediator.currentPage);
}
```

Niz amfDataProvider se koristi za amf varijantu, jsonDataProvider za json i xmlJavaDataProvider za xml slučaj. Sva tri niza imaju istu strukturu, pune se objektima tipa Object. Svaki objekt, koji predstavlja element niza, ima tri osobine callNumber, timeValue, average i requested. Niz jsonDataProvider ima oblik:

jsonDataProvider[0]		jsonDataProvider[1]		jsonDataProvider[2]			jsonDataProvider[51]	
callNumber	1	callNumber	2	callNumber	3	...	callNumber	51
timeValue	0.272	timeValue	0.13	timeValue	0.154		timeValue	0.144
average	0.1603	average	0.1603	average	0.1603		average	0.1603
requested	100	requested	100	requested	100		requested	100

Slika 3.7.39. Prikaz jsonDataProvider niza

Na kraju tela metode, unutar odabranog proksija, pristupa se javnoj promenljivoj steper i setuje se na 1, a average se setuje na nula, t.j. proxy.average=0, i onda se poziva metoda getAllStudentiFlex odabranog proksija, koja počinje učitavanje podataka iz baze. Metodi getAllStudentiFlex kao parameter se prosleđuje pozicija sloga u bazi odakle treba da započne čitanje podataka. Dakle, mehanizam radi, tako što traži getAllStudentiFlex od proksija, od json-a na primer traži studente, sa stranicom currentPage, sa nulom na primer. Od nula pa nadalje. Kao rezultat izvršenja dobija se ivent ApplicationFacade.STUDENT_LOADED u metodi handleNotification.

Metod onAddPageHandler se poziva, u slučaju da se klikne na osobinu view.diagram.addPage komponente diagram, view.javaPhpComp.addPage komponente javaPhpComp, view.comparationDiagram.addPage komponente comparationDiagram ili na view.javaXmlCompar.addPage komponente javaXmlCompar. Unutar ove metode se poziva statička metoda addText klase AddTextPdfComponent, i kao parameter joj se prosleđuje naziv metode onAddPdfTextHandler.

```
private function onAddPageHandler(evt:MouseEvent):void
{
        AddTextPdfComponent.addText(onAddPdfTextHandler);
}
```

Metod onAddPdfTextHandler dodaje sliku i tekst pdf stranici.

```
private function onAddPdfTextHandler(event:CloseEvent):void
{
	var pdfObject:Object = new Object();
	pdfObject.text = event.currentTarget.text;
	if (!pages)
		pages = new Array();
	var page:BitmapData;
//page je tipa BitmapData. To znači da se svaka tačka predstavlja na svoj
način
if (view.currentState == "diagrams"){
	page = ImageSnapshot.captureBitmapData(view.diagram.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
else if (view.currentState == "comparation"){
	page =
ImageSnapshot.captureBitmapData(view.comparationDiagram.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	else if (view.currentState == "javaPhp"){
	page = ImageSnapshot.captureBitmapData(view.javaPhpComp.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	else if (view.currentState == "javaPhpXml"){
page = ImageSnapshot.captureBitmapData(view.javaXmlCompar.captureArea);
		pdfObject.page = page;
		pages.push(pdfObject);
	}
	Alert.show("PDF page added!!");
}
```

Metod _onCompleteHadler daje obaveštenje da je pdf fajl kreiran.

```
private function _onCompleteHadler(event:ProcessingEvent):void
{
	Alert.show("PDF created!!");
```

```
	//ovde nas obavestava da je gotovo
}
```

Metod onCreatePDFHandler kreira pdf fajl u RAM-u, kreira fajl na hard disku, povezuje dva fajla, smešta podatke iz RAM-a na hard disku I omogućava korisniku da odabere lokaciju na hardisku kao i naziv fajla.

```
private function onCreatePDFHandler(evt:MouseEvent):void
{
pdf = new MSPDF(Orientation.PORTRAIT,Unit.POINT,true,Size.A4);
pdf.addEventListener(ProcessingEvent.COMPLETE,_onCompleteHadler);
if (!pages)
	return;

for (var i:int=0;i<pages.length;i++){
		pdf.addPage();

pdf.addImageStream( PNGEncoder.encode(pages[i].page),
ColorSpace.DEVICE_RGB, null, PAGE_X, PAGE_Y, PAGE_WIDTH, PAGE_HEIGHT,0,1 );

		pdf.setXY(20, 450);
		pdf.writeText(10,pages[i].text);
}

file = new File();
	//File je fleksova klasa za manipulaciju sa fajl sistemom.
file.addEventListener(Event.SELECT, onFileSelectHandler);
file.browseForSave("Save PDF");
}
```

Metod onFileSelectHandler vrši strimovanje podataka, otvara dijalog za izbor lokacije na računaru I imenovanje fajla.

```
private function onFileSelectHandler(event:Event):void
{
	var f:FileStream = new FileStream();
	f.open( file, FileMode.WRITE);
	var bytes:ByteArray = pdf.save( Method.LOCAL );
	f.writeBytes(bytes);
	f.close();
}
```

U listNotificationInterests se registruje šta sluša ovaj medijator.

```
override public function listNotificationInterests():Array
{
	return [
		ApplicationFacade.STUDENT_LOADED

	]
}
```

Metod handleNotification prihva događaj, koji je registrovan u listNotificationInterests. Pomoću switch konstrukcije, utvrđuje se ime događaja. Ako se desio događaj ApplicationFacade.STUDENT_LOADED, onda se pristupa proveri sadržaja za isTest. Ako je njena vrednost false, naredbom break izlazi se iz switch bloka. Međutim, nije false zato što je setovana isTest na true u metodi onTestHandler. To se uradi pre nego što se proverava ovaj deo koda. . Onda se proveri da li je steper <= 50. Jeste, pošto je steper bio 1 u prvom prolazu, u prvom zahtevu. Inkrementira se steper, steper++ i ponovo se poziva getAllStudentiFlex. Sada ne ulazi u blok za else, već se izvršava metod getAllStudentiFlex u proksiju, i time se izvršava drugi zahtev, a onda se u metod getAllStudentiResultHanlder proksija, šalje notifikacija sendNotification(ApplicationFacade.STUDENT_LOADED);, t.j. obaveštenje da je drugi zahtev za učitavanje završen. Notifikacija se registruje u listNotificationInterests, a prihva i obrađuje u handleNotification, i tako se ciklus ponavlja 51 put. Naime, kada je 50 zahtev, on i tad odradi

```
proxy.getAllStudentiFlex(controlMediator.currentPage);.
```

Kada steper dostigne vrednost 51, ulazi se u else blok, gde se isTest setuje na false, steper na 1. Sada se pristupa proveri koji je proxy selektovan . U liniji koda **if** (view.diagram) proverava se da li postoji objekat diagram, ako postoji, onda njegovojosobini averageJson se dodeljuje vrednost :

```
view.diagram.averageAmf=
Number(StudentListVO.amfDataProvider.getItemAt(0).average).toFixed(4);
```

averageJson služi za ispisivanje prosečne vrednosti u dijagramu. Ispis se vrši u ResultDiagram, u liniji displayName="Prosečno vreme izvršenja {averageJson}". U osobini

view.diagram.requestedJson se čuva izabrani broj slogova za učitavanje iz baze, u ComboBox-u.

```
view.diagram.requestedJson=StudentListVO.jsonDataProvider.getItemAt(0).
requested;
```

U slučaju da je proksi StudentProxyJson selektovan, pored komponente view.diagram, može biti aktivna i komponenta view.javaPhpComp, što se proverava if naredbom **if** (view.javaPhpComp), a onda se pristupa njenoj osobini view.javaPhpComp.averageJson i setuje se na vrednos, linijom koda:

```
view.javaPhpComp.averageJson =
Number(StudentListVO.jsonDataProvider.getItemAt(0).average).toFixed(4);

override public function
handleNotification(notification:INotification):void
{
      switch(notification.getName()){
            case ApplicationFacade.STUDENT_LOADED:
if (isTest == false) break;
      if (steper <= 50){
            steper++;
      proxy.getAllStudentiFlex(controlMediator.currentPage);
      }
      else {
            isTest = false;
      steper = 1;//steper se stavlja na 1
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyJson.NAME){
      if (view.diagram){

view.diagram.averageJson =
Number(StudentListVO.jsonDataProvider.getItemAt(0).average).toFixed(4);
view.diagram.requestedJson =
StudentListVO.jsonDataProvider.getItemAt(0).requested;
                              }
if (view.javaPhpComp) {
view.javaPhpComp.averageJson=Number(StudentListVO.jsonDataProvider.getItemA
t(0).average).toFixed(4);
}
```

U narednom bloku koa, proverava se da li je selektovan proxy StudentProxyAmf. Ako je uslov ispunjen, pristupa se osobini view.diagram.averageAmf komponente diagram i postavlja se na vrednost osobine average, objekta, iz nultog elementa niza amfDataProvider. Osobini view.diagram.requestedAmf dodeljuje se vrednost osobine requested, nultog objekta, iz niza amfDataProvider.

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyAmf.NAME){
view.diagram.averageAmf =
Number(StudentListVO.amfDataProvider.getItemAt(0).average).toFixed(4);
view.diagram.requestedAmf =
StudentListVO.amfDataProvider.getItemAt(0).requested;
}
```

U narednom bloku koda, proverava se da li je proxy StudentProxyXml aktivan. Ako jeste, pristupa se proveri stanja. Ako je trenutno stanje "diagrams",pristupa seosobini view.diagram.averageXml komponente diagram i postavlja se na vrednost osobine average, objekta, iz nultog elementa niza xmlDataProvider. Osobini view.diagram.requestedXml dodeljuje se vrednost osobine requested, nultog objekta, iz niza xmlDataProvider. U slučaju da je trenutno stanje "javaPhpXml"komponentejavaXmlCompar, osobini view.javaXmlCompar.averageXml, komponente javaXmlCompar, dodeljuje se vrednost osobine average, objekta, iz nultog elementa niza xmlDataProvider.

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyXml.NAME){
      if (view.currentState == "diagrams"){
view.diagram.averageXml =
Number(StudentListVO.xmlDataProvider.getItemAt(0).average).toFixed(4);
view.diagram.requestedXml =
StudentListVO.xmlDataProvider.getItemAt(0).requested;
}
else if (view.currentState == "javaPhpXml"){
view.javaXmlCompar.averageXml =
Number(StudentListVO.xmlDataProvider.getItemAt(0).average).toFixed(4);
      }
}
```

U narednom kodu, za slučaj da je selektovan proksi StudentProxyJavaJson, osobini view.javaPhpComp.averageJavaJson dodeljuje se vrednost.

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaJson.NAME)
```

```
view.javaPhpComp.averageJavaJson=Number(StudentListVO.jsonJavaDataProvider.
getItemAt(0).average).toFixed(4);
```

Ako je aktivan proksi StudentProxyJavaXml, osobina
`view.javaXmlCompar.averageJavaXml`, komponente javaXmlCompar,
se setuje na vrednost
`StudentListVO.xmlJavaDataProvider.getItemAt(0).average.`

```
      if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaXml.NAME)
view.javaXmlCompar.averageJavaXml =
Number(StudentListVO.xmlJavaDataProvider.getItemAt(0).average).toFixed(4);
            }
      break;
      }
}
public functionget proxy():*
{
      _proxy = facade.retrieveProxy(ApplicationFacade.SELECTED_PROXY);
      return _proxy;
}
```

U metodi get proxy(), linija koda:
`_proxy = facade.retrieveProxy(ApplicationFacade.SELECTED_PROXY);`
znači, nađi proksi imena ApplicationFacade.SELECTED_PROXY, što se prosledi kao string.

3.7.8 Klasa StudentAMF

Kada se pokrene aplikacija, pravi se objekat, klase StudentAMF.mxml, i pošto se kreiraju svi čaildovi ove aplikacije, i kreira aplikacija StudentAMF, onda se dispečuje ivent creationComplete, a time poziva funkcija windowedapplication1_creationCompleteHandler.
Posle svega, kada se kreira aplikacija StudentAMF, kreira se objekat fasada. This je objekat klase StudentAMF.mxml. U ovom objektu je sve što se nalazi u aplikaciji Student.AMF.
U funkciji windowedapplication1_creationCompleteHandler(event) pravi se instanca Facade, _façade. Instanca "StudentAMF" se napravi kada se pokrene ova aplikacija. U ovoj metodi Singlton klasa ApplicationFacade ptistupa statičkoj metodi getInstance, prosleđuje joj jedan string "StudentAMF". This je objekat klase StudentAMF.mxml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
creationComplete="windowedapplication1_creationCompleteHandler(event)"
xmlns:gui="com.school.puremvc.view.gui.*"
>
<fx:Style source="assets/css/style.css" />
```

U tagu <s:states> desfinisana su stanja u komponenti, svako stanje se opisuje klasom State. <s:states> su izrazi, to je jedan array stanja. Klasa State, sadrži osobinu name, prvom stanju je dato ime managament, drugom diagrams, trećem javaPhp, četvrtom javaPhpXml i petom comparation. Dakle, Instanca klase StudentAMF poseduje pet stanja, managament, diagrams, javaPhp, javaPhpXml i comparation. Prva dva stanja su grupisana u grupu regular što znači da se primenjuju iste stvari na oba stanja. Osobina name označava ime stanja, a stanja se mogu grupisati u grupama. Ovde ne mora da se kaže includeIn ="managament and diagrams", nego se piše includeIn="regular" i on će ga primeniti u stanju. U <s:states> array prvo stanje je "managament".

```
      <s:states>
            <s:State name="managament" stateGroups="regular"/>
            <s:State name="diagrams" stateGroups="regular"/>
            <s:State name="javaPhp" />
            <s:State name="javaPhpXml" />
            <s:State name="comparation" />
      </s:states>
      <fx:Script>
            <![CDATA[
                  …
private var _facade:ApplicationFacade;

protected function
windowedapplication1_creationCompleteHandler(event:FlexEvent) :void
{
      trace("inicijalizacija _____");
      _facade = ApplicationFacade.getInstance("StudentAMF");
      _facade.startup(this);
      trace("index elementa: "+vg1.getElementIndex(controlStudent));
}
```

currentState je ugrađena promenljiva, koja čuva trenutno stanje aplikacije. Kada se ne stavi ništa ispred currentState, to se računa kao this. U liniji koda:

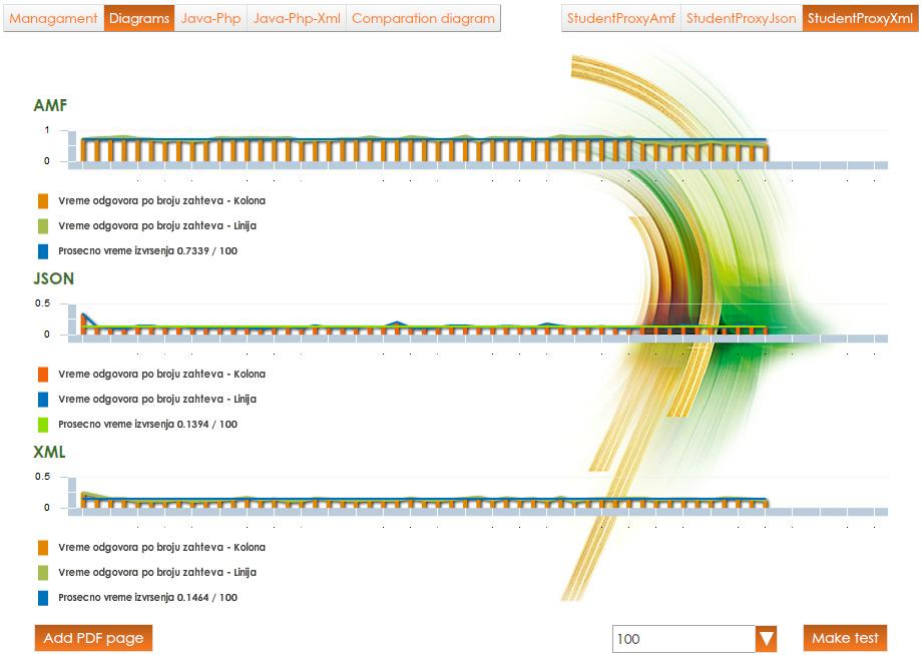
```
currentState = stateSelector.selectedItem.state;
```

stateSelector je id, a selectedItem selektovani objekat i pristupa se promenljivoj state unutar provajdera <fx:Object label="Managament" state="managament"/> u selektovanom objektu. Naime, ovde se čita vrednost state odavde <fx:Object label="Managament" state="managament"/> i smešta se u trenutno stanje aplikacije currentState. A pošto su date iste vrednosti u state <fx:Object label="Managament" state="managament"/>, kao imena stanja, kako se bude menjala poruka state odavde <fx:Object label="Managament" state="XXXX"/>, menja se i stanje aplikacije.

Vrednost koja se smesti u currentState, proverava se u nizu states-ova, ako postoji, primeniće to stanje.

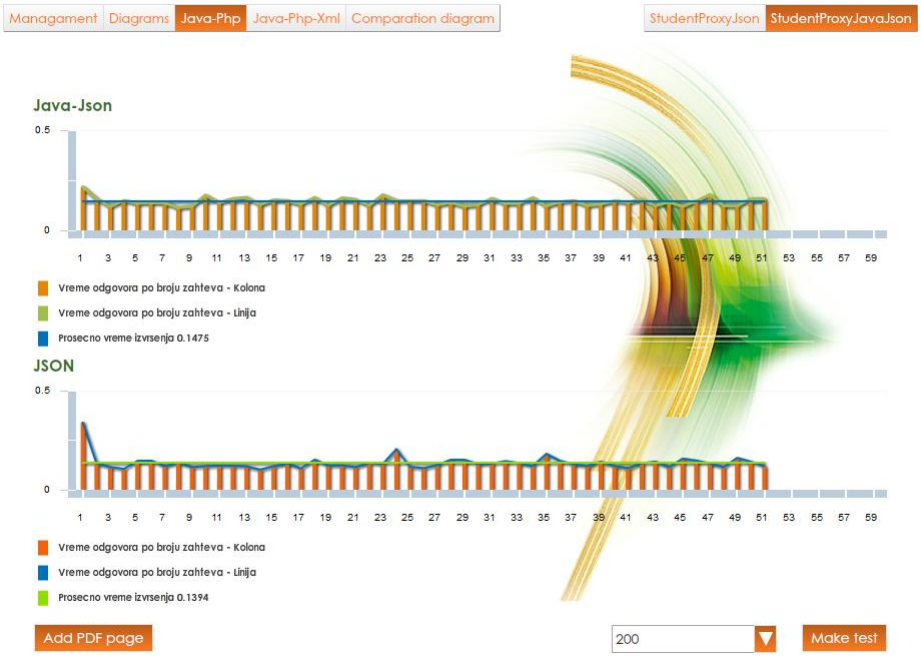
```
<s:states>
<s:State name="managament" stateGroups="regular"/>
<s:State name="diagrams" stateGroups="regular"/>
<s:State name="javaPhp" />
<s:State name="javaPhpXml" />
<s:State name="comparation" />
</s:states>
```

Ako se stavi nešto u currentState, što ne postoji u <s:states>, izazvaće se greška, zato što ne postoji takvo stanje u aplikaciji. U stanju *management*, aplikacija izgleda kao na slici 3.7.34. U stanju *diagrams*, aplikacija ima oblik:



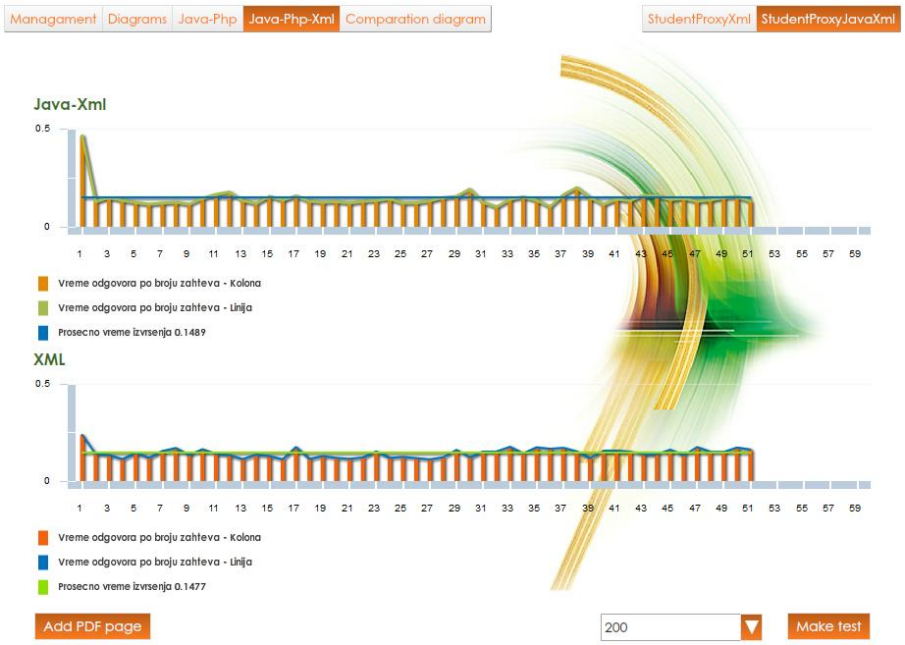
Slika3.7.40. Izgled aplikacije u stanju *diagrams*

U stanju *javaPhp*, aplikacija izgleda kao na slici 3.7.41.



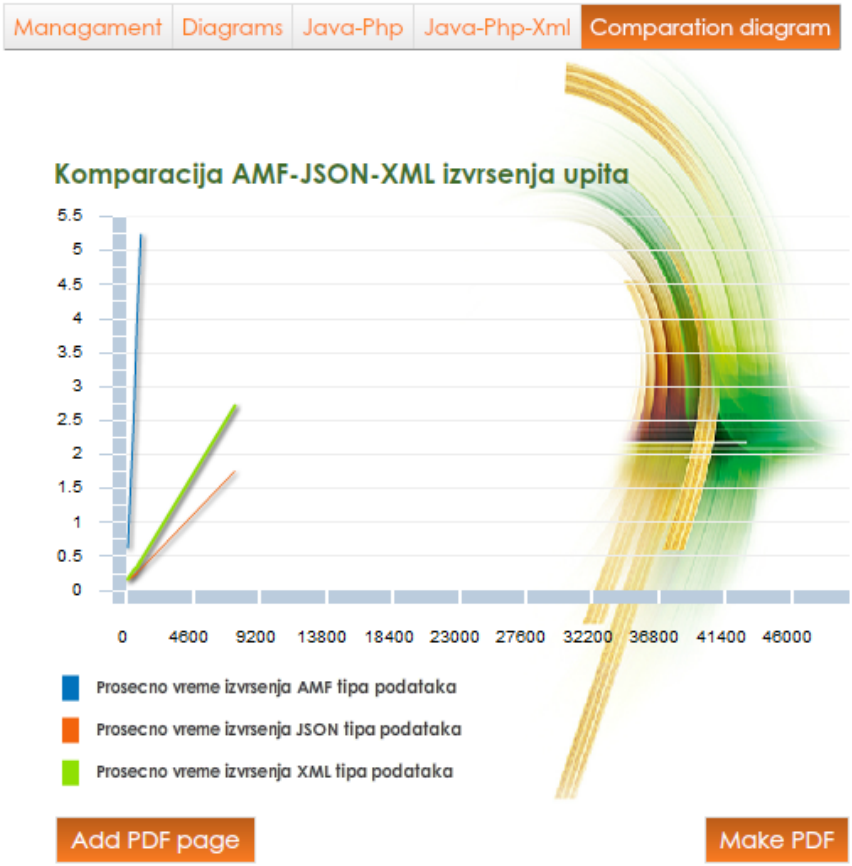
Slika3.7.41. Izgled aplikacije u stanju *javaPhp*

U stanju *javaPhpXml*, aplikacija izgleda:



Slika3.7.42. Izgled aplikacije u stanju *javaPhpXml*

U stanju *comparation*, aplikacija ima oblik:



Slika3.7.43. Izgled aplikacije u stanju *comparation*

```
protected function stateSelector_changeHandler(event:IndexChangeEvent):void
{
	try {
		currentState = stateSelector.selectedItem.state;
	}
	catch(err:Error){
		trace("current state error: "+err.message)
	}
}
]]>
</fx:Script>
<fx:Declarations>
```

U odeljku za deklaraciju, deklarisan je niz, koji ima četiri elementa, tipa Object. Svaki objekt ima dve osobine label i state. Object je dinamičkog tipa, jer se može navesti proizvoljan broj osobina, i po izboru dati ime osobini.

```
	<s:ArrayList
		id="statesProvider"
		>
```

```
        <fx:Object label="Managament" state="managament"/>
        <fx:Object label="Diagrams" state="diagrams"/>
        <fx:Object label="Java-Php" state="javaPhp"/>
        <fx:Object label="Java-Php-Xml" state="javaPhpXml"/>
        <fx:Object label="Comparation diagram" state="comparation"/>
        </s:ArrayList>
    </fx:Declarations>

    <fx:Script>
        <![CDATA[
        ]]>
    </fx:Script>
    <s:SkinnableComponent
        width="100%" height="100%"
        skinClass="com.school.puremvc.view.gui.skins.BackgroundSkin"
        id="backgroundSkin"
        />
```

stateSelector je ime ButtonBar-a, a statesProvider je provajder. Dakle, dataProvider je arrayList statesProvider, koji je deklarisan u deklaraciji(<fx:Declarations>). Ovaj ButtonBar ima pet dugmadi Managament, Diagrams, Java-Php, Java-Php-Xml i Comparation diagram. Osobina labelField definiše ime objekta koji će da prikaže kao label, što znači, u labelField je ime promenljive iz objekta koji će da prikazuje. Prema tome, kada nađe objekat <fx:Object label="Managament" state="managament"/>, ono što piše u label, prikazaće kao tekst a state je drugi deo objekta i služi za selekciju. change funkcija povezuje dugmat na ButtonBar-u.

```
    <s:ButtonBar
        top="25" left="30"
        dataProvider="{statesProvider}"
        labelField="label"
        id="stateSelector"
        change="stateSelector_changeHandler(event)"
        />
```

Kada se stavi dataProvider.regular, onda se podržava jedno i drugo stanje, managament i diagrams. Stanja se grupišu, da bi se u nekim elementima primenila stanja, koja čine jednu grupu.

```
<s:ButtonBar
```

```
id="selectProxyBar"
dataProvider.regular="{new ArrayList(ApplicationFacade.PROXIES)}"
dataProvider.javaPhp="{new ArrayList(ApplicationFacade.PROXIES_JAVA)}"
            dataProvider.javaPhpXml="{new
      ArrayList(ApplicationFacade.PROXIES_JAVA_XML)}"
            selectedIndex="0"
            top="25" right="30"
            excludeFrom="comparation"
            />
```

Kada se pokrene aplikacija, program analizira elemente, prvi element HGroup je vidljiv, zato što nije definisano stanje, zatim, VGroup je vidljiv, jer je u menadžmentu, pravi se u "managament" a ResultDiagrams je dodat u "diagrams". Komponenta JavaPhpComparationDiagram je dodata u "javaPhp", dok je komponenta JavaPhpXmlComparationDiagram dodata u "javaPhpXml" i ComparationDiagram je includeIn u "comparation".

```
      <s:HGroup
            width="100%" height="100%"
            top="100" left="40" right="40"
            id="hg1"
            >
```

Struktura VGroup poziva metod addChild() ili addElement(), t.j. direktno hg1.addElement(vg1). Kontejner VGroup sadrži dva elementa HGroup i StudentControlComponent. Kontejner VGroup se sastoji od HGroup-e, koja sadrži tabelu StudentTableComponent, menadžer komponentu StudentManagerComponent i kontrole StudentControlComponent.U array stanja, prvo stanje je managament, i ono se prvo prikazju kada se pokrene aplikacija.Naredbom includeIn="managament" u <s:VGroup> kontejneru, definiše se stanje, kada se prikazuje ovaj kontejner.

```
      <s:VGroup
            width="100%" height="100%"
            includeIn="managament"
            id="vg1"
            >
            <s:HGroup
                  width="100%"
                  height="100%"
                  id="hg2"
                  >
<gui:StudentTableComponent id="tableStudent" width="70%" height="90%"/>
```

```
<gui:StudentManagerComponent id="managerStudent" width="30%"/>
</s:HGroup>
<gui:StudentControlComponent id="controlStudent" />
</s:VGroup>
<gui:ResultDiagrams width="100%" height="100%" id="diagram"
includeIn="diagrams"/>
```

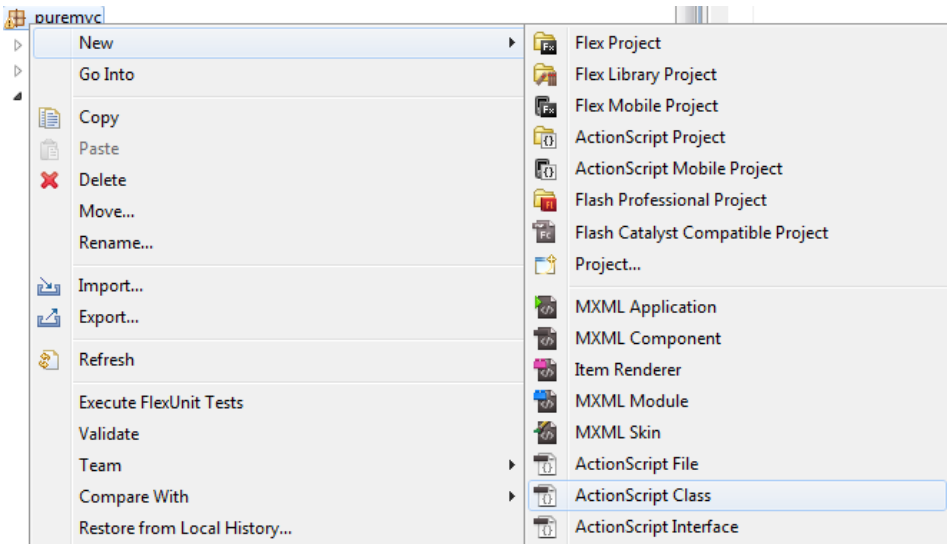
Kada se pokrene aplikacija, ne prikazuje se dijagram, zato što je po difoltu prvo stanje managament. Kada se ima array stanja, uzima se nulti element, a prvi string je "managament".

```
<gui:JavaPhpComparationDiagram width="100%" height="100%" id="javaPhpComp"
includeIn="javaPhp"/>
<gui:JavaPhpXmlComparationDiagram width="100%" height="100%"
id="javaXmlCompar" includeIn="javaPhpXml"/>
<gui:ComparationDiagram width="100%" height="100%" id="comparationDiagram"
includeIn="comparation"/>
</s:HGroup>
</s:WindowedApplication>
```

Metod getInstance se nalazi u klasi ApplicationFacade.

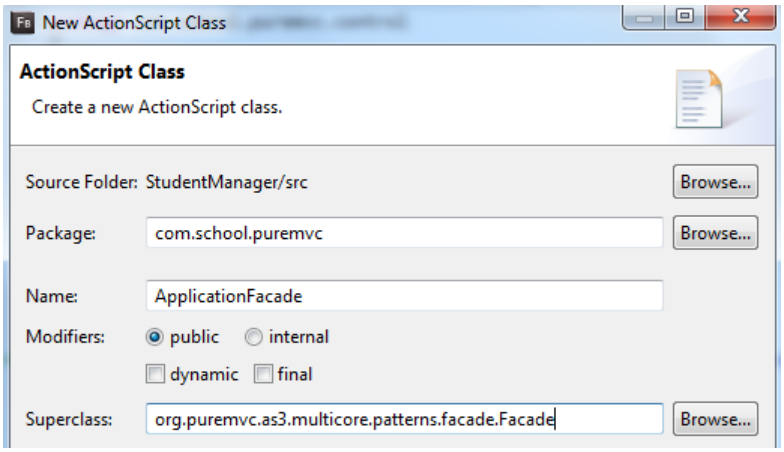
3.7.9 Klasa ApplicationFacade

MVC se sastoji od tri stvari, jedna je contol druga model i view. Sve one čine jednu celinu. A ima nešto što ih sve spaja ili monitoriše, to se zove facada. Ta četvrta stvar Facada po pravilu treba da stoji u nekoj strukturi. Dodaje se:



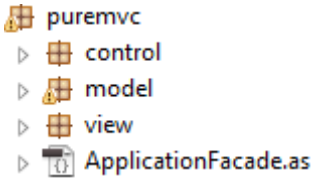
Slika 3.7.44. Način kreiranja klase ApplicationFacade

Klasi je dato ime ApplicationFacade, i ona nasleđuje klasu facade:



Slika 3.7.45. Dijalog prozor za kreiranje klase ApplicationFacade

ApplicationFacade se nalazi u paketu puremvc.

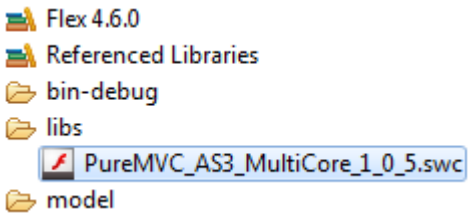


Slika 3.7.46. Prikaz strukture paketa puremvc

ApplicationFacade izgleda:

```
ackage com.school.puremvc
{
	import org.puremvc.as3.multicore.patterns.facade.Facade;

	public class ApplicationFacade extends Facade
	{
		public function ApplicationFacade(key:String)
		{
			super(key);
		}
	}
}
```

ApplicationFacade je nasledila klasu *Façade*. Klasa *Façade* je ugrađena u biblioteku *puremvc-as3-multicore-framework-master.mvc* .



Slika 3.7.47. Prikaz paketa libs, gde se dodaje *puremvc-as3-multicore-framework-master.mvc*

Façade klasa je singlton klasa, što znači da ne može da se napravi više objekata od singlton klase. Praktično, instanca klase postoji samo jednom u svom izdanju, instancira se samo jednom i ona živi za svo to vreme. Njoj se pristupa, tako što se uvek napravi jedna funkcija koja se zove getInstance.

Postoje dve verzije MVC biblioteka, Single i MultiCore. MultiCore znači više jezgara, i može podržati više aplikacija. Dakle, *ApplicationFacade*, podržava više aplikacija, jer je odabrana MultiCore verzija MVC-a, pošto se računa da je jedna aplikacija jedno jezgro. Aplikacije u našem projektu su:

- *StudentDataGridHeaderRenderer.mxml*
- *AddTextPdfComponent.mxml*
- *StudentAMF.mxml*
- *StudentDataGridSkin.mxml*

Klasa ApplicationFacade sadrži četiri metode i devet statičkij javnih konstanti. Konstante su:

```
public static const PROXIES:Array =
["StudentProxyAmf","StudentProxyJson","StudentProxyXml"];
public static const PROXIES_JAVA:Array =
["StudentProxyJson","StudentProxyJavaJson"];
public static const PROXIES_JAVA_XML:Array =
["StudentProxyXml","StudentProxyJavaXml"];

[Bindable]
public static var SELECTED_PROXY:String = "StudentProxyAmf";

//Ovde je kreirana konstantu STARTUP i ima vrednost startup.
public static const STARTUP:String = "startup";
public static const STUDENT_SELECTED:String = "studentSelected";
public static const STUDENT_DELETED:String = "studentDeleted";
```

```
public static const STUDENT_ADDED:String = "studentAdded";
public static const STUDENT_LOADED:String = "studentLoaded";
```

Konstante PROXIES, PROXIES_JAVA i PROXIES_JAVA_XML su nizovi, koji sadrže elemente tipa string.

Metode ove klase su:

- *Konstruktor metod ApplicationFacade*
- *startup*
- *initializeController*
- *getInstance*

Metod getInstance, ima oblik:

```
publicstaticfunction getInstance(key:String):ApplicationFacade
  {
      if (instanceMap[key] == null)
        instanceMap[key] = new ApplicationFacade(key);
      return instanceMap[key] as ApplicationFacade;
  }
```

instanceMap je promenljiva unutar klase Façade, nizovnog je tipa[1,3]. Konstrukcija instanceMap[key] može se posmatrati i kao asocijativni niz, gde se elementima niza pristupa pomoću vrednosti key, gde key može sadržati različite key. Pošto je MultiCore jezgro, klasa Façade može da radi sa više aplikacija u isto vreme. Ako je jedna aplikacija onda je to instanceMap, ako je više aplikacija onda je to array instanceMap[key], gde key određuje zapravo sa kojom aplikacijom se trenutno radi. Postoje još dve funkcije koje treba da postoje u čitavoj priči, jedna je starup funkcija a druga je initializeController.

U redu if (instanceMap[key] == null) proverava se, da li je instanceMap od key ravna nuli, gde je instanceMap tipa Array. Ako je instanceMap[key] jednako null, istom ovom objektu(instanceMap['StudentAMF']) se dodeljuje nova vrednost klase ApplicationFacade, i prosleđuje se StudentAMF. Ovde je instanceMap nasleđena iz roditeljske klase Façade. Napravljena je nova instanca ove klase ApplicationFacade, i prosleđen je samo ključ StudentAMF. Ako nije instanceMap['StudentAMF'] jednak null, to znači da objekat instanceMap['StudentAMF'] već postoji, instanciran je ranije, onda se ne instancira već se vraća postojeći u liniji koda:

```
return instanceMap['StudentAMF'];
```

Dakle, vrati se vrednost iz metode. Objekat klase ApplicationFacade ima polje NAME i njemu se prosleđuje ime, koje je id, unikatno. Façade je MultiCore klasa, što znači da ovaj freim može raditi sa više jezgara, sa više klasa. To znači da se može sledećem objektu dodati još jedna aplikacija, koja se na primer zove ProfesorAmf, t.j. instanceMap['ProfesorAmf']. Sada se proverava da li postoji instanca ProfesorAmf:

```
if instanceMap['ProfesorAmf'] == null
```

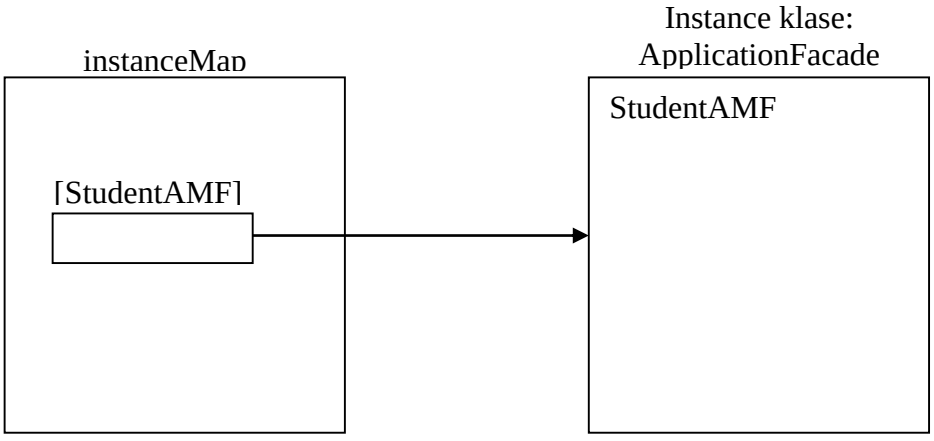
ako je izraz ravan nuli, to znači da ne postoji instanca pod imenom professor ili referenca, t.j. polje unutar objekta instanceMap je ravno nuli. Sada je potrebno kreirati novi object, koji se zove ProfesorAmf.

```
instanceMap['ProfesorAmf'] = new ApplicationFacade("ProfesorAmf")
```

Sada postoji u ovoj fasadi, u celokupnom mehanizmu dva objekta klase ApplicationFacade, koji se zovu StudentAMF i ProfesorAmf. Oba objekta su istog tipa, tipa klase ApplicationFacade. Ovde može da se definiše koji objekat radi sa kojim medijatorima i proksijima, i mogu se razmenjivati informacije između ove dve aplikacije. Ova dva objekta praktično dele iste proksije, dele iste medijatore i dele iste kontrolere, samo je potrebno u aplikaciji napraviti notifikacije, koji određuje ko će šta da radi. Kako je Façade singleton klasa, to znači da se ne mogu napraviti dva objekta sa imenom ProfesorAmf. Može da se napravi samo jedan, ako se pokuša napraviti drugi, utvrdiće da instanceMap['ProfesorAmf'] nije null, i vratiće onaj stari objekat, naredbom:

```
return instanceMap['StudentAMF'];
```

Sada može da se napravi i treći objekat, ali može da se napravi samo jedanput.Proces instanciranja objekta tipa ApplicationFacade, prikazan je narednom slikom.



Slika 3.7.48. Grafički prikaz instanciranja objekta, klase ApplicationFacade.

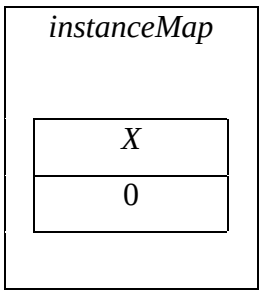
instanceMap nalazi se unutar MVC framework-a, u klasi Façade. instanceMap je objekat, objekti mogu imati ključeve koji su asocijativni. Naziv polja u objektu je key. Konstruktor klase ApplicationFacade je:

```
public function ApplicationFacade(key:String)
{
	super(key);
	trace("korak 3 _____________________________")
}
```

Konstruktor klase ApplicationFacade zahteva jedan parameter tipa string, koji je ključ, nije opcioni već je obavezan. Taj parameter se prosleđuje konstruktoru roditeljske klase super(key); Ovaj ključ je jedinstveni id. Objekti unutar niza instanceMap su dinamički. U liniji koda instanceMap.x=0; ako ne postoji promenljiva x u objektu instanceMap, kreiraće polje x a onda mu dodeliti vrednost 0, a ako postoji samo će u njega upisati nulu. instanceMap posmatran kroz objektni nivo, može se predstaviti i grafički.

```
var instanceMap:Object;

instanceMap.X=0;
```

Objekt kao i Array dopušta definisanje polja izrazom: instanceMap[‘X’]=0;. X je polje unutar objekta instanceMap, i ima vrednost nula.



Slika 3.7.49. Grafički prikaz objekta instanceMap

Ako klasa nije singlton, može se više objekata pod istim imenom instancirati u memoriji. Sistem raspoznaje te objekte a korisnik ne.

```
for (var i:int=0;i<2;i++)
var app:ApplicationFacade = new ApplicationFacade("StudentAmf");
```

U ovom slučaju se smatra da klasa ApplicationFacade nije singleton. U primeru, app za i=0, ukazivao bi na prvi objekat, a za i=1, app bi pokazivao na drugi objekat, tako da bi postojala dva objekta u memoriji, a u ovom slučaju app bi pokazivao na zadnji od njih.

Ako je u pitanju singleton klasa onda se samo prvi put alocira memorija a svaki naredni put se ne alocira već se dodaje stari objekat, objekat koji već postoji u nekom okruženju, promenljivoj koji je prethodno alociran:

```
for (var i:int=0;i<2;i++)
var app:ApplicationFacade = new ApplicationFacade("StudentAmf");
```

Ovde se smatra, kao i što jeste, da je klasa ApplicationFacade tipa singleton.

Kada se u klasi StudentAMF pozove metod
`windowedapplication1_creationCompleteHandler,`
u prvom koraku se pozove statička metod getInstance, klase ApplicationFacade, kojoj se prosledi string "StudentAMF". U metodi getInstance instancira se objekat ako ne postoji, čije ime je "StudentAMF". U liniji koda instanceMap[key] = new ApplicationFacade(key);, alocira se memorija za objekat. U momentu alociranja memorije, poziva se konstruktor metod klase ApplicationFacade. U konstruktor metodi klase ApplicationFacade, poziva se konstruktor osnovne klase naredbom super(key);. Iz konstruktora osnovne klase, poziva se override metod initializeController(). U ovoj metodi vrši se registracija događaja STARTUP i STUDENT_DELETED. Registracija command class se dešava prilikom instanciranja klase ApplicationFacade, ovde se registruje klasa StartupCommand, koja se pokreće slanjem događaja STARTUP. Takođe, u metodi initializeController registruje se i klasa StudentDeletedCommand, koja se pokreće slanjem događaja STUDENT_DELETED.

```
overrideprotected function initializeController():void{
	trace("registracija macro command")
	super.initializeController();
	registerCommand(STARTUP,StartupCommand);
	registerCommand(STUDENT_DELETED,StudentDeletedCommand);
	//registerCommand znaci, facade.children.push(StartupCommand)
}
```

U drugom koraku, unutar metode windowedapplication1_creationCompleteHandler, klase StudentAMF, naredbom _facade.startup(**this**);, instancirani objekat _facade, pristupa metodi startup, kao parameter prosleđujejoj this t.j. instance klase StudentAMF. Unutar metode startup, šalje se notifikacija. U notifikaciji se navodi ime događaja STARTUP i telo događaja value, a to je instance klase StudentAMF.

```
public function startup(value:StudentAMF):void{
      sendNotification(STARTUP,value);
}
```

sendNotification je ugrađena funkcija klase Façade. Funkcija sendNotification šalje obaveštenje tipa STARTUP, i prosleđuje value, value je vrednost cele aplikacije. Slanjem događaja STARTUP, pokreće seklasa StartupCommand.

3.7.10 Mediator StudentControlMediator

Medijator povezuje sistem mvc-a sa grafičkim interfejsom. Grafički interfejs komunicira sa klijentom, medijator prihvata sve zahteve od korisnika i prikazuje podatke korisniku i zato se zove grafički korisnički interfejs. Dakle, medijator upravlja korisničkim grafičkim interfejsom, prihvata sve zahteve od korisničkog interfejsa, prosleđuje ih onom kome treba da ih prosledi, a to su proksiji. Ako se klikne na dugme next, znači korisnik zahteva neke podatke, medijator je prihavatio zahtev, prosledi ga proksiju. Proksi prihvata zahtev, obrađuje ga, komunicira sa serverom, prihvata rezultate, obrađuje rezultate, i vraća rezultate medijatoru, a medijator ispisuje te rezultate na grafički interfejs. Najpravilnije je da medijator prihvati zahtev od korisničkog interfejsa, pošalje informaciju kontroleru, a kontroler aktivira proksi, i onda proksi vrati informaciju medijatoru. Pitanje je inženjeringa, kada je u pitanju povezivanje Kontrolera, Medijatora i Kontrolera. Problem je ako se pravi velika kompleksna aplikacija, gde se ima na stotine upita u bazu podataka, a kontroler je simpl kontroler, SimpleCommand. On prihvata samo jednu komandu, to znači da se mora napravidi sto klasa komande. Ovde dolazi do balansa projektanta projekta, bitno je, gde se koriste komande i u kojoj meri se koriste komande. Ovde se koristi minimalan broj komandi, prva komanda kojom se startuje je StartupCommand, i ta komanda će da startuje dve druge komande. Jedna će da registruje proksije a druga registruje medijatore. Komande se mogu koristiti po željama , praktično jedna komanda bi bila next dugme, daj sledeću grupu podataka. Druga komanda bi bila daj prethodni blok podataka. Komanda bi bila obriši studente, komanda bi bila dodaj studente. Ovde pritiskom na dugme delete, kombinuje se medijator i komanda, reaguje komanda StudentDeletedCommand i medijator StudentManagerMediator. U medijatoru se briše red u rešeci, a u komandi se poziva metod reset medijatora StudentManagerMediator, koja setuje input poja na nulu.

Medijator StudentControlMediator prihvata zahteve od objekta controlStudent, klase StudentControlComponent. Objekat controlStudent registruje se u klasi StudentAMF. Ovaj medijator sadrži jednu statičku konstanu, tri osobine i jedanaest metoda.

Konstanta i osobune u klasi deklarisane suna sledeći načim.

public static const NAME:String = **"StudentControlMediator"**;
private var managerMediator:StudentManagerMediator;
private var _proxy:*;
[**Bindable**]
public var currentPage:int = 0;

Metode koje se koriste, date su u listi:

- *StudentControlMediator*
- ***get*** *view*
- *onRegister*
- *onRemove*
- *onSearchHandler*
- *onPrevioustHandler*
- *onNextHandler*
- *onAddHandler*
- *onEditHandler*
- *onDeleteHandler*
- ***get*** *proxy*

Mediator StudentControlMediator se instancira, registruje i pokreće u klasi komande ViewStartupCommand, linijom koda:

```
facade.registerMediator(new StudentControlMediator(app.controlStudent));
```

Registracijom se u listu za registraciju mediatora čuva objekat klase medijator StudentControlMediator. U momentu instanciranja, kao argument konstruktoru klase StudentControlMediator, neophodno je navesti grafičku komponentu, jer konstruktor očekuje grafičku komponentu, čija je sfera interesovanja. Sfera interesovanja instance klase StudentControlMediator je grafička komponenta app.controlStudent. Naredbom za instanciranje new StudentControlMediator, poziva se konstruktor funkcija.

```
public function StudentControlMediator(_viewComponent:Object)
```

```
{
	super(NAME, _viewComponent);
	trace("construct mediator _____________________________")
}
```

Konstruktor funkciji prosleđen je grafički objekat app.controlStudent iz komand klase ViewStartupCommand, prilikom poziva konstruktor funkcije mediatora StudentControlMediator. Na grafički objekat app.controlStudent sada referencira i _viewComponent:Object promenljiva. U konstruktoru StudentControlMediator se poziva super funkcija, t.j. konstruktor osnovne klase kojem se eksplicitno prosleđuje ime "StudentControlMediator";, a to je unikatno ime ovog medijatora i prosleđuje mu se komponenta _viewComponent sa kojom će komunicirati. Ovde postoji i ugrađena promenljiva viewComponent, koja prihvata sadržaj iz _ viewComponent u konstruktoru osnovne klase mediatora. Grafička komponenta, čija sfera interesovanja ovog medijatora je StudentControlComponent.mxml.

Pored konstruktor funkcije, koristi se i geter funkcija get view. Dakle, komponenta viewComponent ima vrednost objekta _viewComponent. Ovde se koristi **get** view zbog kastovanja, jer je viewComponent tipa Object

```
public functionget view():StudentControlComponent
{
	return viewComponent as StudentControlComponent;
}
```

U ovom objektu, pored konstruktora, implicitno se poziva i override metod onRegister. Prvo se izvrši super funkcija pa tek onda se registruje. Ova funkcija se pokrene na registraciju. Isti posao može da se uradi u konstruktoru, i u onRegister funkciji, ali se to ne preporučuje, onda bi se dobio veći fajl i sporije radi aplikacija. Sve se radi u onRegister iz prostog razloga da se ne opterećuje konstruktor previše. Pusti se konstruktor da on što pre završi svoj posao. On Register je ugrađen u klasu medijator. Naime, u onRegister se registruju onListener-i. Prema tome, ova funkcija se odmah poziva pri instanciranju i registraciji ovog medijatora.

```
override public function onRegister():void
{
trace("onRegister mediator _____________________________")
view.next.addEventListener(MouseEvent.CLICK, onNextHandler,false,0,true);
view.prev.addEventListener(MouseEvent.CLICK,onPrevioustHandler,false,0,true);
```

```
view.addStudent.addEventListener(MouseEvent.CLICK, onAddHandler,false,0,true);
view.editStudent.addEventListener(MouseEvent.CLICK, onEditHandler,false,0,true);
view.delStudent.addEventListener(MouseEvent.CLICK, onDeleteHandler,false,0,true);
view.searchStudent.addEventListener(MouseEvent.CLICK,onSearchHandler,false,0,true);
managerMediator = facade.retrieveMediator(StudentManagerMediator.NAME) as
StudentManagerMediator;
}
```

StudentControlMediator je mediator koji kontroliše dugmad.U metodi onRegister osobinama view.next, view.prev, view.addStudent, view.editStudent, view.delStudent, view.searchStudent, dodeljuje se listener, na događaj MouseEvent.CLICK, i poziva se metod onNextHandler, onPrevioustHandler, onAddHandler, onEditHandler, onDeleteHandler, onSearchHandler respektivno. Linijom koda:

```
managerMediator = facade.retrieveMediator(StudentManagerMediator.NAME) as
StudentManagerMediator;
```

Pronalazi se instanca klase StudentManagerMediator iz liste mediatora, i referenca na taj objekat se dodeljuje promenljivoj managerMediator. Dakle, postoji zahtev, da se otkrije mediator imena StudentManagerMediator.NAME, i time se ostvaruje mogućnost komuniciranja sa mediatorom managerMediator, naime, sada mogu da komuniciraju medijatori, jedan sa drugim. Osobine, kojima je dodeljen listener u onRegister funkciji, prikazane su na narednoj slici3.7.50.

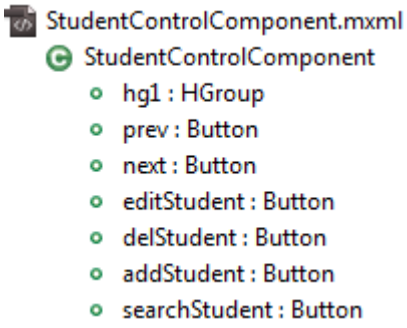
Slika3.7.50. Dugmat za navigaciju

Metod onRemove vrši ukidanje slušaoca događaja na osobine view.next, view.prev, view.addStudent, view.editStudent, view.delStudent i view.searchStudent u slučaju klika na njih.

```
override public function onRemove():void{
view.next.removeEventListener(MouseEvent.CLICK, onNextHandler);
view.prev.removeEventListener(MouseEvent.CLICK, onPrevioustHandler);
view.addStudent.removeEventListener(MouseEvent.CLICK, onAddHandler);
view.editStudent.removeEventListener(MouseEvent.CLICK, onEditHandler);
view.delStudent.removeEventListener(MouseEvent.CLICK, onDeleteHandler);
view.searchStudent.removeEventListener(MouseEvent.CLICK, onSearchHandler);
}
```

3.7.11 Klasa StudentControlComponent

Klasa StudentControlComponent.mxml, predstavlja grafičku komponentu, i sadrži šest javnih osobina:



Slika 3.7.51. Struktura klase StudentControlComponent

Kod koji opisuje klasu StudentControlComponent.mxml ima oblik:

```
<s:HGroup
	width="100%"
	left="40" right="40" bottom="20"
	id="hg1"
	>
	<s:Button
		id="prev"
		label="Previous"
		/>
	<s:Button
		id="next"
		label="Next"
		/>
	<s:Spacer width="100" id="spacer" />
	<s:Button
		label="Edit student"
		id="editStudent"
		/>
	<s:Button
		label="Delete student"
		id="delStudent"
		/>
	<s:Button
		label="Add student"
		id="addStudent"
		/>
```

```
            <s:Button
                  label.normal="Filter"
                  label.search="Back"
                  id="searchStudent"
                  />
      </s:HGroup>
</s:Group>
```

U ovoj klasi su desfinisana stanja <s:states> u komponenti. <s:states> su izrazi, odnosno jedan array stanja. Unutar taga <s:states> može se definisati klasa State. States klasa definiše name, i prvom stanju je dato ime normal a drugom search. Dakle, Ovde postoje dva stanja, normal i search.

```
      <s:states>
            <s:State name="normal" />
            <s:State name="search" />
      </s:states>
```

Po difoltu prvo stanje je normal. U slučaju da aplikacija ima array stanja, uzima se stanje u nultom elementu. Kada se pokrene aplikacija, program gleda po elementima, i prvi element HGroup je vidljiv, zato što nije definisano stanje, zatim, Button elementi "prev", "next", "editStudent", "delStudent" "searchStudent" i "spacer" su vidljivi, jer nijedno stanje nije definisano u ovim elementima.

Prema tome, kada se pokrenuta aplikacija, sadržaj osobine label objekta searchStudent, tipa Button je Filter, zato što je po difoltu prvo stanje normal. S obirom da ima array stanja, promenljiva *currentState* uzima stanje na nultoj poziciji niza, a to je string ="normal".

```
            <s:Button
                  label.normal="Filter"
                  label.search="Back"
                  id="searchStudent"
                  />
```

Metod *onSearchHandler koja* se nalazi u fajlu *StudentControlMediator*.as, se poziva, klikom na dugme *view.searchStudent*.

```
private function onSearchHandler(evt:MouseEvent):void
{
```

```
        if (view.currentState == "normal"){ //Ako je normal, prebaci na
search
                view.currentState = "search";
                managerMediator.view.currentState = "search";
}
        else {     //Ako je search, vrati na normal
                view.currentState = "normal";
                managerMediator.view.currentState = "normal";
        }
}
```

Dakle, osobina label dugmeta view.searchStudent u zavisnosti od stanja unutar klase StudentControlComponent.mxml, može uzeti dve različite vrednosti, Search i Back. Kada je stanje normal, osobina label ima vrednost Search, a kada je stanje search, label je Back, što je prikazano na slici 3.7.52:

Slika3.7.52. Izgled dugmadi, kada je stanje normal

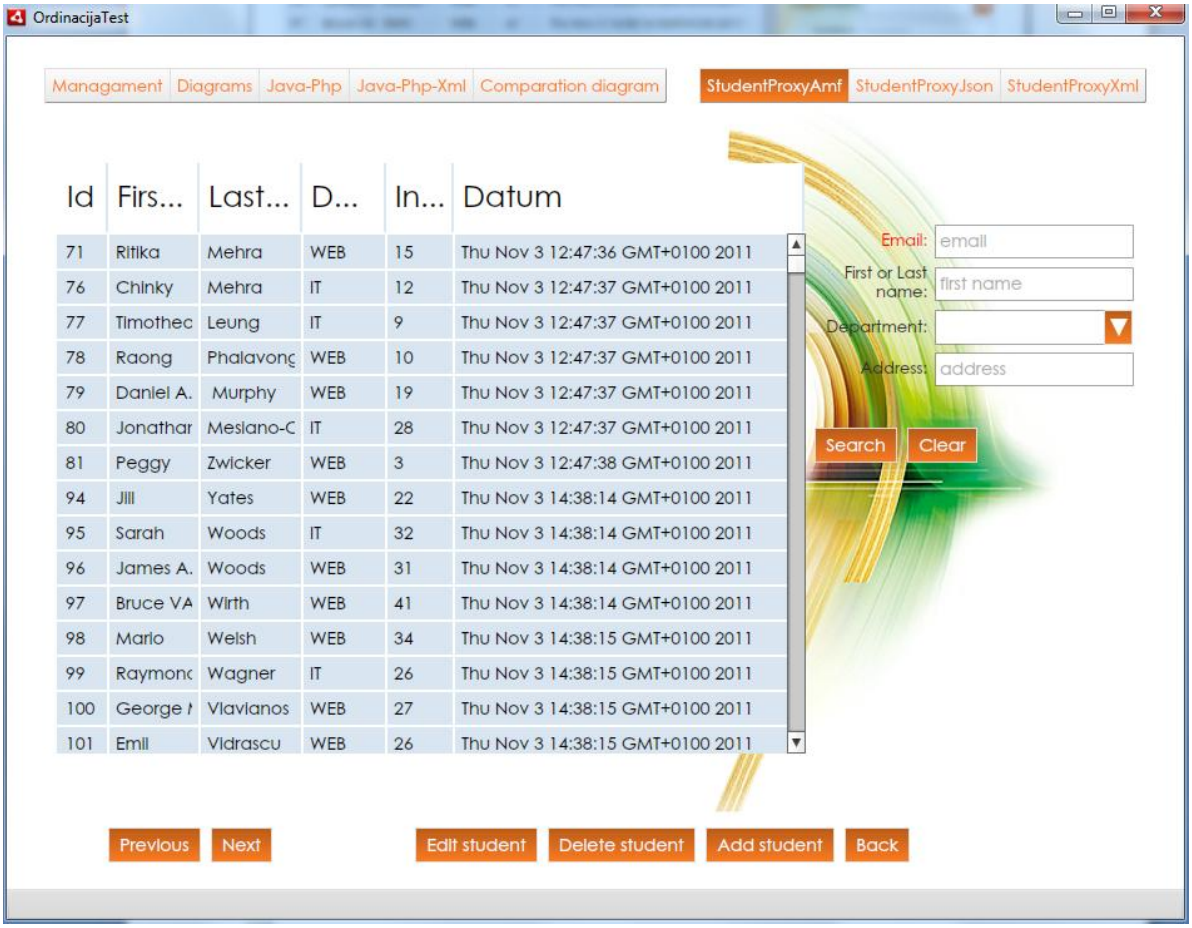
Slika 3.7.53. Izgled dugmadi, kada je stanje search

Promena stanja u aplikaciji, a time i izgled za dugme searchStudent, realizju se u metodi onSearchHandler. Kada se pokrene aplikacija, currentState uzima vrednost iz <s:states>, koja je navedena prva u arraylist-i, a to je normal. Pritiskom na dugme searchStudent, poziva se metod onSearchHandler. U njoj se if naredbom proverava, da li je stanje u grafičkoj komponenti StudentControlComponent.mxml **"normal", a jeste, onda se stanje menja u "search" naredbom** view.currentState = **"search"**;, a zatim se menja stanje u grafičkoj komponenti StudentManagerComponent.mxml. Pritiskom na dugme searchStudent (slika 3.7.34.), čija osobina label ima vrednost Filter, stanje grafičke komponente StudentControlComponent.mxml i StudentManagerComponent.mxml se menja u search. Dakle kada se pritisne na searchStudent, u metodi onSearchHandler se vrši promena stanja iz normal u search, a to se trenutno reflektuje i na dugme searchStudent, gde label u stanju search ima vrednost Back,

```
                <s:Button
```

```
label.normal="Filter"
label.search="Back"
id="searchStudent"
/>
```

i aplikacija ima oblik:



Slika 3.7.54. Izgled aplikacije, kada je stanje search

Klikom na osobinu view.prev poziva se metod onPrevioustHandler.

```
private function onPrevioustHandler(evt:MouseEvent):void
{
	if (currentPage < StudentListVO.requestNumber){
		return;
	}
	currentPage = currentPage - StudentListVO.requestNumber;
	proxy.getAllStudentiFlex(currentPage);
}
```

Klikom na osobinu view.next poziva se metod onNextHandler.

```
private function onNextHandler(evt:MouseEvent):void
{
	if (currentPage >= StudentListVO.totalResults){
		return;
	}
	currentPage = currentPage + StudentListVO.requestNumber;
	proxy.getAllStudentiFlex(currentPage);
}
```

Klikom na osobinu view.addStudent, poziva se metod onAddHandler.

```
private function onAddHandler(evt:MouseEvent):void
{
	managerMediator.addStudent();
}
```

Klikom na osobinu view.editStudent poziva se metod onEditHandler.

```
private function onEditHandler(evt:MouseEvent):void
{
	managerMediator.editStudent();
}
```

Klikom na osobinu view.delStudent poziva se metod onDeleteHandler.

```
private function onDeleteHandler(evt:MouseEvent):void
{
	managerMediator.deleteSelectedStudent();
}
```

U metod get proxy , traži se proksi imena ApplicationFacade.SELECTED_PROXY, što se prosledi kao string.

```
public functionget proxy():*
{
	_proxy = facade.retrieveProxy(ApplicationFacade.SELECTED_PROXY);
	return _proxy;
}
```

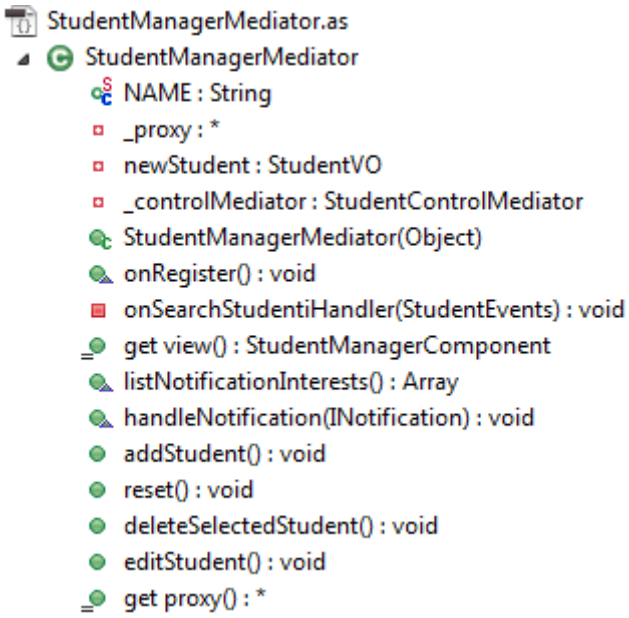
3.7.12 Medijator StudentManagerMediator

StudentManagerMediator je klasa, koja uspostavlja vezu između Facade i grafičkog elementa managerStudent, klase StudentManagerComponent koji je registrovan u klasi StudentAMF, linijom koda:

```
<gui:StudentManagerComponent id="managerStudent" width="30%"/>
```

Klasa StudentManagerMediator sadrži jednu statičku konstantu, tri osobine i jedanaest metoda.

Osobine i metode su prikazane na slici 3.7.55.



Slika3.7.55. Prikaz osobina i metoda za klasu StudentManagerMediator

Objekat klase StudentManagerMediator se instancira i registruje u kontroleru ViewStartupCommand. U momentu instanciranja, poziva se konstruktor metod klase StudentManagerMediator, kao parameter prosleđuje mu se grafički objekat managerStudent.

```
public function StudentManagerMediator(viewComponent:Object)
{
      super(NAME, viewComponent);
}
```

Metod onRegister se poziva u momentu registracije objekta medijatora. U ovoj metodi dodaje se eventListener na grafičku komponentu StudentManagerComponent. Kada se desi događaj

SEARCH_STUDENTI, poziva se metod onSearchStudentiHandler. Događaj SEARCH_STUDENTI se dispečuje u metod searchButton_clickHandler, koja se poziva klikom na dugme Search.

```
override public function onRegister():void
{
view.addEventListener(StudentEvents.SEARCH_STUDENTI,
onSearchStudentiHandler);
}
```

U metodi onSearchStudentiHandler, linijom koda if (_controlMediator == null), se proverava da li je promenljiva _controlMediator ravna nuli, ako jeste, promenljivoj se dodeljuje referenca na objekat, čije ime je StudentControlMediator.NAME.

```
_controlMediator = facade.retrieveMediator(StudentControlMediator.NAME) as
StudentControlMediator;
```

Objekat se nalazi u nizu unutar facade, gde su registrovani svi medijatori. U drugom koraku pravi se dinamički i generički objekat obj. Instanci obj se pridružuje osobina fname, i dodeljuje joj se vrednost view.fname.text. Sa view.fname.text pristupa se osobini fname.text komponente StudentControlMediator. Lokalnom objektu obj, pridružuje se osobina address i dodeljuje joj se vrednost iz TextInput polja view.address.text. Na isti način se kreira osobina email objektu obj i dodeljuje mu se vrednost view.email.text. U liniji koda:

```
if (view.departm.selectedItem != null)
      obj.department = view.departm.selectedItem.id;
else
      obj.department = "";
```

proverava se da li je selektovana stavka iz ComboBox-a. Ako jeste, onda se obj.department dodeljuje redni broj selektovane stavke view.departm.selectedItem.id. Ako nije ništa selektovano onda se osobini department dodeljuje prazan string. U osobinu start, objekta obj, dodeljuje se vrednost osobine currentPage, objekta _controlMediator. Na kraju se kreira i osobina requested, koja sadrži zahtevani broj slogova iz baze podataka, i dodeljuje joj se vrednost iz statičke promenljive StudentListVO.requestNumber. Kada se svi podaci iz TextInput polja smeste u objekat obj, poziva se metod proksija filterStudents, koja uspostavlja konekciju sa serverom.

```
private function onSearchStudentiHandler(event:StudentEvents):void
```

```
{
      //trace("search")
 if (_controlMediator == null)
   _controlMediator = facade.retrieveMediator(StudentControlMediator.NAME)
as StudentControlMediator;
            var obj:Object = new Object;
            obj.fname = view.fname.text;
            obj.address = view.address.text;
            obj.email = view.email.text;

            if (view.departm.selectedItem != null)
                  obj.department = view.departm.selectedItem.id;
            else
                  obj.department = "";

            obj.start = _controlMediator.currentPage;
            obj.requested = StudentListVO.requestNumber;
            proxy.filterStudents(obj);
      }
```

Metod get view vrši kastovanje viewComponent, jer je viewComponent tipa object, a sadrži grafičku komponentu tipa StudentManagerComponent

```
public functionget view():StudentManagerComponent
{
      return viewComponent as StudentManagerComponent;
}
```

Svaki medijator ima ugrađenu funkciju koja se zove listNotificationInterests. Ova funkcija vraća jedan Array. Ovde se registruje lista notifikacija za koje je ovaj medijator zadužen. Praktično ovaj medijator sluša događaje STUDENT_SELECTED, STUDENT_DELETED i STUDENT_ADDED. Kada se desi neka od ovih notifikacija, onda se se pokreće funkcija, koja se zove handleNotification.

```
override public function listNotificationInterests():Array {
      return [
                  ApplicationFacade.STUDENT_SELECTED,
                  ApplicationFacade.STUDENT_DELETED,
                  ApplicationFacade.STUDENT_ADDED
            ];
}
```

handleNotification je funkcija koja služi za obradu notifikacija, registrovanih u metod listNotificationInterests. Svaka notifikacija je tipa INotification, što znači klasa notification primenjuje interfejs INotification. Svaka notifikacija kao objekat poseduje tri bitne informacije Body, Name i Type. U switch bloku se vrši obrada, na osnovu imena notifikacije. Ako je selektovan red u tabeli, onda se dispečuje događaj STUDENT_SELECTED. Ovaj događaj se obrađuje u metod handleNotification. Objektu _selectedStudent, klase StudentManagerComponent, se pristupa pomoću seter funkcije view.selectedtSudent, gde je view grafička komponenta StudentManagerComponent, a selectedtSudent setter funkcija. U liniji koda:

```
view.selectedStudent = notification.getBody() as StudentVO;
```

iz notifikacije se uzima telo getBody(), a to je objekat klase StudentVO, i prosleđuje se kao parameter funkciji **set** selectedStudent.Dakle, kada se selektuje student, poziva se metod medicineGrid_clickHandler, klase StudentTableMediator. U ovoj metodi if služi da dodeli vrednost selectedStudent objektu samo ako je kliknut red u tabeli, u drugim slučajevima se neće ništa desiti. Linijom koda **if** (event.target **is** DefaultGridItemRenderer) proverava se da li je selektovani objekat red u tabeli.

```
protected function medicineGrid_clickHandler(event:MouseEvent):void
{
if (event.target is DefaultGridItemRenderer){
view.selectedStudent =  view.medicineGrid.selectedItem as StudentVO;
sendNotification(ApplicationFacade.STUDENT_SELECTED,view.selectedStudent);
      }
}
```

view.selectedStudent jednak je onome što je selektovano u tabeli. Kada se objekat selectedStudent popuni vrednostima iz reda tabele, šalje se notifikacija STUDENT_SELECTED i prosledi objekat StudentVO koji je selektovan t.j. view.selectedStudent. StudentManagerMediator prihvati notifikaciju STUDENT_SELECTED, i view.selectedStudent je jednako onom, što je stiglo kao telo notifikacije. Tad se pozove funkcija **set** selectedStudent. Onog trenutka kada se izjednači sa =, kao kada se dodeli vrednost promenljivoj, tada je pokrenuta seter funkcija **set** selectedStudent.

```
public functionset selectedStudent(item:StudentVO):void
```

```
{
_selectedStudent = item;
for (var i:int = 0; i < StudentListVO.departmentArray.length; i++){
if (_selectedStudent.department==
StudentListVO.departmentArray.getItemAt(i).id) {
      departm.selectedItem = StudentListVO.departmentArray.getItemAt(i);
            }
      }
}
```

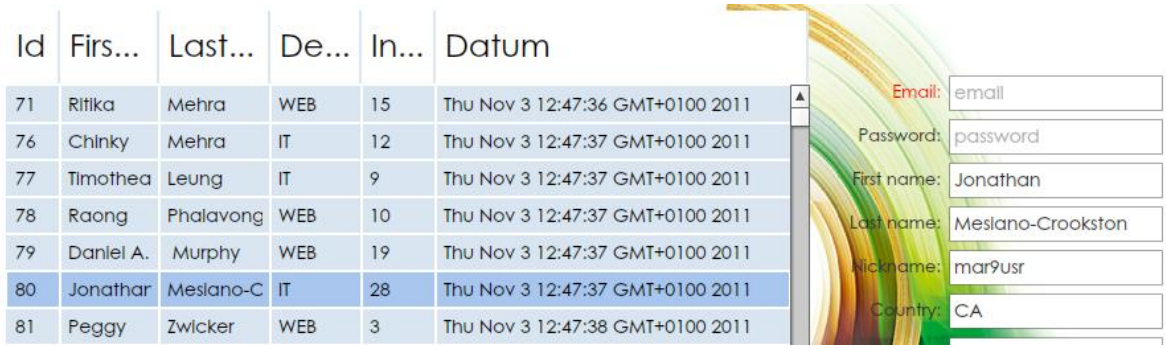
U funkciji selectedStudent privatnoj promenljivoj _selectedStudent, dodeli se vrednost iz item. Objekat depart je ComboBox, kreira se u klasi StudentManagerComponent, konstrukcijom:

```
<s:ComboBox width="70%" id="departm"
dataProvider="{StudentListVO.departmentArray}" labelField="name"/>
```

Ono što se selektuje u ComboBoxu departm.selectedItem jednako je StudentListVO.departmentArray.getItemAt(i), koji zadovoljava uslov **if** (_selectedStudent.department = = StudentListVO.departmentArray.getItemAt(i).id). (_selectedStudent.department je id, i deklarisan je u klasi StudentVO kao int:

```
public var department:int;
```

Proverava se da li je integer _selectedStudent.department jednak integer-u StudentListVO.departmentArray.getItemAt(i).id iz departmentArray. Ako je jednak, onda je departm.selectedItem jednako objektu iz tabele department, jer su podaci iz tabele department smešteni u niz departmentArray. I tako kada se selektuje neki student na IT smeru, u kombo boksu se dobije IT, a kada se selektuje student na WEB smeru u kombo boksu se dobije WEB. Dakle ovde se vrši upoređivanje id-a iz tabele, kolona Department, sa id iz departmentArray, ako je id jedan u tabeli, jednako id-u jedan u departmentArray, primeni taj objekat.



Slika3.7.56. Prikaz rešetke u aplikaciji.

```
override public function
handleNotification(notification:INotification):void {
switch(notification.getName()){
      case ApplicationFacade.STUDENT_SELECTED:
            view.selectedStudent = notification.getBody() as StudentVO;
                  break;
            case ApplicationFacade.STUDENT_DELETED:
                  if (notification.getType() == "manager")
                        trace("ovo je manager");
                              //view.selectedStudent = null;
                              //reset();
                  break;
            case ApplicationFacade.STUDENT_ADDED:
                  if (newStudent){
                        newStudent.id = int(notification.getBody());
                        StudentListVO.StudentiArray.addItem(newStudent);
                        reset();
                  }
                  break;
      }
}
```

Ako je dispečovan event STUDENT_DELETED, onda se u metod handleNotification obrađuje događaj. Proverava se tip događaja if (notification.getType() == "manager"), ako je zadovoljen uslov, vrši se ispis, trace naredbom.

U slučaju klika na osobinu addStudent tipa Button, komponente StudentControlComponent, poziva se metod onAddHandler, klase StudentControlMediator. Untar metode onRegister postavljen je eventListener na dugme addStudent.

```
override public function onRegister():void
{
                  ...
view.addStudent.addEventListener(MouseEvent.CLICK, onAddHandler,false,0,true);
}
```

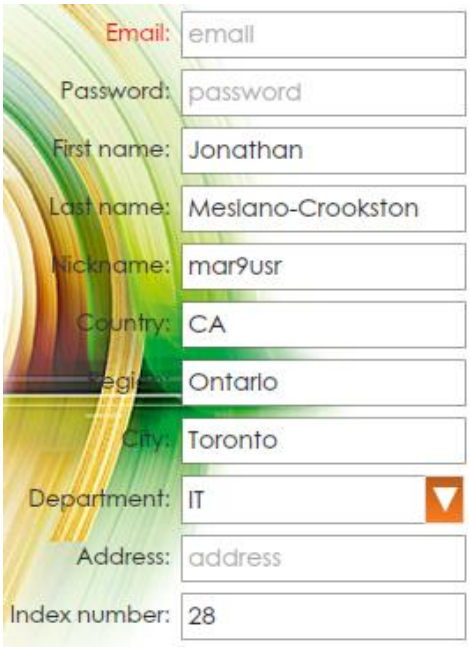
Metod onAddHandler, ima oblik:

```
private function onAddHandler(evt:MouseEvent):void
{
      managerMediator.addStudent();
}
```

U metodi onAddHandler objekat managerMediator klase StudentManagerMediator, pristupa metodi addStudent.

```
public function addStudent():void {
	if (view.fname.text.length > 2 && view.lname.text.length > 2){
		newStudent = new StudentVO();
		newStudent.address = view.address.text;
		newStudent.city = view.city.text;
		newStudent.country = view.country.text;
		newStudent.department = view.departm.selectedItem.id;
		newStudent.email = view.email.text;
		newStudent.first_name = view.fname.text;
		newStudent.index_num = view.indexNum.text;
		newStudent.last_name = view.lname.text;
		newStudent.nickname = view.nname.text;
		newStudent.password = view.pass.text;
		newStudent.region = view.region.text;
		proxy.addStudentiFlex(newStudent);
	}
}
```

U ovoj metodi view vraća grafičku komponentu StudentManagerComponent. U grafičkoj komponenti StudentManagerComponent nalaze se TextInput osobine, kao na slici 3.7.57.



Slika 3.7.57. Prikaz selektovane stavke iz rešetke u TextInput poljima

U metod addStudent, proverava se da li su upisana najmanje dva karaktera u TextIput polja fname i lname. Ako jesu, pristupa se instanciranju objekta newStudent. Svakoj osobini instance newStudent dodeljuje se vrednost iz TextInput polja, kao I iz ComboBox-a. Na kraju se poziva metod u proksiju addStudentiFlex, prosleđujući mu objekat newStudent, koji sadrži sve vrednosti TextInput polja. proxy.addStudentiFlex(newStudent);

Proxi metod addStudentiFlex, uspostavlja konekciju sa serverom, prosleđujući mu vrednosti iz newStudent. Sa server se vraća id poslednjeg upisanog podatka u metod proksija addStudentiFlexResultHandler. Notifikacija STUDENT_ADDED, sa telom event.result, se šalje iz addStudentiFlexResultHandler. U ovom slučaju event.result će imati vrednost id iz poslednjeg unosa u DB.

```
private function addStudentiFlexResultHandler(event:ResultEvent):void
{
removeListeners(addStudentiFlexResultHandler,addStudentiFlexFaultHandler);
	if (event.result == false){
		Alert.show("greska unosa");
	}
	else {
		trace("dodat")

	sendNotification(ApplicationFacade.STUDENT_ADDED,event.result);
	}
}
```

Notifikacijom STUDENT_ADDED rukovodi handleNotification, u delu koda:

```
	if (newStudent){
		newStudent.id = int(notification.getBody());
		StudentListVO.StudentiArray.addItem(newStudent);
		reset();
	}
```

If naredbom se proverava da le je instanciran objekat newStudent. Objekat je instanciran u metodi addStudent i popunjena su sva polja, osim id-a. Sada se telo notifikacije smešta u newStudent.id. Telo notifikacije predstavlja vrednost id iz poslednjeg unosa u DB. Objekat newStudent se dodaje u niz StudentiArray. Na kraju se poziva metod reset, koja popunjava sva polja praznim stringom. Metod reset ima oblik:

```
public function reset():void
{
	view.address.text = "";
```

```
        view.city.text = "";
        view.country.text = "";
        view.departm.selectedIndex = 0;
        view.email.text = "";
        view.fname.text = "";
        view.indexNum.text = "";
        view.lname.text = "";
        view.nname.text = "";
        view.pass.text = "";
        view.region.text = "";
}
```

MetoddeleteSelectedStudent se koristi za brisanje reda iz rešetke. Da bi se red obrisao, mora se najpre selektovati. Selekcijom reda, poziva se prvo metod medicineGrid_clickHandler, klase StudentTableMediator. Ovde, if služi da dodeli vrednost selectedStudent objektu samo ako je kliknut red u tabeli, u drugim slučajevima se neće ništa desiti. U ovaj metod view je komponenta StudentTableComponent a selectedStudent je osobina ove komponente tipa StudentVO. Dakle osobini view.selectedStudent se dodeli selektovani red. Sa sendNotification šalje se notifikacija STUDENT_SELECTED, i telo view.selectedStudent.

```
protected function medicineGrid_clickHandler(event:MouseEvent):void
{
if (event.target is DefaultGridItemRenderer){
view.selectedStudent =  view.medicineGrid.selectedItem as StudentVO;
 sendNotification(ApplicationFacade.STUDENT_SELECTED,view.selectedStudent);
      }
}
```

U metod handleNotification, vrši se obrada događaja STUDENT_SELECTED. Naredbom view.selectedStudent = notification.getBody() as StudentVO;
Poziva se seter metod set selectedStudent u klasi StudentManagerComponent, I kao parameter prosleđuje se telo notifikacije notification.getBody(). U metodi se popunjava ComboBox sa vrednošću.

```
public functionset selectedStudent(item:StudentVO):void
{
      _selectedStudent = item;
      for (var i:int = 0; i < StudentListVO.departmentArray.length; i++){
```

```
            if (_selectedStudent.department==
      StudentListVO.departmentArray.getItemAt(i).id) {
      departm.selectedItem = StudentListVO.departmentArray.getItemAt(i);
            }
      }
}
```

U metod onRegister postavljen je eventListener na osobinu delStudent komponente StudentControlComponent. Kada se posle selekcije reda u rešeci, klikne na osobinu delStudent, klase StudentControlComponent, poziva se metod onDeleteHandler, klase StudentControlMediator. U ovoj metodi poziva se metod deleteSelectedStudent, objekta managerMediator, klase StudentManagerMediator.

```
private function onDeleteHandler(evt:MouseEvent):void
{
      managerMediator.deleteSelectedStudent();
}
```

If naredbom, se proverava da li je objekat selectedStudent popunjen vrednostima, koji se nalazi u objektu klase StudentManagerComponent. Ako je uslov zadovoljen, poziva se metod proksija deleteStudentiFlex I prosleđuje joj se vrednost id elementa u bazi koji se briše.

```
public function deleteSelectedStudent():void
{
      if (view.selectedStudent)
            proxy.deleteStudentiFlex(view.selectedStudent.id);
}
```

Metod editStudent se koristi za promeni vrednosti u selektovanom redu rešetke. U početku se proverava da li je objekat selectedStudent prazan, ako nije, instancira se lokalna instance chMed, klase StudentVO. U chMed upisuju se izmenjene vrednosti iz TextInput polja, i na kraju se poziv metod changeStudentiDataFlex proksija, koji uspostavlja konekciju sa bazom, i rši promenu u bazi.

```
public function editStudent():void {
      if (view.selectedStudent == null) return;
      var chMed:StudentVO = view.selectedStudent;
      chMed.address = view.address.text;
      chMed.city = view.city.text;
      chMed.country = view.country.text;
      chMed.department = view.departm.selectedItem.id;
      chMed.email = view.email.text;
```

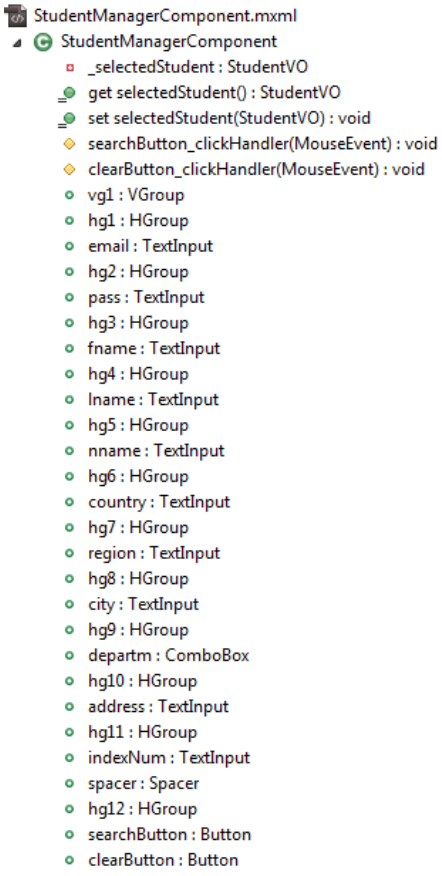
```
        chMed.first_name = view.fname.text;
        chMed.index_num = view.indexNum.text;
        chMed.last_name = view.lname.text;
        chMed.nickname = view.nname.text;
        chMed.region = view.region.text;
        //method izmeni podatke u bazi za selektovani student
        proxy.changeStudentiDataFlex(chMed);
}
```

Metod **get** proxy traži proksi imena, ovoga, što se prosledi kao string

```
public functionget proxy():*
{
        _proxy = facade.retrieveProxy(ApplicationFacade.SELECTED_PROXY);
        return _proxy; }
```

3.7.13 Klasa StudentManagerComponent.mxml

Grafička komponenta StudentManagerComponent sadrži dvadeset i sedam javnih osobina, jednu privatnu osobinu i četiri metode, dve javne i dve zaštićene. Metode i osobine su prikazane na slici 3.7.58.



Slika 3.7.58. Prikaz osobina i metoda za klasu StudentManagerComponent

```
<s:states>
<s:State name="normal" />
<s:State name="search" />
</s:states>

<fx:Script>
<![CDATA[
            . . .
      [Bindable]
      private var _selectedStudent:StudentVO;

      [Bindable]
      public functionget selectedStudent():StudentVO
      {
            return _selectedStudent;
      }

      //21 cas, 1:26 objasnjenje
      public functionset selectedStudent(item:StudentVO):void
      {
           _selectedStudent = item;
      for (var i:int = 0; i < StudentListVO.departmentArray.length; i++){
            if (_selectedStudent.department ==
StudentListVO.departmentArray.getItemAt(i).id){
        departm.selectedItem = StudentListVO.departmentArray.getItemAt(i);
                  }
            }
      }
```

Metod searchButton_clickHandler, proverava se da li je nešto upisano u bilo koje polje TextInput-a email, fname ili address. Ako jeste, onda se trigeruje, događaj pomoću metode dispatchEvent. Događaj koji se trigeruje nosi naziv SEARCH_STUDENTI. Statička konstanta SEARCH_STUDENTI nalazi se u klasi StudentEvents.U dispatchEvent instancira se objekat iventa StudentEvents, i nosi naziv SEARCH_STUDENTI. Sada osluškivač na ovaj događaj reaguje kada se desi, a desit će se, kada se pritisne na dugme search.

```
protected function searchButton_clickHandler(event:MouseEvent):void
   {
      if (email.text.length > 0 || fname.text.length > 0 ||
```

```
address.text.length > 0 || departm.selectedItem != null)
      dispatchEvent(new tudentEvents(StudentEvents.SEARCH_STUDENTI));
    }
```

U metod clearButton_clickHandler, briše se sadržaj iz TextInput polja email, fname i lname, i iz ComboBox-a depart.

```
protected function clearButton_clickHandler(event:MouseEvent):void
   {
      departm.selectedItem = null;
      departm.textInput.text = "";
      email.text = "";
      fname.text = "";
      lname.text = "";
    }
      ]]>
</fx:Script>

<s:VGroup
      width="100%" top="50"
      >
```

HGroup ima dve vrste poravnanja horizontalAlign i verticalAlign. Osobina horizontalAlign vrši poravnanje po horizontali, tekst moža biti na levu stranu poravnat, na desnu i na sredini. Dok osobina verticalAlign vrši poravnanje teksta po vertikali, tako da tekst može biti podignut gore, može biti na dnu HGroup-e i na sredini. Ako se za HGroup ne navede širina, to znači da će se primeniti difoltne vrednosti, njegova širina bila bi nula ili koliko su njagova deca širine. Širina koja se navodi za komponentu TextInput ili Label je relativna širina, računa se u odnosu na roditeljsku širinu. Širina se odnosi na kontejnersku komponentu u kojoj se nalaze osobine. Label je širok po difoltu koliko je tekst širok koji se upiše, a TextInput je po difoltu širok 160px. Dakle, širina HGropu-e bila bi jednaka širini labele plus 160px, koliko je širina TextInput polja. TextInput polja se popunjavaju iz objekta _selectedStudent. Naredbom text="{selectedStudent.email}" poziva se get selectedStudent() funkcija, koja vraća objekat _selectedStudent. U objektu se pristupa osobini email, i njen sadržaj se kopira u osobinu text objekta email. Na isti način se popunjavaju i druge kontrole iz objekta _selectedStudent. U ovoj klasi definisana su dva stanja "normal" i "search". Ako nijedno stanje nije definisano u kontroli, kontrola se dodaje u oba stanja. U oba stanja dodaje se ceo prvi HGroup kontejner, drugi HGroup kontejner se dodaje u stanju normal, treći HGroup kontejner se vidi u oba stanja,

ali osobina text ima vrednost "First name:" u stanju normal i "First or Last name:" u stanju search, četvrti, peti, šesti, sedmi, osmi i deveti, deseti i jedanaesti HGroup kontejner se vidi u stanju normal. Stil labelText primenjuje se na labelu, koji je definisan u style.css fajlu.

```
<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
      <s:Label text="Email:" width="40%" styleName="labelText"
color="0xff0000"/>
<s:TextInput width="70%" id="email" text="{selectedStudent.email}"
prompt="email"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Password:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="pass" prompt="password"
displayAsPassword="true"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
      <s:Label text.normal="First name:" text.search="First or Last name:"
      width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="fname"
text="{selectedStudent.first_name}" prompt="first name"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Last name:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="lname"
text="{selectedStudent.last_name}" prompt="last name"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Nickname:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="nname" text="{selectedStudent.nickname}"
prompt="nickname"/>
</s:HGroup>
```

```
<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Country:" width="40%" styleName="labelText" />
      <s:TextInput width="70%" id="country"
text="{selectedStudent.country}" prompt="country"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Region:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="region" text="{selectedStudent.region}"
prompt="region"/>
</s:HGroup>
<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="City:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="city" text="{selectedStudent.city}"
prompt="city"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
      <s:Label text="Department:" width="40%" styleName="labelText"/>
<s:ComboBox width="70%" id="departm"
dataProvider="{StudentListVO.departmentArray}" labelField="name"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
      <s:Label text="Address:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="address"
text="{selectedStudent.address}" prompt="address"/>
</s:HGroup>

<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%"
includeIn="normal">
      <s:Label text="Index number:" width="40%" styleName="labelText"/>
      <s:TextInput width="70%" id="indexNum"
text="{selectedStudent.index_num}" prompt="index number"/>
</s:HGroup>
```

Kontrola Spacer, pravi razmak 20px od drugih polja po vertikali.

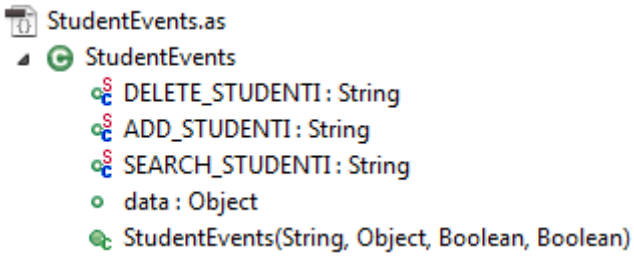
```
<s:Spacer height="20" />
```

Ovde su definisana dve kontrole searchButton i clearButton tipa Button, koje su vidljive, kada je stanje komponente StudentManagerComponent search. Klikom na kontrolu searchButton, poziva se metod searchButton_clickHandler, a klikom na dugme clearButton, poziva se metod clearButton_clickHandler.

```
<s:HGroup
      width="100%"
      <s:Button
            includeIn="search"
            id="searchButton"
            label="Search"
            click="searchButton_clickHandler(event)"
            />
      <s:Button
            includeIn="search"
            id="clearButton"
            label="Clear"
            click="clearButton_clickHandler(event)"
            />
</s:HGroup>
</s:VGroup>
</s:Group>
```

3.7.14 Klasa StudentEvents.as

U klasi StudentEvents definisana su tri događaja. Klasa sadrži tri statičke konstante, jednu javnu osobinu i jednu metodu. Struktura klase je prikazana na slici3.7.59.



Slika 3.7.59. Prikaz osobina i metoda za klasu StudentEvents

DELETE_STUDENTI je ime događaja unique za ovu klasu, a "deleteStudenti" je njegova vrednost i mora biti unique na nivou aplikacije.

```
public class StudentEvents extends Event
{
	public static const DELETE_STUDENTI:String = "deleteStudenti";
	public static const ADD_STUDENTI:String = "addStudenti";
	public static const SEARCH_STUDENTI:String = "searchStudenti";
	public var data:Object;
data je tipa Object, što znači da se u data može upisati vrednost  bilo
kojeg tipa i proizvoljan broj osobina može sadržati.
	public function StudentEvents(type:String, _data:Object = null,
bubbles:Boolean=false,
	cancelable:Boolean=false)
	{
			super(type, bubbles, cancelable);
			this.data = _data;
		}
	}
```

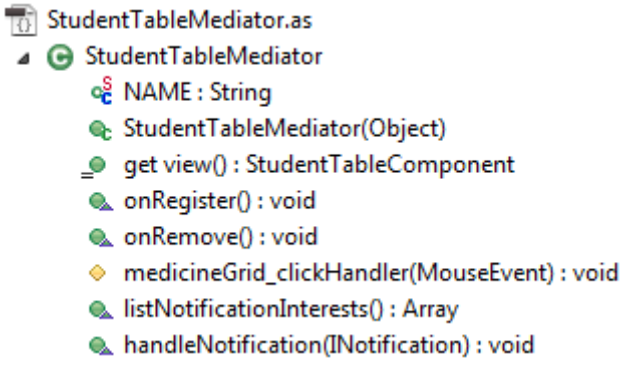
Ovde se koriste tri događaja DELETE_STUDENTI, ADD_ STUDENTI i SEARCH_STUDENTI, koji su konstante i ne mogu se menjati. Konstante su statičke, što znači da im se može pristupiti pomoću klase.

3.7.15 Klasa StudentTableMediator.as

Klasa StudentTableMediator sadrži jednu statčku konstantu, tipa string i sedam metoda. Konstanta NAME sarži ime instance, klase StudentTableMediator.

```
public static const NAME:String = "StudentTableMediator";
```

Metode klase i konstanta su prikazani na slici3.7.60.



Slika 3.7.60. Prikaz osobina i metoda za klasu StudentTableMediator

Kada se instancira objekat klase StudentTableMediator u klasi ViewStartupCommand, poziva se konstruktor funkcija StudentTableMediator, kojoj se prosleđuje grafička komponenta StudentTableComponent u parameter viewComponent tipa Object.

```
public function StudentTableMediator( viewComponent:Object)
{
	super(NAME, viewComponent);
}
```

Metod get view vrši kastovanje, jer viewComponent tipa Object prihvata grafičku komponentu. Dakle get view vraća instance tipa StudentTableComponent. viewComponent je ugradjen u klasi Mediator koju je ova klasa nasledila.

```
public functionget view():StudentTableComponent
{
	return viewComponent as StudentTableComponent;
}
```

Metod onRegister poziva se pri registraciji instance klase StudentTableMediator, u ViewStartupCommand. MouseEvent.CLICK je ivent koji ima osobina view.medicineGrid. U slučaju klika na view.medicineGrid, poziva se funkcija medicineGrid_clickHandler.

```
override public function onRegister():void
{
	view.medicineGrid.addEventListener(MouseEvent.CLICK,
medicineGrid_clickHandler);
}
```

Metod onRemove uklanja slušaoc događaja sa osobine view.medicineGrid.

```
override public function onRemove():void
{
	view.medicineGrid.removeEventListener(MouseEvent.CLICK,
medicineGrid_clickHandler);
}
```

U ovoj metodi if služi da dodeli vrednost selectedStudent objektu samo ako je kliknut red u tabeli, u drugim slučajevima se neće ništa desiti. Posle dodele vrednosti iz tabele, šalje se notifikacija. Notifikacija sadrži naziv eventa STUDENT_SELECTED i telo view.selectedStudent.

```
protected function medicineGrid_clickHandler(event:MouseEvent):void
{
if (event.target is DefaultGridItemRenderer){
      view.selectedStudent = view.medicineGrid.selectedItem as StudentVO;

sendNotification(ApplicationFacade.STUDENT_SELECTED,view.selectedStudent);
      }
}
```

Prepisana metod listNotificationInterests registruje događaj STUDENT_DELETED.

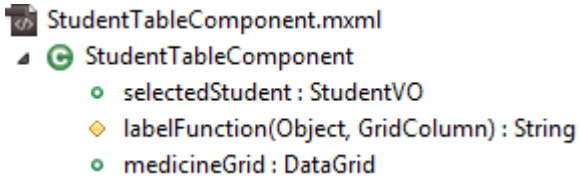
```
override public function listNotificationInterests():Array {
      return [
            ApplicationFacade.STUDENT_DELETED
      ];
}
```

Iz tabele je izbrisan selektovani student. Student se briše, slanjem notifikacije iz proksija, Naredbom *sendNotification(ApplicationFacade.STUDENT_DELETED, null, "student");*. Notifikacija se registruje u metodi listNotificationInterests a obrađuje u metod handleNotification.

```
override public function
handleNotification(notification:INotification):void {

switch(notification.getName()){
      case ApplicationFacade.STUDENT_DELETED:
            if (view.selectedStudent)
            StudentListVO.StudentiArray.removeItem(view.selectedStudent);
                  break;
      }
}
```

3.7.16 Klasa StudentTableComponent.mxml

Grafička komponenta StudentTableComponent sadrži dve javne osobine I jednu protected metodu. Struktura klase prikazana je na slici3.7.61.



Slika 3.7.61. Prikaz osobina i metoda za klasu TableComponent

Klasa je opisana, narednim linijama koda.

```
<fx:Script>
<![CDATA[

[Bindable]
public var selectedStudent:StudentVO;
protected function labelFunction(item:Object, column:GridColumn):String {
var value:String = "";
      for (var i:int = 0; i < StudentListVO.departmentArray.length; i++){
                  if (item.department ==
StudentListVO.departmentArray.getItemAt(i).id){
      value = StudentListVO.departmentArray.getItemAt(i).name;
      }
}
      return value;
}
]]>
</fx:Script>
```

dataProvider je osobina DataGrid-a, i služi da popuni rešetku podacima. U ovom slučaju podaci koji pune rešetku smešteni su u statički niz StudentiArray, koji je definisan u ActionScript klasi StudentListVO.as. S obzirom da statički niz pripada klasi a ne objektu, to se može ovom elementu pristupiti preko imena klase, ne čekajući da se instancira objekat t.j. StudentListVO.StudentiArray. Rešetka se sastoji kolona, koje se definišu sa parom tagova <s:columns> i </s:columns>. Svaka kolona je tipa ArrayList. U DataGrid-u definiše se ponaosob svaka kolona. Kolona ima dve osobine: headerText="Opis" definiše naziv kolone, a headerText definiše odakle se puni kolona. headerText="description" ovde inicira, da se kolona puni iz polja description, ko se nalazi u nizu StudentListVO.StudentiArray. Da bi se polje u tabeli popunilo, neophodno je u osobini dataField, klase GridColumn, navesti naziv osobine objekta, iz kojeg se čita podatak. Međutim, za kolonu Dipartment ako bi napisali dataField="dipartment", dobili bi brojeve, jer je dipartment u bazi predstavljena sa brojevima. Zato je način popunjavanja ove kolone promenjen. Ovde je potreban mehanizam da se id, koji

se zove department, pretvori u natpis. U tu svrhu postoji nešto, što se zove labelFunction. labelFunction je funkcija koja omogućava da se obradi podatak <s:GridColumn headerText="Department" labelFunction="labelFunction" /> koji je prosleđen određenoj koloni, i kao rezultat te obrade, neophodno je dobiti čitljiv rezultat. Funkcija ima svoje zaglavlje, koje je uvek isto. Zaglavlje funkcije labelFunction(item:Object, column:GridColumn):String zahteva da se prosledi objekat koji je trenutan u redu i kolonu. Objekat koji se šalje je iz provajdera StudentiArray. Za prvi red to je objekat getItemAt(0). Red tabele je jedan objekat iz provajdera StudentListVO.StudentiArray. Kolona koja se prosleđuje je "Department" kolona. U telu se pravi lokalna promenljiva value, tipa string, i trenutna vrednost stringa je prazan string.

Po definiciji, funkcija vraća string, i na kraju se vraća value. U slučaju da se ništa ne desi, vratiće se prazan string. Ovde postoji departmentArray, koji sadrži listu svih department-a, pričitanih iz baze podataka department. departmentArray se popunjava u proksiju, koristeći metode. Metod getDepartments je funkcija koja se nalazi na serveru u okviru klase StudentiAPI.

```
public function getDepartments():void {
addListeners(getAllDepartmentResultHanlder,addStudentiFlexFaultHandler,"Stu
dentiAPI","StudentiAPI");
_myConnection.getDepartments();
}
```

Ovde se departmentArray popunjava podacima iz baze.

```
private function getAllDepartmentResultHanlder(event:ResultEvent):void {
if (event.result != null){
      StudentListVO.departmentArray = new ArrayList(event.result as Array);
      }
}
```

U proksiju, unutar metode onRegister se poziva metod getDepartments.

```
override public function onRegister():void
{
      getDepartments();
}
```

Znači onog trenutka, kada se registruje proksi, poziva se funkcija getDepartments. Funkcija getDepartments poziva bazu podataka linijom koda _myConnection.getDepartments();. Metod getAllDepartmentResultHanlder prihvata podatke iz baze podataka i puni niz

departmentArray. U for petlji, vrši se iteracija kroz sve elemente departmentArray. item je objekat klase StudentVO, i sadrži podatke za jedan red.

U if naredbi proverava se ako je item.department jednak sa trenutnim elementom departmentArray.getItemAt(i).id iz iteracije, onda se u value setuje name iz objekta getItemAt(i), i onda se pošalje value odakle je pozvana fukcija.

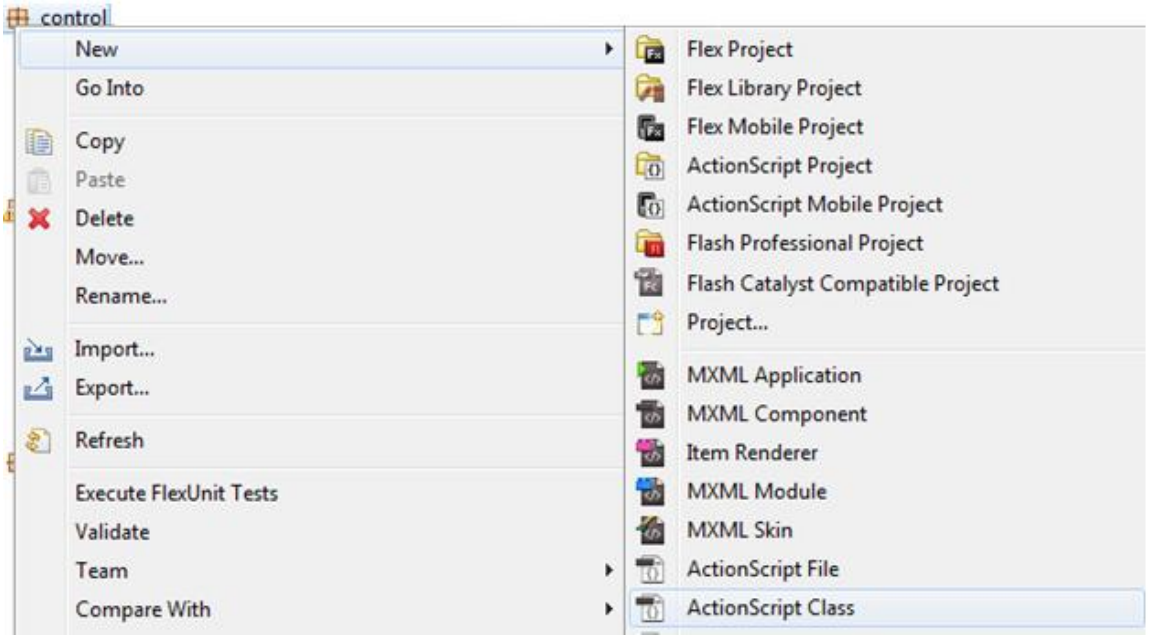
Slika 3.7.62. Tabela department u bazi

Iz ove tabele, u bazi podataka se uzima name i id. labelFunction se primenjuje na celu kolonu Department u DataGrid, za svaki red posebno. Praktično šalje se red i kolona u labelFunction, red se smešta u item:Object, a kolona u column:GridColumn. labelFunction je korisnička funkcija. Department je kolona u rešeci.

```
<s:DataGrid
	id="medicineGrid"
	width="100%" height="100%"
	dataProvider="{StudentListVO.StudentiArray}"
	top="0" bottom="0"
	>
<s:columns>
<s:ArrayList>
	<s:GridColumn headerText="Id" dataField="id" width="40"/>
	<s:GridColumn headerText="First Name" dataField="first_name" />
	<s:GridColumn headerText="Last Name" dataField="last_name" />
	<s:GridColumn headerText="Department" labelFunction="labelFunction"
/>
	<s:GridColumn headerText="Index no." dataField="index_num" />
	<s:GridColumn headerText="Datum" dataField="creationDate" />
	</s:ArrayList>
</s:columns>
</s:DataGrid>
</s:Group>
```

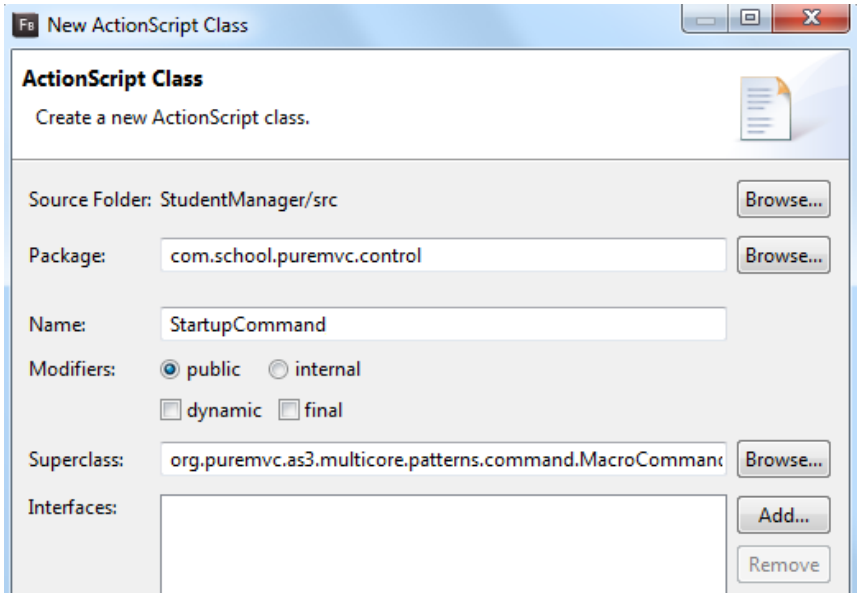
3.7.17 Kontroler StartupCommand

StartupCommand je kontroler klasa. Kontroleri izdaju komande, setuju proksije, medijatore, upravljaju aplikacijom, izdaju naređenja pojedinim delovima aplikacije. Klasa StartupCommand nalazi se u paketu control. Kreira se selekcijom paketa control i klikom na desno dugme miša formira se meni.



Slika3.7.63. Izbor opcije ActionScript Class iz Menia, za kreiranje StartupCommand-e

Iz menija se odabere stavka ActionScript Class, usled čega se otvara dijalog prozor.



Slika3.7.64. Dijalog za definisanje imena klase

Klasa se zove StartupCommand. Ovo je komanda koja će se pokrenuti na samom startu pa je logično da se nazove StartupCommand, i ako može da se nazove bilo kako. Ova klasa treba da nasledi što se tiče kontrolera ili komande, komandu mvc. Postoji dve vrste kontrolera ili komandi u MVC-u, jedna je Macrocommand a druga je singlCommand. Sada je napravljena jedna klasa, koja je extends MacroCommand.

Bitno je naglasiti da klasa StartupCommand koja je nasledila sadržaj klase MacroCommand, može imati samo jednu jedinu funkciju, a to je *initializeMacroCommand***,** koja je ugrađena ali se overrajduje. U početku prvo se poziva startup funkcija klase ApplicationFacade. Startup funkcija ove klase šalje notifikaciju. Opserver notifikacije je ugrađen u klasu Fasada(Façade), i sluša sve te komande notifikacije. Kada prepozna ko sluša STARTUP komandu, on odradi to.

```
package com.school.puremvc.control
{
	import org.puremvc.as3.multicore.patterns.command.MacroCommand;
	public class StartupCommand extends MacroCommand
	{
		public function StartupCommand()
		{
			super();
		}

		overrideprotected function initializeMacroCommand():void
		{
			addSubCommand(ModelStartupCommand);
			addSubCommand(ViewStartupCommand);
		}
	}
}
```

U ovoj klasi prvo se registruju podaci ModelStartupCommand pa onda komponente ViewStartupCommand, jer ViewStartupCommand koristi podatke, ako ih nema izavaće grešku. Ovde se odvija asinhroni metod rada, prvo se izvršava addSubCommand(ModelStartupCommand); ali se ne isprate svi koraci do kraja već kada se pošalje zahtev proksiju, i čeka se odgovor od servera, nastavlja se sa izvršavanjem donje naredbe addSubCommand(ViewStartupCommand);. Dakle, ovde se dodaju dve komande za izvršenje. Jedna je ModelStartupCommand a druga je ViewStartupCommand.

3.7.18 Kontroler ModelStartupCommand

Komanda ModelStartupCommand se automatski izvršava, čim je navedena u klasi StartupCommand, naredbom addSubCommand(ModelStartupCommand);
Klasa ModelStartupCommand je tipa SimpleCommand, koja ima u sebi ugrađenu funkciju execute. Ovde se override-uje funkcija. Metod execute se izvršava onog trenutka kada se pozove SimpleCommand. SimpleCommand se pozvala u liniji koda addSubCommand(ModelStartupCommand). Može se pozvati i tako što se registruje putem notifikacije. Naime, pozivom ModelStartupCommand, odmah se izvršava metod execute. Ovde facade tipa je IFacade, je zajednički objekat. IFacade je ugrađen u modele, kontrolere i medijatore. Praktično IFacade je tipa ApplicationFacade. facade ima registerProxy funkciju, koja služi da registruje Proxy klasu. Ovde se registruju klase StudentProxyAmf, StudentProxyJson, StudentProxyXml, StudentProxyJavaJson i StudentProxyJavaXml, koje su napravljene. Kada su registrovane instance proksija, onda je deklarisana promenljiva studentProxy koja može da prihvati vrednost bilo kojeg tipa. Metod retrieveProxy pronalazi objekat iz liste objekata, koji je prethodno facade motodom registerProxy registrovan.

```
facade.registerProxy(new StudentProxyAmf());
```

znači

```
facade.children.push(StudentProxyAmf),
```

facade poseduje array children, u koji se naredbom push smešta instanca od StudentProxyAmf. Dakle, sve što se registruje ovde, pakuje se u neki niz. Kasnije kada se desi neki event, kao u narednom primeru, gde se vrši iteracija kroz sve child-ove, i proverava se svaki od njih, da li u listener osluškuje događaj event.STARTUP, ako osluškuje, u osobinu data od child(i) smešta se telo notifikacije, t.j. event.StudentAMF.

```
var event:NotificationEvent = new NotificationEvent(STARTUP,StudentAMF);
            for (var i:int=0;i < application.getNumOfChildren;i++){
                  if (application.getChild(i).listener == event.STARTUP){
                        application.getChild(i).data = event.StudentAMF;
      }
}
```

Metod registerProxy stavlja u listu instancirani objekat, a retrieveProxy nalazi objekat u listi po njegovom imenu, koji se prosleđuje kao parametar ovoj metodi. Proksiju nije prosleđeno telo pri instanciranju, već samo ime, zato što objekat ptoxy klase, radi obradu podataka. Svaku proksi koji je registrovan funkcijom registerProxy, može se u svakom momentu obrisati i ponovo napraviti objekat iste klase StudentProxyAmf i može se ponovo registrovati. Ovde je napravljeno više objekata proksija StudentProxyAmf, StudentProxyJson, StudentProxyXml, StudentProxyJavaJson, StudentProxyJavaXml i registrovani su u istoj fasadi. Dakle, ostavljeno je pravljenje više proksija ali drugog imena. I opet su to samo Singlton-i, jer ako se pokuša napraviti ponovo drugi objekat iste klase, i prosledi se isto ime, samo će vratiti prvi objekat. Neće napraviti drugi objekat, već on će iz liste vratiti prvi objekat, a to se rešava linijama koda:

```
if (instanceMap[key] == null)
      instanceMap[key] = new ApplicationFacade(key);
return instanceMap[key] as ApplicationFacade;
```

Ovde je definisano, ako ne postoji objekat pod imenom key, vrati novi a ako postoji vrati stari. Naime, u mvc-u postoje dve vrste komandi MacroCommand i SimpleCommand. MacroCommand je komanda, koja može da izvrši više komandi u isto vreme, kao što je slučaj u klasi StartupCommand, gde se izvršavaju dve komande u isto vreme:

```
addSubCommand(ModelStartupCommand);
addSubCommand(ViewStartupCommand);
```

Funkcija addSubCommand služi da pokrene više komandi u isto vreme, tako što joj se dodele klase ModelStartupCommand i ViewStartupCommand. ModelStartupCommand je SimpleCommand. SimpleCommand znači jednostavna komanda, koja može da ima samo jednu komandu execute, jer se pokreće funkcijom execute. Dakle, funkcija execute se izvršava samo jedanput, i zato je SimpleCommand. SimpleCommand može se dodeliti ime notifikacije koju će slušati. Kontroleri služe za izvršenje samo jedne komande.

```
package com.school.puremvc.control
{
      ...
public class ModelStartupCommand extends SimpleCommand
     {
      override public function execute(notification:INotification):void {
            trace("model je startovan")
```

```
            // ovde se registruje proksi
            facade.registerProxy(new StudentProxyAmf());
            facade.registerProxy(new StudentProxyJson());
            facade.registerProxy(new StudentProxyXml());
            facade.registerProxy(new StudentProxyJavaJson());
            facade.registerProxy(new StudentProxyJavaXml());
var studentProxy:*=facade.retrieveProxy(ApplicationFacade.SELECTED_PROXY);
            studentProxy.getAllStudentiFlex(0);
            // ovde je iz komande pokrenult proksi
            }
      }
}
```

3.7.19 Kontroler ViewStartupCommand

ViewStartupCommand je SimpleCommand, što znači da se funkcija execute izvršava samo jednom. Klasa ViewStartupCommand je singleton klasa. Naime, ako je u pitanju singleton klasa onda se samo prvi put alocira memorija a svaki naredni put se ne alocira već se dodaje stari objekat, objekat koji već postoji u nekom okruženju, promenljivoj koji je prethodno alociran.

U liniji koda var app:StudentAMF = notification.getBody() as StudentAMF; pravi se promenljiva app tipa StudentAMF, i jednaka je notification.getBody() as StudentAMF. U ovom momentu notifikacija nosi objekat StudentAMF. Ova notifikacija je poslata iz metode startup, klase ApplicationFacade, u liniji koda sendNotification(STARTUP,value);. sendNotification ima STARTUP i value, a value je StudentAMF. sendNotification je startovao komandu STARTUP, StartupCommand, registrovana naredbom registerCommand (STARTUP,StartupCommand); koja se nalazi u metod initializeController, klase ApplicationFacade. U notifikaciji telo objekta je value, a naziv STARTUP. Dakle,na isti poziv, notifikaciju, odreagovala je klasa ViewStartupCommand, kao što je pre ove klase odreagovao kontroler StartupCommand.

Notifikacija u sebi ima getBody, a to je način kako da se izvuče value. Naime naredbom: var app:StudentAMF = notification.getBody() as StudentAMF; je rečeno getBody je ono što je poslato, a to je tipa StudentAMF, zato se sa as sugeriše, da se getBody() tretira kao StudentAMF. Objekat app služi, da bi mogli da se registruju medijatori. Medijatori zahtevaju u konstruktoru objekat sa kojim će da rade. Svi objekti u ovoj aplikaciji nalaze se unutar

instance klase StudentAMF. Pri registraciji medijatora, kao parametar, šalje se onaj objekat sa kojim se radi.

U narednim linijama koda registruju se medijatori. Kada se kaže new, poziva se konstruktor funkcija medijatora i prosleđen je konstruktor funkciji objekat. To je vizuelni objekat, sa kojim medijator radi. U konstruktor funkciji, poziva se konstruktor osnovne klase, prosledi se jedinstveno ime medijatora i komponenta sa kojom komunicira. Promenljiva viewComponent tipa Object nalazi se u paketu Mediator. Ovde se nasleđuje medijator, kada se nasledi medijator nasleđuje se kompletan sadržaj, metode njegove i osobine.

U liniji koda super(NAME, _viewComponent); vrši se dodela promenljive _viewComponent promenljivoj viewComponent. Dakle, medijator StudentTableMediator radi sa app.tableStudent objektom, StudentManagerMediator medijator radi sa app.managerStudent objektom, StudentControlMediator radi sa app.controlStudent i medijator ApplicationMediator radi sa app objektom. tableStudent objekat je registrovan unutar StudentAMF-a, naredbom: <gui:StudentTableComponent id="tableStudent" width="70%" height="90%"/>. StudentTableComponent je tabela. Zatim, imamo drugu komponentu unutar aplikacije StudentManagerComponent, i ona se registruje naredbom facade.registerMediator(new StudentManagerMediator(app.managerStudent));, to su TextInput polja. Treća komponenta koja se registruje je controlStudent, a to su Button-i.

Ovde StudentControlComponent je klasa a objekat je controlStudent, to se definiše konstrukcijom: <gui:StudentControlComponent id="controlStudent" />. Objekat controlStudent se registruje u trećoj iniji koda. U niz objekta facade, upisuju se objekti za četiri medijatora.

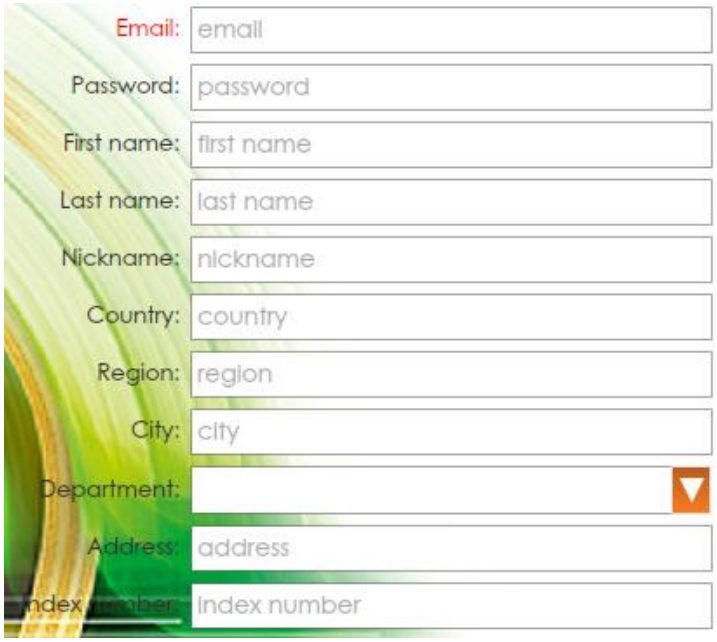
```
public class ViewStartupCommand extends SimpleCommand
{
override public function execute(notification:INotification):void {
      trace("mediator command je registrovan")
      var app:StudentAMF = notification.getBody() as StudentAMF;
facade.registerMediator(new StudentTableMediator(app.tableStudent));
facade.registerMediator(new StudentManagerMediator(app.managerStudent));
facade.registerMediator(new StudentControlMediator(app.controlStudent));
facade.registerMediator(new ApplicationMediator(app));
          }
}
```

Naime, u drugoj liniji koda, registruje se medijator StudentControlMediator. Grafička komponenta app.tableStudent je prikazana na sledećoj slici 3.7.65.

Id	First Name	Last Name	Department	Index no.	Datum
71	Ritika	Mehra	WEB	15	Thu Nov 3 12:47:36 GMT+0100 2011
76	Chinky	Mehra	IT	12	Thu Nov 3 12:47:37 GMT+0100 2011
77	Timothea	Leung	IT	9	Thu Nov 3 12:47:37 GMT+0100 2011
78	Raong	Phalavong	WEB	10	Thu Nov 3 12:47:37 GMT+0100 2011
79	Daniel A.	Murphy	WEB	19	Thu Nov 3 12:47:37 GMT+0100 2011
80	Jonathan	Meslano-Crookston	IT	28	Thu Nov 3 12:47:37 GMT+0100 2011
81	Peggy	Zwicker	WEB	3	Thu Nov 3 12:47:38 GMT+0100 2011
94	Jill	Yates	WEB	22	Thu Nov 3 14:38:14 GMT+0100 2011
95	Sarah	Woods	IT	32	Thu Nov 3 14:38:14 GMT+0100 2011
96	James A.	Woods	WEB	31	Thu Nov 3 14:38:14 GMT+0100 2011

Slika 3.7.65. Grafička komponenta app.tableStudent

U trećoj liniji koda registruje se medijator StudentManagerMediator, a kao parameter instance ovog medijatora dodeljuje se grafička komponenta app.managerStudent, t.j. sfera interesovanja ovog medijatora. Grafička komponenta app.managerStudent data je na slici 3.6.89, kada je stanje normal, t.j. <s:State name=" normal "/>. U slučaju da je stanje search t.j. <s:State name="search"/>, Grafička komponenta app.managerStudent ima izgleda kao na slici3.7.66.



Slika 3.7.66. Grafička komponenta app.managerStudent

Slika 3.3.7.67. Deo aplikacije za pretragu

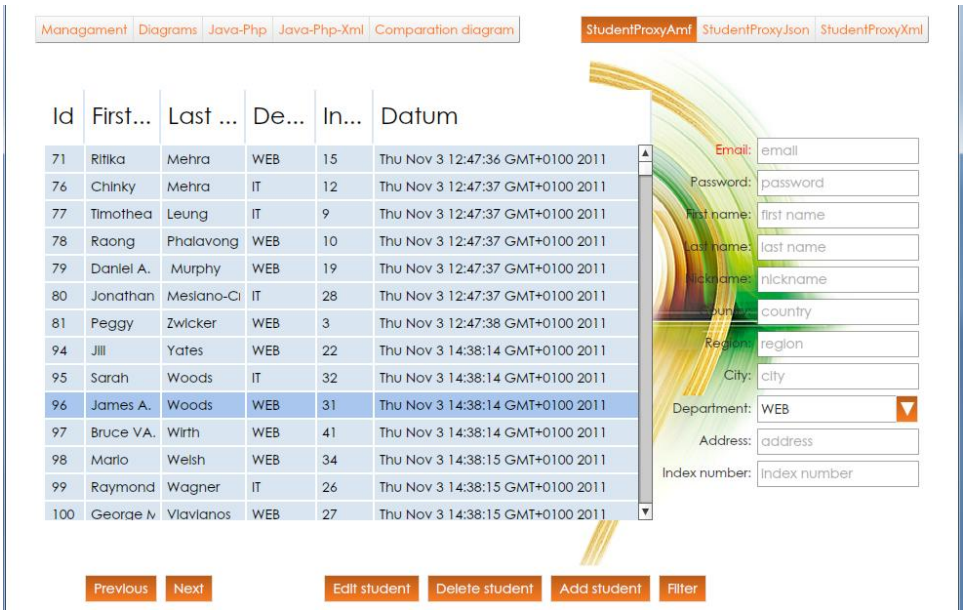
U četvrtoj liniji koda registruje se mediator StudentControlMediator, čija je sfera interesovanja, grafička komponenta app.controlStudent. Grafička komponenta app.controlStudent ima izgled u stanju normal, t.j. <s:State name="normal"/>,

Slika 3.7.68. Grafička komponenta app.controlStudent u stanju normal

Dok u stanju search t.j. <s:State name="search"/>, grafička komponenta app.controlStudent izgleda:

Slika 3.7.69. Grafička komponenta app.controlStudent u stanju search

U petoj liniji koda registruje se mediator glavne aplikacije ApplicationMediator, čija sfera interesovanja je grafička komponenta app t.j. StudentAMF.mxml, čiji izgled je dat na slici 3.7.70.



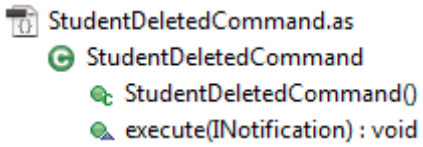
Slika 3.7.70. Komponenta StudentAMF.mxml

3.7.20 Kontroler StudentDeletedCommand

Ovaj kontroler sadrži dve metode:

- *Konstruktor funkcija StudentDeletedCommand*
- *Execute*

Struktura ove klase može se grafički predstaviti:



Slika 3.7.71. Struktura klase StudentDeletedCommand

Kod klase ima oblik:

```
package com.school.puremvc.control
{
                ...
public class StudentDeletedCommand extends SimpleCommand
      {
            public function StudentDeletedCommand()
            {
                  super();
            }

override public function execute(notification:INotification):void
{
trace("student je obrisan")
var mediator:StudentManagerMediator=
facade.retrieveMediator(StudentManagerMediator.NAME) as
StudentManagerMediator;
            mediator.reset();
            }
      }
}
```

Kontroler StudentDeletedCommand se pokreće na događaj STUDENT_DELETED, koji registruje metod initializeController klase ApplicationFacade, linijom koda:
registerCommand(STUDENT_DELETED,StudentDeletedCommand);

Naime, kada se desi STUDENT_DELETED, pokreće se komanda StudentDeletedCommand. StudentDeletedCommand služi samo da napravi delete. Notifikacija STUDENT_DELETED se dispečuje u metod deleteStudentiFlexResultHandler, klase StudentProxyAmf. Ova metod ima strukturu:

```
private function deleteStudentiFlexResultHandler(event:ResultEvent):void{
```

```
removeListeners(deleteStudentiFlexResultHandler,deleteStudentiFlexFaultHand
ler);
sendNotification(ApplicationFacade.STUDENT_DELETED,null,"student");
}
```

Metod deleteStudentiFlexResultHandler se poziva, klikom na dugme Delete student u aplikaciji (slika 3.7.34.).

Grafički proces se može predstaviti.

StudentProxyAmf. deleteStudentiFlexResultHandler

```
sendNotification(ApplicationFacade.STUDENT_DELETED,null,"student");
```

↓

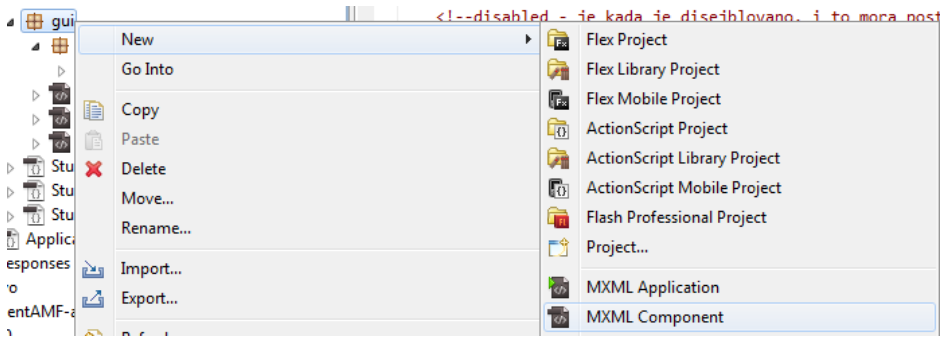
```
registerCommand(STUDENT_DELETED,StudentDeletedCommand);
```

↓

```
var mediator:StudentManagerMediator = facade.retrieveMediator(StudentManagerMediator.NAME)
as StudentManagerMediator;
mediator.reset();
```

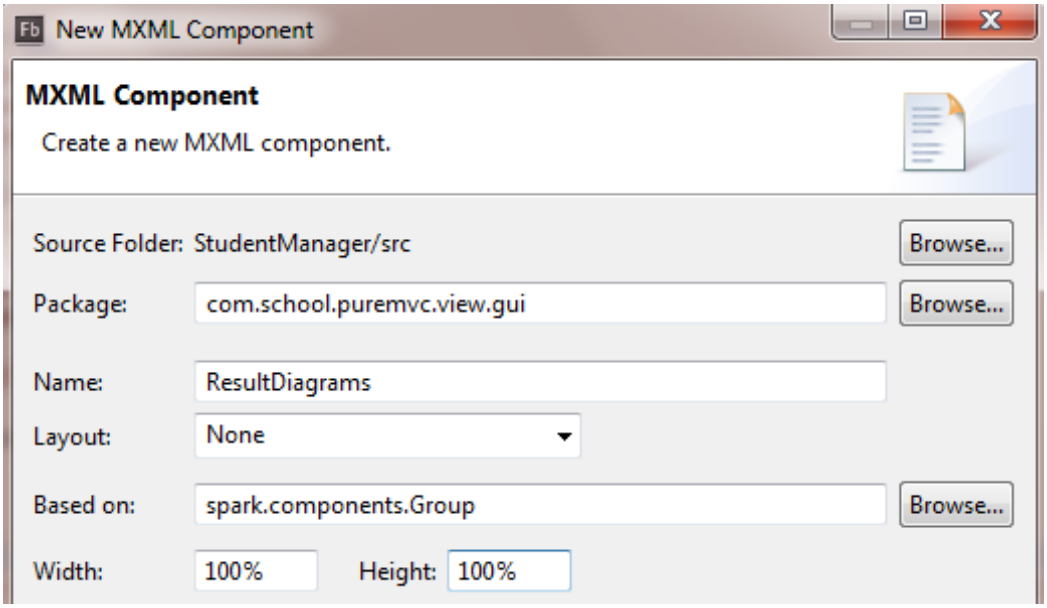
3.7.21 Klasa ResultDiagrams.MXML

Ovde se kreira jedan grafički interfejs t.j. grafička komponenta, koja se smešta je u paket gui. Komponenta se pravi, desnim klikom miša na paket gui, odabirom stavke new i na kraju u pomoćnom meniju bira se stavka MXML Component, što se vidi na narednoj slici3.7.72.



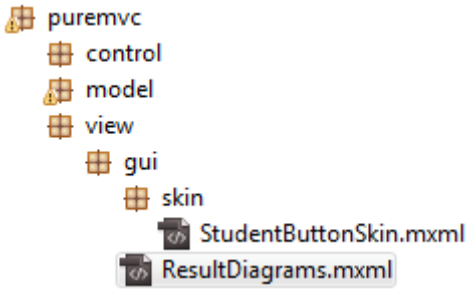
Slika 3.7.72. Prikaz menija za izbor grafičke komponente ResultDiagrams.mxml

Sada se otvara prozor za dijalog, u polje name upisuje se naziv grafičke mxml komponente, *ResultDiagrams*. Širina i visina je postavljena na 100%. Ovde je vrednost za širinu i visinu relativna, jer su dimenzije postavljene u odnosu na roditeljsku klasu. Dakle, dimenzije komponente se saopštavaju roditelju, jer ako mu se ne kaže, on neće primeniti to.



Slika 3.7.73. Dijalog prozor za definisanje mxml komponente ResultDiagrams

Struktura paketa u PackageExplorer-u je prikazana na slici 3.7.74.

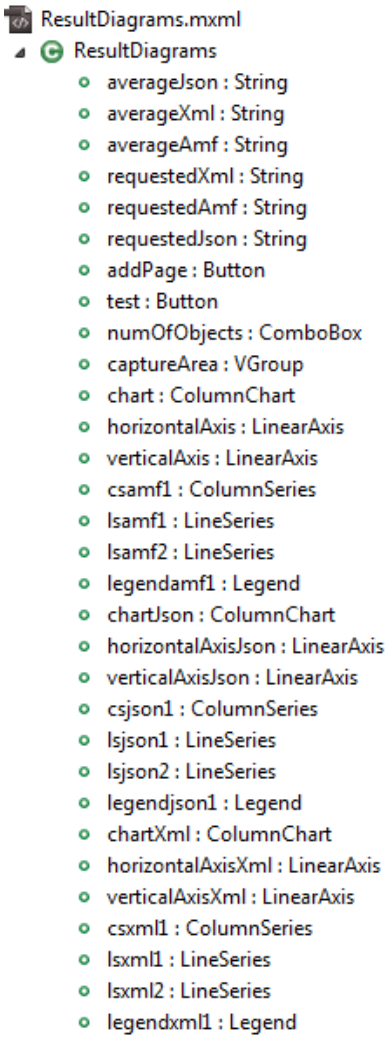


Slika 3.7.74. Struktura paketa u PackageExplorer

Grafički interfejs ResultDiagrams.mxml pravi se u glavnoj aplikaciji StudentAMF.mxml, gde se definišu osobine width, height i id. Naziv instance ove klase je diagrams.

```
<gui:ResultDiagrams width="100%" height="100%" id="diagram"
includeIn="diagrams"/>
```

Klasa ResultDiagrams sadrži trideset i jednu javnu osobinu. Struktura klase je prikazana na slici3.7.75.



Slika 3.7.75. Grafički prikaz klase ResultDiagrams sa osobinama

U klasi ResultDiagrams kreiraju se tri grafikona, za AMF, JSON i XML. Na početku klase definiše se verzija xml dokumenta i sistem kodovanja.

```
<?xml version="1.0" encoding="utf-8"?>
```

Svaki grafikon vrši 50 učitavanja iz baze podataka, određeni broj slogova. Prosečno vreme učitavanja se smešta u promenljive averageJson, averageXml, averageAmf za slučaj JSON-a, Xml-a ili AMF-a. Zahtevani broj podataka, koji se učitavaju iz baze smešta se u promenljive requestedXml, requestedAmf i requestedJson.

```
        ..
[Bindable]
public var averageJson:String = "-";

[Bindable]
public var averageXml:String = "-";
[Bindable]
```

```
public var averageAmf:String = "-";

[Bindable]
public var requestedXml:String = "-";

[Bindable]
public var requestedAmf:String = "-";

[Bindable]
public var requestedJson:String = "-";
```

U ovoj klasi prave se kontrole addPage, test tipa Button, numOfObjects tipa ComboBox, kontejner captureArea tipa VGroup. Za crtanje grafikona koristi se instanca chart klase ColumnChart. ComboBox numOfObjects, popunjava se podacima iz statičkog niza arrayRequestNumber, koji je definisan u klasi StudentListVO.

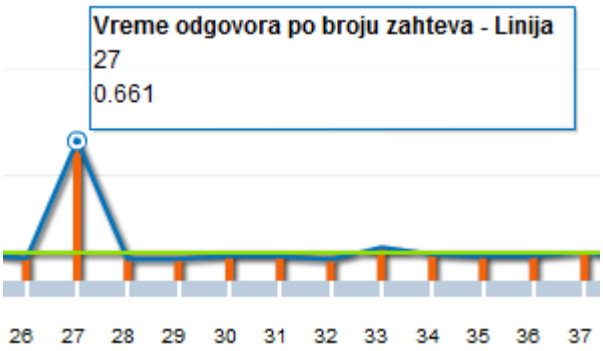
```
public static const arrayRequestNumber:Array =
[100,200,500,750,1000,1500,2000,2500,5000,7500,10000,15000,20000,30000,40000];
```

```
      <s:Button
            id="addPage"
            label="Add PDF page"
            bottom="10" left="20"
            />
      <s:Button
            id="test"
            label="Make test"
            bottom="10" right="20"
            />
      <s:ComboBox
            id="numOfObjects"
            dataProvider="{new
ArrayList(StudentListVO.arrayRequestNumber)}"
            bottom="10" right="130" selectedIndex="2"
            />
      <s:VGroup
            width="100%" height="100%"
            top="20" left="20" right="20" bottom="50"
            id="captureArea"
            >
```

U VGroup definiše se labelu lb1, koja primenjuje stil diagramHeader, definisan u klasi Style.css.

```
<s:Label text="AMF" styleName="diagramHeader" id="lb1" />
```

U klasi StudentListVo.as, definisan je statički niz amfDataProvider, tipa ArrayCollection, koji sadrži podatke, koji se koriste za punjenje DataProvider-a u grafikonima. Naime, linija koda dataProvider="{StudentListVO.amfDataProvider}" definiše, odakle se puni dataProvider grafikona. Osobina showDataTips="true" dopušta kada se pređe mišem preko grafika, da pokaže vrednost, što se vidi na slici3.7.76.



Slika 3.7.76. Grafikon prikazuje vreme učitavanja podataka

```
<mx:ColumnChart
	id="chart"
	height="100%" width="100%"
	dataProvider="{ StudentListVO.amfDataProvider}"
	showDataTips="true"
>
```

Između para tagova <mx:horizontalAxis>…</mx:horizontalAxis> definiše se horizontala osa na grafikonu. Maksimalan broj podeoka po horizontali je 60, maximum="60". Na svaki podeok učita se isti broj komada iz baze, i meri se vreme, a osa za vreme je y osa. id="horizontalAxis" definiše naziv horizontalne linije. Automatski podešavanje broja podeoka po širini nije dozvoljeno, autoAdjust="false". Takođe, nije podešeno da linije počnu iz koordinatnog početka, t.j. nule, baseAtZero="false". Osobina interval="1" određuje interval između podeoka na horizontalnoj osi, ovde je razlika između linija jedan podeok. minorInterval="1" definiše rastojanje između linija.

```
<mx:horizontalAxis>
	<mx:LinearAxis
		id="horizontalAxis"
```

```
                autoAdjust="false"
                baseAtZero="false"
                interval="1"
                minorInterval="1"
                maximum="60"
                />
        </mx:horizontalAxis>
```

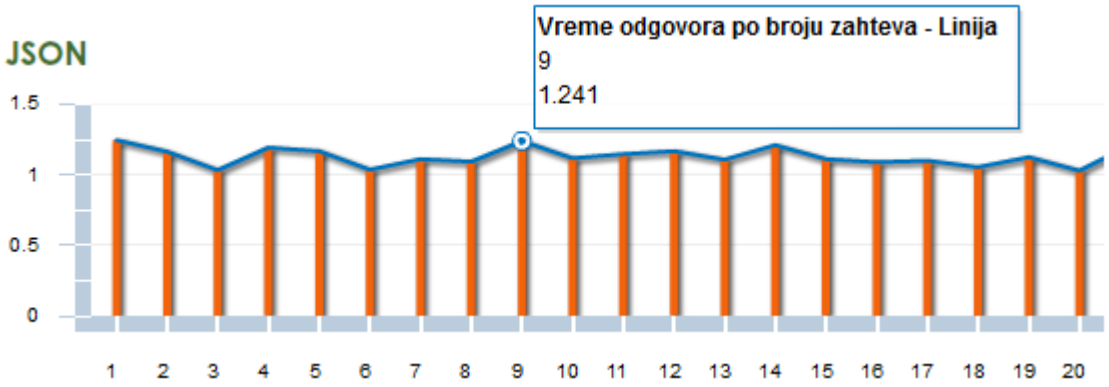
Elemenat <mx:VerticalAxis> definiše y osu. "verticalAxis" je naziv objekta koji definiše vertikalnu osu. Ovde se automatski podešava visina, autoAdjust="true". Interval između podeoka na vertikalnoj osi je 0.5, pošto se na **y** osi mere sekunde; interval="0.5".

```
        <mx:verticalAxis>
              <mx:LinearAxis id="verticalAxis" autoAdjust="true"
interval="0.5"/>
        </mx:verticalAxis>
```

Tag <mx:series> definiše serije. Serije su svi crteži u ovaj prikaz. MX komponenta ColumnSeries definiše vertikalnu kolonu, poseduje pet osobina. Linijom xField="callNumber", definiše se, koja promenljiva je na x osu, a to je "callNumber".
Konstrukcija yField="timeValue" definiše koja je promenljiva na y osu. Ovde se sa xField i timeValue definišu koordinate na x i y osi. xField dobija vrednost iz callNumber a yField iz timeValue. callNumber i timeValue su imena osobina objekta obj, koji predstavlja elemenat niza amfDataProvider. Osobine callNumber i timeValue napravljene su i dodejena im je vrednost u proksiju, unutar metode getAllStudentiFlex.

```
        obj.callNumber=steper;
        obj.timeValue=responseTime/1000;
```

Ovde je neophodno osigurati, da se promenljive isto zovu, za x(xField) i y(yField) osu, da bi program mogao ispravno da ih upari. Osobina visible="true" definiše da li su stubići kolone, na y osi vidljivi ili ne. Ako je vrednost za visible true, onda su vidljivi.
displayName="*Vreme odgovora po broju zahteva - Kolona*". Kada se postavi pokazivač miša na stubić u grafikonu, prikazuje se sadržaj osobine displayName. Stubići daju informacije koje je vreme potrebno od trenutka upućivanja zahteva serveru od strane klijenta, do trenutka dostavljanja podataka klijentu, od strane servera, za pojedine pokušaje, prvi, drugi pokušaj i tako do pedeset. Sada se može meriti vrednost, što dopušta opcija *showDataTips*="true".



Slika 3.7.77. Grafikon prikazuje stubce, koji predstavljaju vreme odgovora po broju zahteva

Nas slici3.7.77., se dobija informacija, u devetom obraćanju serveru (xField="callNumber", odnosno callNumber=9), podatak je dopremljen za 1.241 sekunde (yField="timeValue" t.j. timeValue=1.241 sekunde).

```
<mx:series>
      <mx:ColumnSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            maxColumnWidth="2"
            displayName="Vreme odgovora po broju zahteva - Kolona"
            />
```

MX komponenta LineSeries, id="lsamfKontinualnaLinija", definiše horizontalnu plavu liniju, poseduje sedam osobina. Linijom xField="callNumber", definiše se, koja promenljiva je na x osu, a to je "callNumber". Konstrukcija yField="timeValue" definiše koja je promenljiva na y osu. Brojne vrednosti za callNumber i yField izvlače se iz niza amfDataProvider. Elementi niza su objekti obj, koji sadrži osobine callNumber i yField.

Osobini displayName unutar mx komponente LineSeries, dodeljuje se vrednost displayName="*Vreme odgovora po broju zahteva - Linija*". Plava linija na grafikonu, povezuje stubiće, i daje informaciju o vremenu učitavanja, za pojedine pokušaje. Vreme se meri od trenutka upućivanja zahteva serveru od strane klijenta, do trenutka dostavljanja podataka klijentu, za svako učitavanje pojedinačno.

```
<mx:LineSeries
            xField="callNumber" yField="timeValue"
            visible="true"
```

```
            displayName="Vreme odgovora po broju zahteva - Linija"
            stroke="{new SolidColorStroke(0x0072bc,1,1)}"
            fill="{new SolidColor(0x0072bc)}"
            id="lsamfKontinualnaLinija"
            />
```

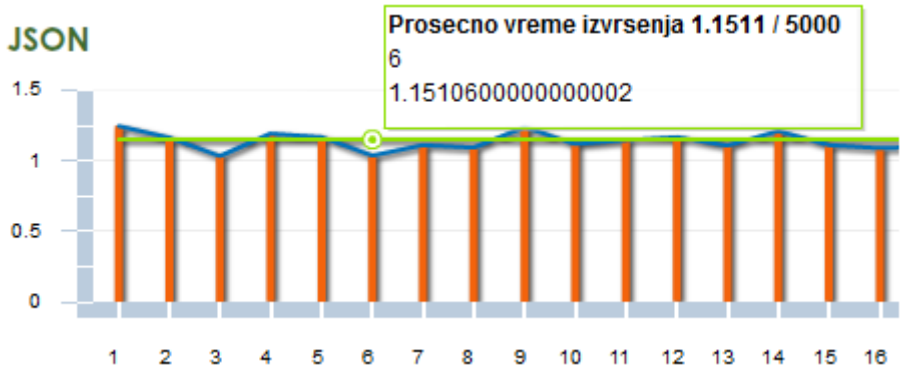
Osobina stroke definiše izgled linije, instance je klase SolidColorStroke. Kao parametri konstruktor funkcije SolidColorStroke navodi se boja 0x0072bc, debljina linije u pikselima, ovde je 1px, i vidljivost linije, alpha je 1.

Osobina fill popunjava kolone. Naime, fill se popunjava objetom klase SolidColor, boje 0x0072bc. Dakle, ovde se definiše okvir linije sa stroke, i ta se linija popunjava sa fill.

Objekat lsamfPravaLinija sadrži sedam osobina xField, yField, displayName, stroke, fill i id. Naziv objekta se nalazi u id. Osobine xField i yField predstavljaju koordinate tačaka na x i y osi, kroz koje se provlači zelena linija. Ova linija predstavlja prosečno vreme učitavanje, za sve pokušaje. U ovom slučaju sve koordinate xField i yField imaju istu vrednost. Osobina visible="true" ukazuje da je horizontalna linija vidljiva, koja definiše prosečnu vrednost. Prosečno vreme se čuva u promenljivu averageAmf. Zahtevani broj komada iz baze smešten je u promenljivu requestedAmf. Osobina styleName definiše stil fonta na grafikonu, a stil fonta je definisan u bloku diagramIspis, unutar fajla Style.css. Sa lineStroke definiše se puna linija, boje 0x0072bc, debljine 2px, i prozirnosti 1.

Dakle, prava linija na grafikonu, daje informaciju, koje je prosečno vreme potrebno od trenutka upućivanja zahteva serveru od strane klijenta, do trenutka dostavljanja podataka klijentu, od strane server za sva učitavanja, što je prikazano na slici3.7.78.

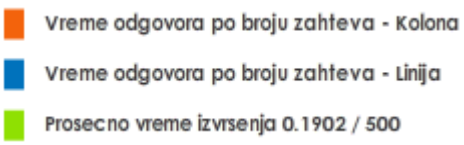
```
      <mx:LineSeries
            xField="callNumber" yField="average"
            visible="true"
            displayName="Prosecno vreme izvrsenja {averageAmf} /
{requestedAmf}"
            styleName="diagramIspis"
            lineStroke="{new SolidColorStroke(0x0072bc,2,1)}"
            id="lsamfPravaLinija"
            />
</mx:series>
```



Slika 3.7.78. Grafikon prikazuje stubce, horizontalnu liniju i pravu liniju, koja predstavlja prosečno vreme učitavanja za n učitavanja.

```
</mx:ColumnChart>
```

Linija koda <mx:Legend dataProvider="{chart}"/> definiše legendu, a to su troje dugmadi, t.j. kvadratića ispod grafikona, što prikazano na slici 3.7.79. dataProvider se snabdeva podacima iz objekta chart, a primenjeni stil diagramIspis, definisan je u fajlu Style.css.



Slika 3.7.79. Kvadratići se odnose na pojedine delove u grafikonu, stubiće, kontinuanu liniju i pravu liniju.

```
<mx:Legend dataProvider="{chart}" styleName="diagramIspis"/>
```

U narednom delu koda se definiše grafikon za JSON tehnologiju učitavanja i slanja podataka.

```
<s:Label text="JSON" styleName="diagramHeader"/>
```

Objekat chartJson klase ColumnChart definiše izgled grafikona. dataProvider se snabdeva podacima iz statičkog niza jsonDataProvider.

```
<mx:ColumnChart
      id="chartJson"
      height="100%" width="100%"
      dataProvider="{ StudentListVO.jsonDataProvider }"
      showDataTips="true"
      >
```

Ovde se definiše horzontalna osa, za json grafikon.

```
<mx:horizontalAxis>
      <mx:LinearAxis
            id="horizontalAxisJson"
```

```
                autoAdjust="false"
                baseAtZero="false"
                interval="1"
                minorInterval="1"
                maximum="60"
                />
    </mx:horizontalAxis>
```

U mx komponenti horizontalAxis, osobina maximum=60, znači da x osa, po horizontali ima 60 podeoka, što se vidi na slici3.7.80. Na svaki podeok učita se isti broj komada iz baze, i meri se vreme, a osa za vreme je y osa.

Slika 3.7.80. Grafikon prikazuje 51 podeok, gde se meri vreme učitavanja podataka u json format

U narednom bloku koda, mx:verticalAxis komponenta, definiše se vertikalna osa.

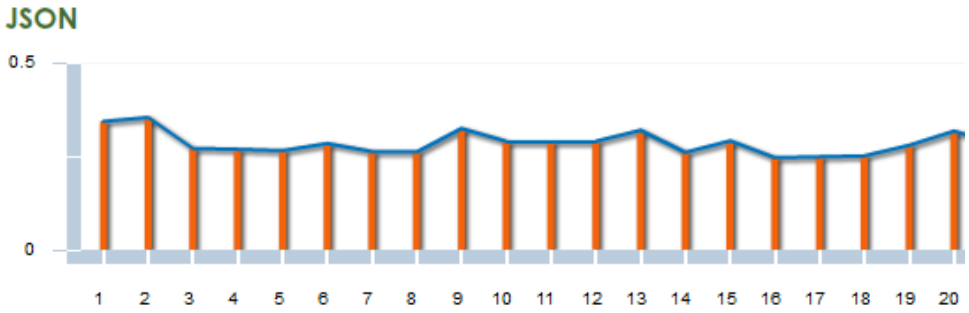
```
<mx:verticalAxis>
<mx:LinearAxis id="verticalAxisJson" autoAdjust="true" interval="0.5"/>
</mx:verticalAxis>
```

Komponenta ColumnSeries definiše stubiće u grafikonu.

```
<mx:series>
<mx:ColumnSeries xField="callNumber" yField="timeValue"visible="true"
maxColumnWidth="2"displayName="Vreme odgovora po broju zahteva -
Kolona"styleName="diagramIspis"fill="{new SolidColor(0xf3630c)}"
/>
```

MX:LineSeries komponenta, definiše izgled horizontalne linije[1], na slici 3.7.81 je prikazana plavom bojom.

```
<mx:LineSeries xField="callNumber" yField="timeValue" visible="true"
displayName="Vreme odgovora po broju zahteva - Linija"
styleName="diagramIspis"lineStroke="{new
SolidColorStroke(0x0072bc,2,1)}"id="lsjsonKontinualnaLinija"
/>
```



Slika 3.7.81. MX:LineSeries komponenta, definiše izgled horizontalne linije

U objektu lsjsonPravaLinija, klase LineSeries, definiše se horizontalna linija na grafikonu, koja predstavlja prosečno vreme učitavanja za svih pedeset učitavanja.

```
<mx:LineSeries xField="callNumber" yField="average" visible="true"
displayName="Prosecno vreme izvrsenja {averageJson} / {requestedJson}"
styleName="diagramIspis"lineStroke="{new
SolidColorStroke(0x8FE002,2,1)}"id="lsjsonPravaLinija"
/>
</mx:series>

      </mx:ColumnChart>
      <mx:Legend dataProvider="{chartJson}" styleName="diagramIspis"/>
```

Naredni blok koda definiše objekat chartXml, klase ColumnChart. U ovom objektu definiše se izgled grafikona za xml tehnologiju pakovanja podataka.

```
<s:Label text="XML" styleName="diagramHeader" />
      <mx:ColumnChart id="chartXml" height="100%" width="100%"
      dataProvider="{ StudentListVO.xmlDataProvider }"
                  showDataTips="true"
>
```

MX komponenta horizontalAxis definiše izgled horizontalne ose.

```
      <mx:horizontalAxis>
            <mx:LinearAxis id="horizontalAxisXml"autoAdjust="false"
                  baseAtZero="false"interval="1"minorInterval="1"
                  maximum="60"
                  />
      </mx:horizontalAxis>
```

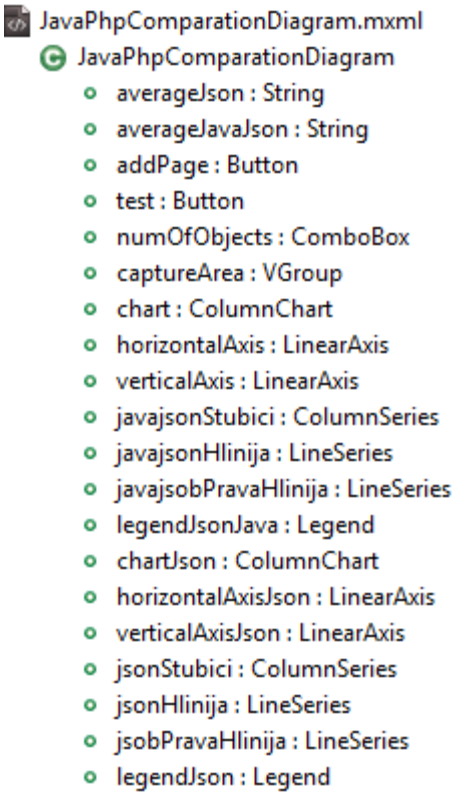
U paru tagova <mx:verticalAxis> i </mx:verticalAxis> definiše se vertikalna osa[1].

3.7.22 Klasa JavaPhpComparationDiagram.mxml

Grafički interfejs JavaPhpComparationDiagram obezbeđuje prikaz dva grafikona. Jedan grafikon se pravi pod Javom, u json formatu se pakuju podaci. Drugi grafikon radi sa json tehnologijom, sa php serverom. Grafička komponenta JavaPhpComparationDiagram se nalazi u paketu gui. Ova komponenta se kreira u glavnoj aplikaciji StudentAMP.mxml, u liniji koda.

```
<gui:JavaPhpComparationDiagram width="100%" height="100%" id="javaPhpComp"
includeIn="javaPhp"/>
```

Objekat javaPhpComp, klase JavaPhpComparationDiagram, dodaje se u stanju javaPhp, koje je definisano unutar glvne aplikacije StudentAMF.mxml. Prikaz ove komponente po širini i visini prostire se na 100% površine roditeljske klase. Klasa JavaPhpComparationDiagram.mxml sadrži dvadeset javnih osobina. Struktura klase je prikazana na slici 3.7.82.



Slika 3.7.82. Grafički prikaz klase JavaPhpComparationDiagram sa osobinama.

Promenljive averageJson i averageJavaJson sadrže prosečno vreme, za sva učitavanja, iz baze podataka. Ove promenljive su *Bindable*, što znači, svaka promena na njima, odraziće se na svim mestima gde se primenjuju[1].

```
[Bindable]
public var averageJson:String = "-";
[Bindable]
public var averageJavaJson:String = "-";
```

Promenljiva averageJson se puni u metod handleNotification, klase ApplicationMediator.ac. Prvo se proverava da li je proksi StudentProxyJson aktivan, a onda u bloku koda, if naredbom se proverava da li je javaPhpComp komponenta aktivna, ako jeste, pristupa se njenoj osobini view.javaPhpComp.averageJson i dodeljuje joj se vrednost average, iz objekta obj, koji je elemenat statičkog niza jsonDataProvider, koji snabdeva podacima dataProvider grafički objekat chart klase ColumnChart, linijom koda dataProvider="{ StudentListVO.jsonJavaDataProvider }".

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyJson.NAME){
...
if (view.javaPhpComp)
     view.javaPhpComp.averageJson =
Number(StudentListVO.jsonDataProvider.getItemAt(0).average).
     toFixed(4);
}
```

Statički niz jsonDataProvider, za slučaj, kada se učitava 40 000 komada iz baze, ima oblik:

jsonDataProvider[0]		jsonDataProvider[1]		jsonDataProvider[2]			jsonDataProvider[51]	
callNumber	1	callNumber	2	callNumber	3	...	callNumber	51
timeValue	10.245	timeValue	10.487	timeValue	10.692		timeValue	0.144
average	0.1603	average	0.1603	average	0.1603		average	0.1603
requested	40 000	requested	40 000	requested	40 000		requested	40 000

Slika 3.7.83. Statički niz jsonDataProvider

U narednim linijama koda vrši se dodela prosečne vrednosti vremena, za sva učitavanja, osobini view.javaPhpComp.averageJavaJson, kada je aktivna komponenta javaPhpComp i setovan proksi StudentProxyJavaJson.

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaJson.NAME)
view.javaPhpComp.averageJavaJson=Number(StudentListVO.jsonJavaDataProvider.
getItemAt(0).average).toFixed(4);
```

Objekat javaPhpComp sadrži dve osobine addPage i test tipa Button, jednu osobinu numOfObjects tpa ComboBox. Instanca numOfObjects se snabdeva podacima iz statičkog niza

arrayRequestNumber, klase StudentListVO. U ComboBoxu selektuje se element iz niza sa indeksom 1, selectedIndex="1". Indeksi u nizu počinju od 0.

```
            <s:Button
                  id="addPage"
                  label="Add PDF page"
                  bottom="10" left="20"
                  />
            <s:Button
                  id="test"
                  label="Make test"
                  bottom="10" right="20"
                  />
            <s:ComboBox
                  id="numOfObjects"
                  dataProvider="{new
ArrayList(StudentListVO.arrayRequestNumber)}"
                  bottom="10" right="130" selectedIndex="1"
                  />
```

U kontejneru VGroup se nalazi komponenta ColumnChart, koja crta grafikone. Objekat VGroupe je captureArea.

```
      <s:VGroup
            width="100%" height="100%"
            top="20" left="20" right="20" bottom="50"
            id="captureArea"
            >
```

Unutar VGroup kontejnera, nalazi se label kontrola, sa text="Java-Json", primenjeni stil je diagramHeader, koji je definisan u style.css fajlu.

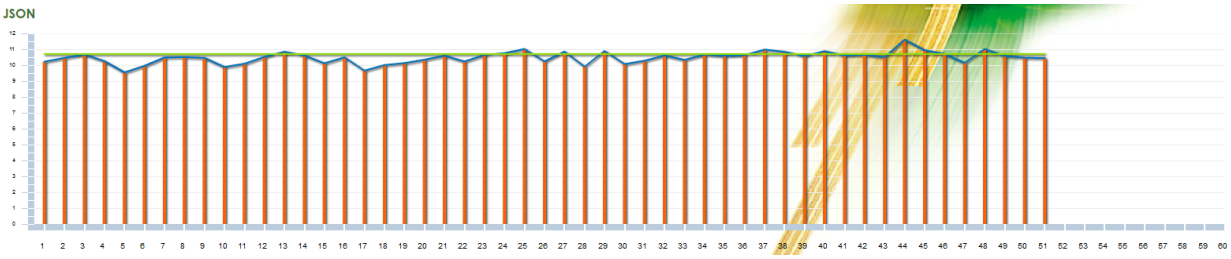
```
            <s:Label text="Java-Json" styleName="diagramHeader" />
```

Klasa koja definiše grafikon je mx:ColumnChart, a objekat ovog grafikona zove se id="chart". Grafička klasa ColumnChart se snabdeva podacima iz statičkog niza jsonJavaDataProvider

```
      <mx:ColumnChart
                  id="chart"
                  height="100%" width="100%"
                  dataProvider="{ StudentListVO.jsonJavaDataProvider }"
                  showDataTips="true"
```

```
>
```

Komponenta horizontalAxis definiše izgled x ose, na grafikonu. Osobina maximum=60, govori da po horizontali ima 60 podeoka, što se vidi na slici. Na svaki podeok se učita 40000 istih komada iz baze, i meri se vreme, a osa za vreme je y osa.



Slika 3.7.84. Komponenta horizontalAxis definiše izgled x ose, na grafikonu

```
<mx:horizontalAxis>
      <mx:LinearAxis
             id="horizontalAxis"
            autoAdjust="false"
            baseAtZero="false"
            interval="1"
            minorInterval="1"
            maximum="60"
            />
</mx:horizontalAxis>
```

mx:verticalAxis komponenta oblikuje izgled vertikalne ose.

```
      <mx:verticalAxis>
<mx:LinearAxis id="verticalAxis" autoAdjust="true" interval="0.5"/>
      </mx:verticalAxis>
```

Klasa ColumnSeries definiše stubiće u grafikonu.

```
<mx:series>
      <mx:ColumnSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            maxColumnWidth="2"
            displayName="Vreme odgovora po broju zahteva - Kolona"
            id="javajsonStubici"
            />
```

Klasa LineSeries definiše horizontalnu liniju, koja povezuje stubiće u grafikonu. Linija se provlači kroz tačke, koje su definisane koordinatama xField i yField. Koordinate za tačke se mogu dati i tabelarno, na primer za slučaj da se učitava 4000 istih komada, na svaki podeok.

```
<mx:LineSeries
      xField="callNumber" yField="timeValue"
      visible="true"
      displayName="Vreme odgovora po broju zahteva - Linija"
      stroke="{new SolidColorStroke(0x0072bc,1,1)}"
      fill="{new SolidColor(0x0072bc)}"
      id="javajsonHlinija"
      />
```

Naredna LineSeries komponenta, definiše pravu liniju, koja predstavlja prosečno vreme za sva učitavanja, pri čemu se na svaki podeok učitava isti broj komada iz baze. Prosečno vreme izvršenja se nalazi u promenljivoj averageJavaJson.

```
<mx:LineSeries
      xField="callNumber" yField="average"
      visible="true"
      displayName="Prosecno vreme izvrsenja {averageJavaJson}"
      styleName="diagramIspis"
      lineStroke="{new SolidColorStroke(0x0072bc,2,1)}"
      id="javajsobPravaHlinija"
      />
</mx:series>
</mx:ColumnChart>
<mx:Legend dataProvider="{chart}" styleName="diagramIspis"/>

<s:Label text="JSON" styleName="diagramHeader"/>
```

Objekat chartJson definiše grafikon za json tehnologiju, i php serverom.

```
<mx:ColumnChart
      id="chartJson"
      height="100%" width="100%"
      dataProvider="{ StudentListVO.jsonDataProvider }"
      showDataTips="true"
      >
```

Komponenta horizontalAxis definiše x osu.

```
<mx:horizontalAxis>
```

```
            <mx:LinearAxis
                  id="horizontalAxisJson"
                  autoAdjust="false"
                  baseAtZero="false"
                  interval="1"
                  minorInterval="1"
                  maximum="60"
                  />
      </mx:horizontalAxis>
```

Klasa verticalAxis oblikuje y osu.

```
      <mx:verticalAxis>
<mx:LinearAxis id="verticalAxisJson" autoAdjust="true" interval="0.5"/>
</mx:verticalAxis>
<mx:series>
```

Klasa ColumnSeries definiše vertikalne stubce, na grafikonu.

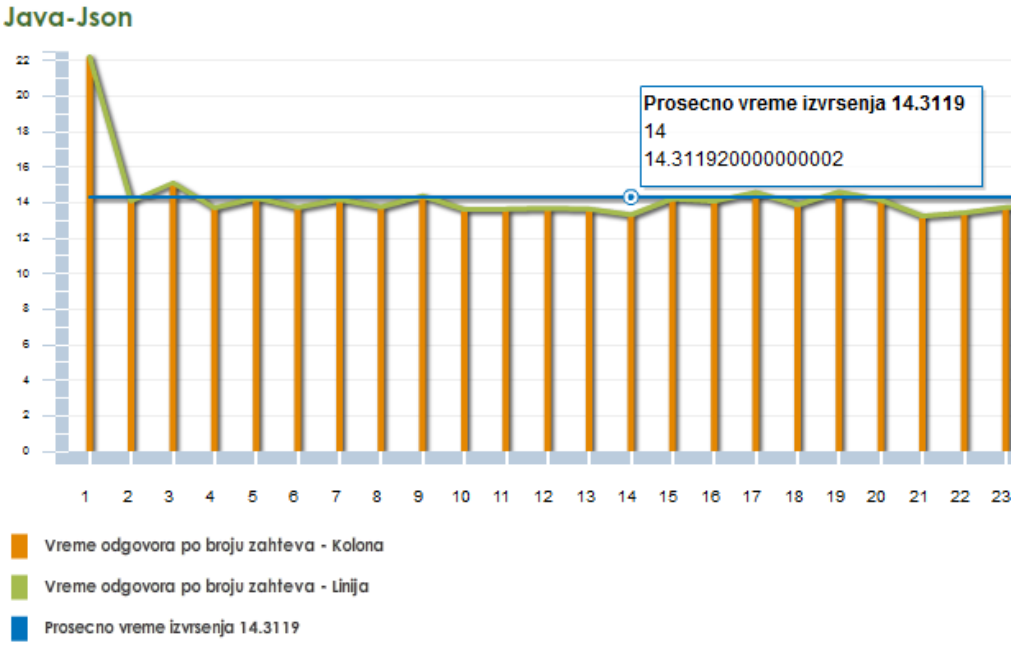
```
<mx:ColumnSeries xField="callNumber" yField="timeValue"visible="true"
maxColumnWidth="2"displayName="Vreme odgovora po broju zahteva - Kolona"
styleName="diagramIspis"fill="{new SolidColor(0xf3630c)}"id="jsonStubici"
/>
```

Komponenta LineSeries definiše liniju, koja se prostire između stubaca na grafikonu.

```
<mx:LineSeries xField="callNumber" yField="timeValue" visible="true"
displayName="Vreme odgovora po broju zahteva-
Linija"styleName="diagramIspis"lineStroke="{new
SolidColorStroke(0x0072bc,2,1)}"
id="jsonHlinija"/>
```

Objekat jsobPravaHlinija klase LineSeries definiše pravu liniju, koja odslikava prosečno vreme učitavanja za sve podeoke, na x osi, slika3.6.85.

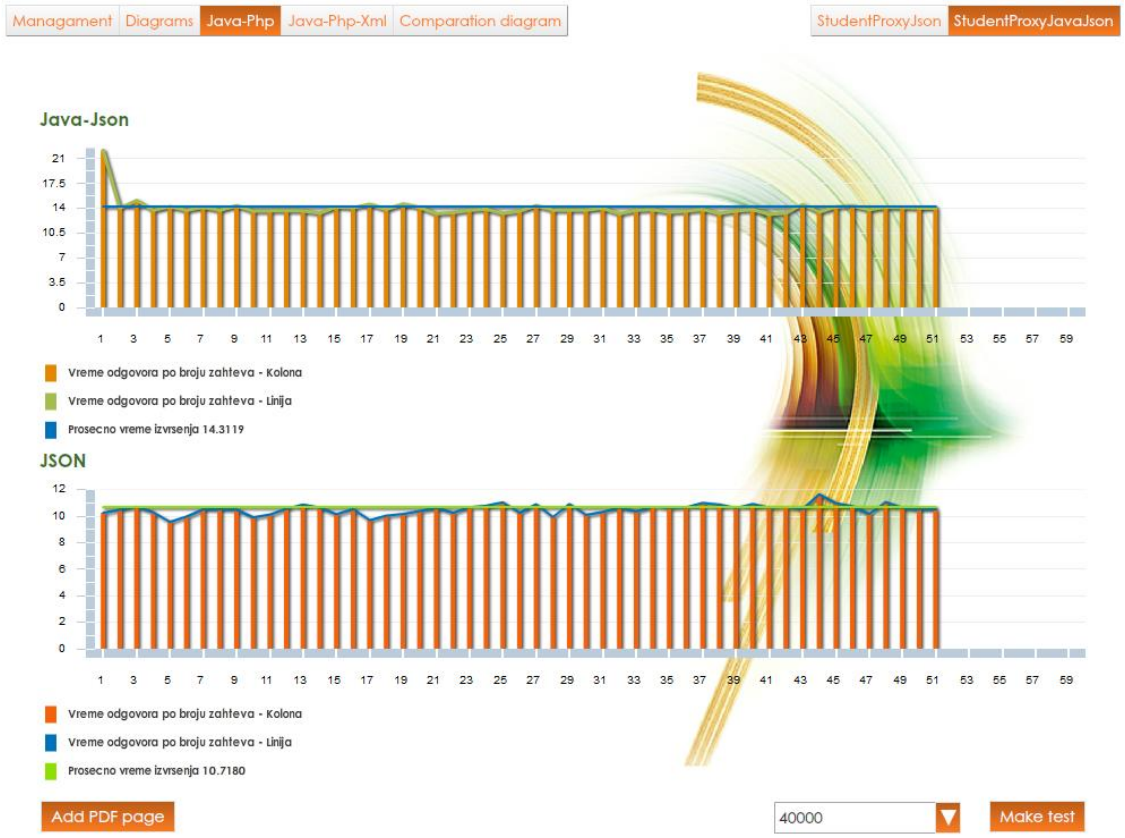
```
<mx:LineSeries xField="callNumber" yField="average" visible="true"
displayName="Prosecno vreme izvrsenja {averageJson}"
styleName="diagramIspis"lineStroke="{new SolidColorStroke(0x8FE002,2,1)}"
id="jsobPravaHlinija"/></mx:series>
```



Slika 3.7.85. Grafikon za format json u Javi.

```
</mx:ColumnChart>
<mx:Legend dataProvider="{chartJson}" styleName="diagramIspis"/>
</s:VGroup>
</s:Group>
```

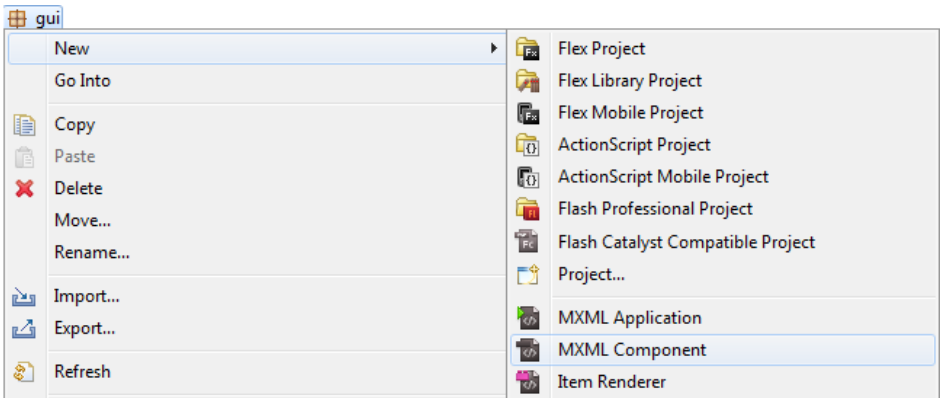
Instanca javaPhp klase JavaPhpComparationDiagram, prdstavljena je na slici3.7.86.



Slika 3.7.86. Komparacijski dijagrami za json format u php-u i Javi

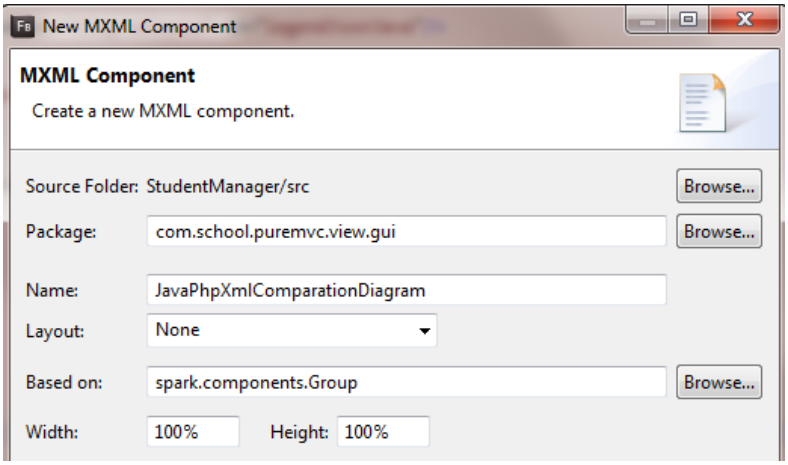
3.7.23 Klasa JavaPhpXmlComparationDiagram.mxml

Grafička komponenta JavaPhpXmlComparationDiagram se nalazi u paketu gui. Kreira se desnim klikom na paket gui, a onda se odabere stavka MXML Component iz menija, što je prikazano na slici3.7.87.



Slika 3.7.87. meni za kreiranje JavaPhpXmlComparationDiagram komponente

Izborom stavke MXML Component, otvara se dijalog prozor.



Slika 3.7.88. Dijalog prozor za kreiranje JavaPhpXmlComparationDiagram klase

U polje Name, daje se ime klasi. Klasa JavaPhpXmlComparationDiagram dodaje se u glavnoj aplikaciji StudentAMF, kada je stanje aplikacije javaPhpXml, linijom koda:

```
<gui:JavaPhpXmlComparationDiagram width="100%" height="100%"
id="javaXmlCompar" includeIn="javaPhpXml"/>
```

Promenljive averageXml i averageJavaXml čuvaju srednju vrednost vremena za sva učitavanja.

```
 [Bindable]
public var averageXml:String = "-";
```

```
[Bindable]
public var averageJavaXml:String = "-";
```

Promenljivoj averageJavaXml dodeljuje se vrednost u metod handleNotification, klase ApplicationMediator, kada se izvrše sva učitavanja iz baze podataka, na svim podeocima.

```
if (ApplicationFacade.SELECTED_PROXY == StudentProxyJavaXml.NAME)
   view.javaXmlCompar.averageJavaXml =
Number(StudentListVO.xmlJavaDataProvider.getItemAt(0).average).toFixed(4);
```

Na isti način se vrši dodela vrednosti i promenljivoj averageXml.

```
view.javaXmlCompar.averageXml =
Number(StudentListVO.xmlDataProvider.getItemAt(0).average). toFixed(4);
```

Grafički interfejs JavaPhpXmlComparationDiagram, sadrži dugme addPage za dodavanje stranice u pdf format, test tipa Button, za startovanje grafikona, objekat numOfObjects tipa ComboBox, koji sadrži listu brojeva, čijim izborom se određuje broj učitanih podataka na svakom podeoku grafikona. U ComboBox-u, dataProvider se snabdeva podacima iz statičkog niza arrayRequestNumber.

```
<s:Button
      id="addPage"
      label="Add PDF page"
      bottom="10" left="20"
      />
<s:Button
      id="test"
      label="Make test"
      bottom="10" right="20"
      />
<s:ComboBox
      id="numOfObjects"
      dataProvider="{new rrayList(StudentListVO.arrayRequestNumber)}"
      bottom="10" right="130" selectedIndex="1"
      />
```

Da bi izvršili određeni zadatak, kada se klikne na osobine addPage, test i numOfObjects, klase JavaPhpXmlComparationDiagram, dodaje se listener u medijatoru ApplicationMediator.ac za svaku osobinu, unutar metode onStateChangeHandler. Medijatori upravljaju grafičkim komponentama.

```
private function onStateChangeHandler(event:FlexEvent):void
{
            …
else if (view.currentState == "javaPhpXml"){
      view.javaXmlCompar.test.removeEventListener(MouseEvent.CLICK,
onTestHandler);
      view.javaXmlCompar.test.addEventListener(MouseEvent.CLICK,
onTestHandler,false,0,true);

view.javaXmlCompar.numOfObjects.removeEventListener(IndexChangeEvent.CHANGE,
       numOfObjects_changeHandler);

view.javaXmlCompar.numOfObjects.addEventListener(IndexChangeEvent.CHANGE,
      numOfObjects_changeHandler,false,0,true);

view.javaXmlCompar.addPage.removeEventListener(MouseEvent.CLICK,
onAddPageHandler);
view.javaXmlCompar.addPage.addEventListener(MouseEvent.CLICK,
onAddPageHandler,false,0,true);

      ApplicationFacade.SELECTED_PROXY = StudentProxyXml.NAME;
      }
}
```

Metod onTestHandler alocira memoriju za statički niz xmlJavaDataProvider, setuje proxy.steper na 1, proxy.average na 0, i poziva metod proksija

```
proxy.getAllStudentiFlex(controlMediator.currentPage),
```

koja uspostavlja vezu sa serverom. Sa server, podaci se obrađuju u metod proksija on_getAllStudentiFlexHandler, gde se i popunjava niz xmlJavaDataProvider vrednostima, na osnovu kojih se crta grafikon. Metod numOfObjects_changeHandler kopira vrednost iz ComboBoxa u promenljivu requestNumber, gde se čuva zahtevani broj komada iz baze podataka.

```
StudentListVO.requestNumber = event.currentTarget.selectedItem;
      <s:VGroup
            width="100%" height="100%"
            top="20" left="20" right="20" bottom="50"
            id="captureArea"
```

```
            >
            <s:Label text="Java-Xml" styleName="diagramHeader" />
      <mx:ColumnChart
            id="chart"
            height="100%" width="100%"
            dataProvider="{ StudentListVO.xmlJavaDataProvider }"
            showDataTips="true"
            >
```

Ovde se definiše horizontalna osa.

```
<mx:horizontalAxis>
      <mx:LinearAxis
            id="horizontalAxis"
            autoAdjust="false"
            baseAtZero="false"
            interval="1"
            minorInterval="1"
            maximum="60"
            />
      </mx:horizontalAxis>
```

Ovde se definiše vertikalna osa.

```
<mx:verticalAxis>
      <mx:LinearAxis id="verticalAxis" autoAdjust="true" interval="0.5"/>
</mx:verticalAxis>
```

Series su grafikoni. ColumnSeries je jedan grafikon, prvi LineSeries je drugi grafikon, a drugi LineSeries je treći grafikon. ColumnSeries definiše stubac, a LineSeries definiše liniju.

```
<mx:series>
      <mx:ColumnSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            maxColumnWidth="2"
            displayName="Vreme odgovora po broju zahteva - Kolona"
            id="javaXmlStubci"
            />
      <mx:LineSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            displayName="Vreme odgovora po broju zahteva - Linija"
```

```
            stroke="{new SolidColorStroke(0x0072bc,1,1)}"
            fill="{new SolidColor(0x0072bc)}"
            id="javaXmlLinija"
            />
      <mx:LineSeries
            xField="callNumber" yField="average"
            visible="true"
            displayName="Prosecno vreme izvrsenja {averageJavaXml}"
            styleName="diagramIspis"
            lineStroke="{new SolidColorStroke(0x0072bc,2,1)}"
            id="javaXmlPravaLinija"
            />
      </mx:series>

</mx:ColumnChart>
      <mx:Legend dataProvider="{chart}" styleName="diagramIspis"
id="legendXmlJava"/>

      <s:Label text="XML" styleName="diagramHeader"/>
<mx:ColumnChart
      id="chartJson"
      height="100%" width="100%"
      dataProvider="{ StudentListVO.xmlDataProvider}"
      showDataTips="true"
      >
```

U tagu <mx:horizontalAxis> definiše se horizontalna osa.

```
<mx:horizontalAxis>
      <mx:LinearAxis
            id="horizontalAxisJson"
            autoAdjust="false"
            baseAtZero="false"
            interval="1"
            minorInterval="1"
            maximum="60"
            />
</mx:horizontalAxis>
```

U tagu <mx:verticalAxis> definiše se vertikalna osa.

```
      <mx:verticalAxis>
```

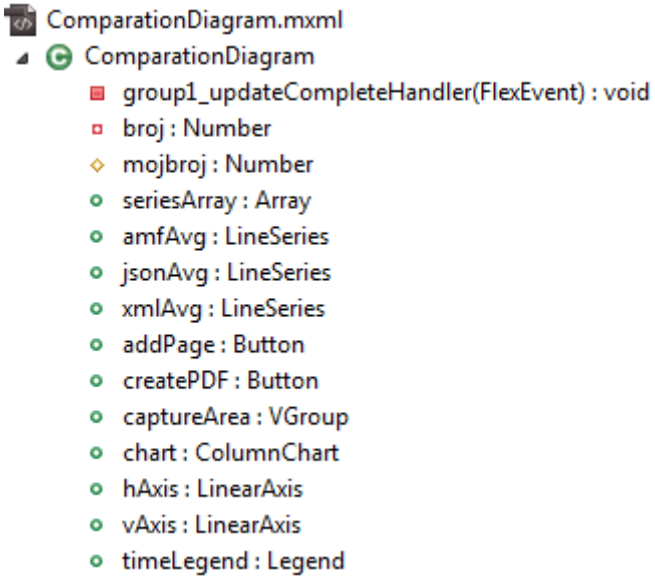
```
            <mx:LinearAxis id="verticalAxisJson" autoAdjust="true"
interval="0.5"/>
      </mx:verticalAxis>
<mx:series>
      <mx:ColumnSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            maxColumnWidth="2"
            displayName="Vreme odgovora po broju zahteva - Kolona"
            styleName="diagramIspis"
            fill="{new SolidColor(0xf3630c)}"
            id="xmlStubci"
            />
```

U tagu LineSeries definiše se linija.

```
      <mx:LineSeries
            xField="callNumber" yField="timeValue"
            visible="true"
            displayName="Vreme odgovora po broju zahteva - Linija"
            styleName="diagramIspis"
            lineStroke="{new SolidColorStroke(0x0072bc,2,1)}"
            id="xmlLinija"
            />
      <mx:LineSeries
            xField="callNumber" yField="average"
            visible="true"
            displayName="Prosecno vreme izvrsenja {averageXml}"
            styleName="diagramIspis"
            lineStroke="{new SolidColorStroke(0x8FE002,2,1)}"
            id="xmlPravaLinija"
            />
</mx:series>

</mx:ColumnChart>
      <mx:Legend dataProvider="{chartJson}" styleName="diagramIspis"
id="legendXml"/>
</s:VGroup>
</s:Group>
```

3.7.24 Klasa ComparationDiagram.mxml

Grafička komponenta ComparationDiagram se nalazi u paketu gui. Ova komponenta sadrži trinaest osobina i jednu metodu. Struktura klase je prikazana na slici3.7.89.



Slika 3.7.89. Prikaz osobina i metoda za klasu StudentProxyJson

Metod group1_updateCompleteHandler izvršava se u momentu kada se apdejtuje displei lista komplet komponente. Ovaj ivent se dešava kada se pokrene ova klasa. Naime, dispečuje se kada objekat klase StudentAMF završi sa pozivom metoda commitProperties(), measure(), i updateDisplayList().

Chart je ceo dijagram, legendData je niz svih podataka za legend iz celokupnog dijagrama. U nizu seriesArray definisane su tri kolone i tri linije. Dakle, legendData je osobina klase ColumnChart tipa Array. On se pravi tako, što se određuje broj child-ova, i toliko ima elemenata i ubacuje podatke potrebne za legendData.

```
private function group1_updateCompleteHandler(event:FlexEvent):void
{
      try {

timeLegend.dataProvider=[chart.legendData[3],chart.legendData[4],chart.lege
ndData[5]];
}
catch(err:Error){
trace("timeLegend error "+ err.message)
```

```
	}
}
		private var broj:Number;
		protectedvar mojbroj:Number;

		<fx:Array id="seriesArray">
```

Elemenat ColumnSeries definiše stubce u dijagramu. Snabdevač podataka je statički niz amfAvarage. Element statičkog niza predstavlja niz sa dva polja dva data i average. U data se nalazi zahtevani broj podataka koji se učitava iz baze a average sadrži prosečno vreme učitavanja za sve podeoke, na kojim se učitava isti broj podataka iz baze.

```
	<mx:ColumnSeries xField="data" yField="average"
		visible="true"maxColumnWidth="1"
		dataProvider="{ StudentListVO.amfAvarage}"
		/>
```

Nizovi StudentListVO.amfAvarage, StudentListVO.jsonAvarage i StudentListVO.xmlAvarage definisani su u klasi StudentListVO. Memorija za ove nizove alocira se u proksijima.

```
<mx:ColumnSeries
	xField="data" yField="average" visible="true"
	maxColumnWidth="1"dataProvider="{ StudentListVO.jsonAvarage }"
	/>
```

Statički niz StudentListVO.xmlAvarage popunjava se u proksiju StudentProxyAmf.ac, u linijama koda:

```
if (StudentListVO.jsonAvarage == null)
		StudentListVO.jsonAvarage = new ArrayCollection();
StudentListVO.jsonAvarage.addItem({data:StudentListVO.requestNumber,average
:average/ 50});

<mx:ColumnSeries xField="data" yField="average" visible="true"
	maxColumnWidth="1"dataProvider="{ StudentListVO.xmlAvarage }"
	/>
```

U ovoj LineSeries definiše se linija za amf.

```
<mx:LineSeries xField="data" yField="average" visible="true"
	displayName="Prosecno vreme izvrsenja AMF tipa podataka"
	lineStroke="{new SolidColorStroke(0x0072bc,1,1)}"
	dataProvider="{ StudentListVO.amfAvarage }"
	id="amfAvg"
```

```
      />
```
U ovom tagu kreira se linija za Json.

```
<mx:LineSeries
      xField="data" yField="average" visible="true"
      displayName="Prosecno vreme izvrsenja JSON tipa podataka"
      lineStroke="{new SolidColorStroke(0xf3630c,1,1)}"
      dataProvider="{ StudentListVO.jsonAvarage }"
      id="jsonAvg"
      />
<mx:LineSeries
      xField="data" yField="average"
      visible="true"
      displayName="Prosecno vreme izvrsenja XML tipa podataka"
      lineStroke="{new SolidColorStroke(0x8FE002,2,1)}"
      dataProvider="{ StudentListVO.xmlAvarage }"
      id="xmlAvg"
      />
</fx:Array>
</fx:Declarations>
```

Ova komponenta sadrži dva Button dugmeta addPage i createPDF za dodavanje stranice u pdf format i za kreiranje pdf formata.

```
<s:Button
      id="addPage"
      label="Add PDF page"
      bottom="10" left="20"
      />
<s:Button
      id="createPDF"
      label="Make PDF"
      bottom="10" right="20"
      />
```

Komponenta sadrži kontejner captureArea tipa VGroup.

```
<s:VGroup
      width="100%" height="100%"
      top="20" left="20" right="20" bottom="50"
      id="captureArea"
      >
```

```
      <s:Label text="Komparacija AMF-JSON-XML izvrsenja upita"
styleName="diagramHeader"/>
```

ColumnChart je klasa, njen objekat je chart-id="chart". Osobine klase ColumnChart su <mx:horizontalAxis>, <mx:verticalAxis>.

```
      <mx:ColumnChart
            id="chart"
            height="100%" width="100%"
            showDataTips="true"
            series="{seriesArray}"
            >
```

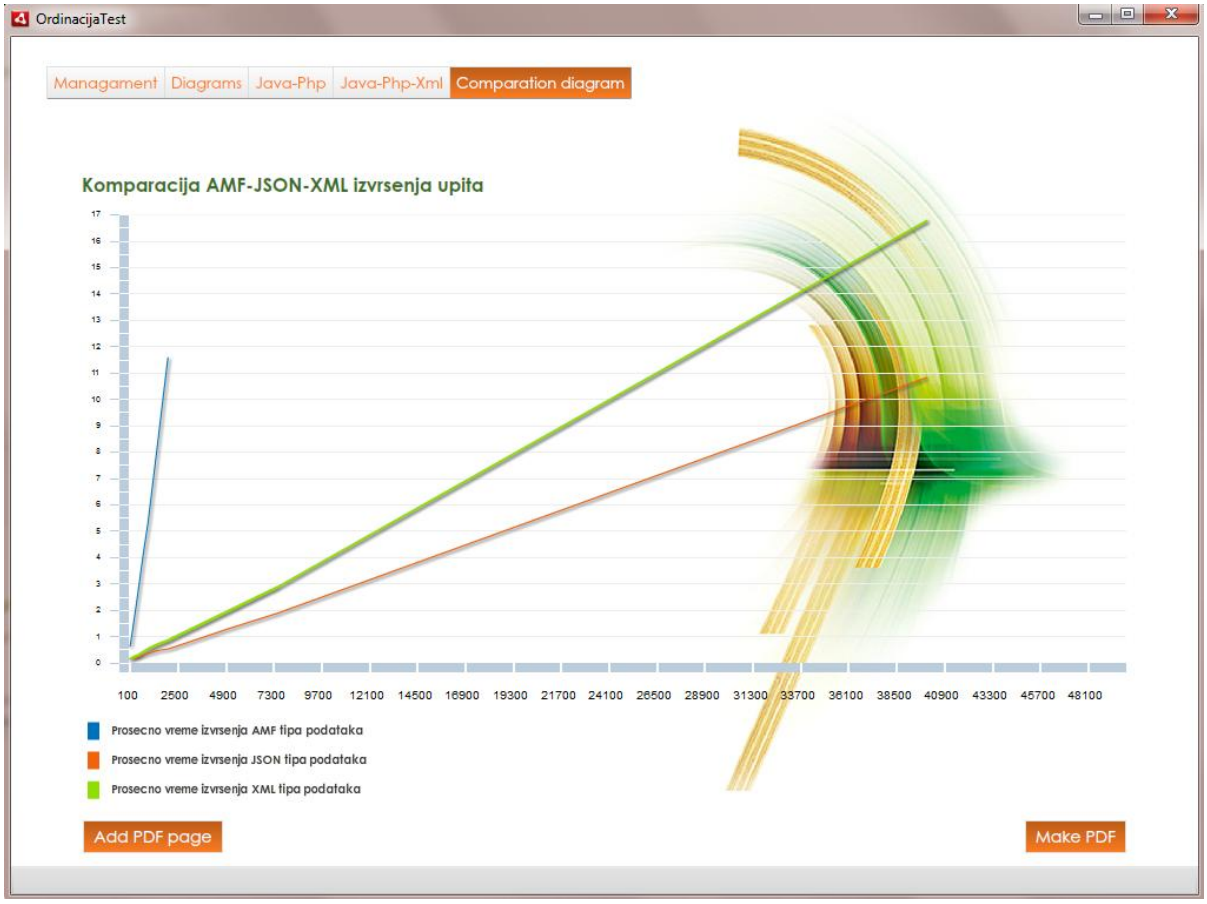
U horizontalAxis-u definišu se horizontalne linije.

```
<mx:horizontalAxis>
<mx:LinearAxis
      id="hAxis"
      autoAdjust="false"
      baseAtZero="false"
      interval="100"
      minorInterval="100"
      maximum="50000"
      />
</mx:horizontalAxis>
```

U tagu <mx:verticalAxis> definišu se vertikalne linije.

```
      <mx:verticalAxis>
            <mx:LinearAxis id="vAxis" autoAdjust="true" interval="0.5"/>
</mx:verticalAxis>

      </mx:ColumnChart>
            <mx:Legend id="timeLegend" styleName="diagramIspis" />
      </s:VGroup>
</s:Group>
```

Test je izvršen za nekoliko slučajeva učitavanja podataka iz baze. Komparacioni dijagram je prikazan na slici 3.7.90.



Slika 3.7.90. Komparacijski dijagram za amf, json i xml

Na osnovu grafika, može se zaključiti da vreme učitavanja podataka kod AMF-a raste veoma brzo sa povećanjem broja učitanih podataka. Kod Json-a i Xml-a ne postoji značajna razlika u brzini učitavanja podataka, pri malom broju učitanih komada iz baze, dok ta razlika postaje značajnija kako raste broj učitanih podataka. Xml fajl zauzima više memorije na hard disku, poseduje teže klase, koje sporije obrađuju pristigle podatke u dnosu na Json klase koje vrše serijalizaciju i deserijačlizaciju podataka. Kod je znatno komplikovaniji kod Xml-a nego kod Json-a.

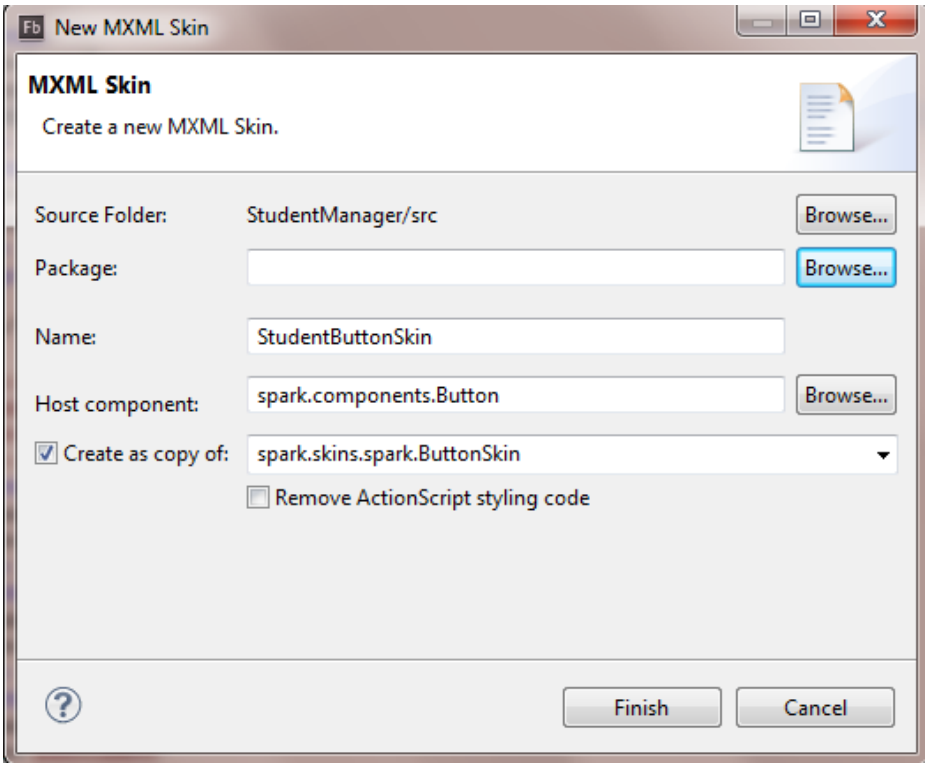
3.7.25 Skinovanje kontrola u Flex-u

Svako dugme generalno može da se skinuje posebno. Dugme ima kod sebe, nešto što se zove SkinClass. SkinClass je promenljiva, tipa klase. Ova klasa postoji unutar kontrole, što se vidi na slici 3.7.91.

```
<s:Button
    id="prev"
    label="Previous"
    skin
    />
```
skinClass : Class - SkinnableComponent

Slika 3.7.91. prikazuje komponentu, gde se definiše skin komponenta.

Izborom klase skinClass, dobija se dijalog prozor:



Slika 3.7.92. Forma za definisanje mxml skin komponente

Klasa za dugme se zove StudentButtonSkin. Paket se bira pritiskom na dugme Browse. *Host component* je roditeljska klasa skina. Za Button kontrolu, biće Button. U polju *Create as copy of* se pravi Button kopija ButtonSkina, pošto ButtonSkin klasa već postoji. Naime, postoji skin za svaku kontrolu. Pritiskom na dugme Finish, Flex sam kopira osnovnu klasu u novu funkciju, koja se sada menja. Skin klase se nalaze u paketu skin.

Ovde je napravljena skin klasa za Previus dugme. Sada je dugme *Previus* potpuno isto kao i *Next*, ili bilo koje dugme, zato što je u potpunosti nasleđena klasa skin, koja se zove ButtonSkin, i ništa novo nije dodato. Dakle, nasleđene su sve osobine klase ButtonSkin. Klasa je nasledila kompletan sadržaj klase SparkButtonSkin. Kod klase StudentButtonSkin, dat je narednim blokom koda.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:SparkButtonSkin xmlns:fx="http://ns.adobe.com/mxml/2009"
              xmlns:s="library://ns.adobe.com/flex/spark"
              xmlns:fb="http://ns.adobe.com/flashbuilder/2009"
              minWidth="50" minHeight="30"
              alpha.disabled="0.5">
```

Osobina minWidth="50", definiše minimalnu širinu dugmeta, ovde je 50 piksela. Visinu dugmeta određuje osobina minHeight="30", to je 30 piksela. Dakle, ovde je definisana minimalna širina i visina, što znači, da ne sme biti manja od inicijalnih vrednosti, mogu da budu veće. Ako se klikne na dugme, onda će proračunati label i proširiće koliko mu je potrebno. Ako je label dugmeta "A", onda će ostati 50x30, pošto "A" može da se smesti na tim dimenzijama. Ako stavimo 50x50, onda će biti stalno 50, i ako napišemo samo slovo "A".

...

<!-- states -->

Klasa poseduje četiri stanja. Svako stanje sadrži osobinu name, gde se definiše ime stanja. Stanja su up, over, down i disabled. Stanje up definiše slučaj kada se pritisne pa pusti dugme. Stanje over je prisutno kada se prelazi preko dugmeta, down je kada se pritisne dugmeta a disabled je kada je disejblovano dugme.

```
<s:states>
<s:State name="up" />
<s:State name="over" />
<s:State name="down" />
<s:State name="disabled" />
</s:states>
```

Naredniom blokom koda, definiše se pravougaonik, koji počinje po jedan piksel levo, desno, gore i dole u odnosu na roditeljski objekat. Ime objekta je shadow.

<s:Rect id="shadow" left="-1" right="-1" top="-1" bottom="-1" radiusX="2">, dakle, objekat shadow, prekriva osnovni objekat. Kada je vrednost za left, right, top i bottom -1, to znači da je ovaj objekat veći od roditelja, 101% su dimenzije u odnosu na osnovni objekat. Dakle, Rectangl čije ime je id="shadow", je za 1px veći od roditelja. Roditelj je ova klasa, StudentButtonSkin. Znači, prvi njegov child je veći za jedan px. Naime, shadow je veći za jedan px od roditelja. shadow je naziv objekta, Taj pravougaonik ima radiusX 2 piksela. Znači, da je zaobljenje po x-si 2px. Ako je left="0" right="0" top="0" bottom="0", onda će objekat

tipa Rectangle, da prekrije prethodni pravougaonik 100%. Dakle, parametar radiusX="2" govori da je pravougaonik zaobljen.

```
<s:Rect top="1" bottom="1" left="1" right="1" >
<s:stroke>
      <s:SolidColorStroke caps="none" color="0xDFDFDF" joints="miter"
      miterLimit="4" weight="1"/>
</s:stroke>
```

Tag <s:fill> govori da se Pravougaonik popunjava bojom. U klasi LinearGradient definiše se način popunjavanja pravougaonika bojom. <s:LinearGradient rotation="-90"> nam govori da se za popunjavanje koristi gradijentni pristup, gde se boja preliva sa jedne strane u drugu stranu po vertikali. LinearGradient ima dva unosa boje. Prva boja je #fd7d22, a druga #bc5d19. Prozirnost za prvu i drugu boju "1". Kada je prozirnost 1, to je neprovidno, osnovna boja, a 0 je providan skroz.

```
<s:fill>
      <s:LinearGradient rotation="-90">
            <s:GradientEntry alpha="1" color="#fd7d22" ratio="0"/>
            <s:GradientEntry alpha="1" color="#bc5d19" ratio="1"/>
      </s:LinearGradient>
      </s:fill>

</s:Rect>
<s:Rect id="border" left="0" right="0" top="0" bottom="0" radiusX="0"
    alpha="0">
<s:stroke>
<s:SolidColorStroke color="0x0072bc" />
</s:stroke>
</s:Rect>
<s:Label id="labelDisplay"textAlign="center" maxDisplayedLines="1"
horizontalCenter="0" verticalCenter="1" verticalAlign="middle" left="10"
right="10" top="2" bottom="2" color="0xffffff" styleName="buttonFonts" >
</s:Label>
```

Na labelu se primenjuje stil buttonFonts, koji je definisan u fajlu style.css

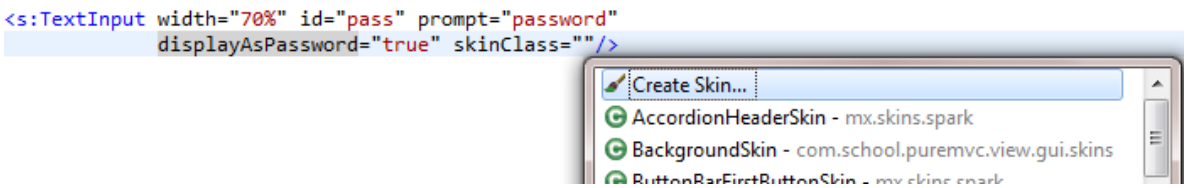
</s:SparkButtonSkin>

3.7.26 Klasa StudentTextInputSkin

U fajlu style.css definiše se stil za TextInput polje.

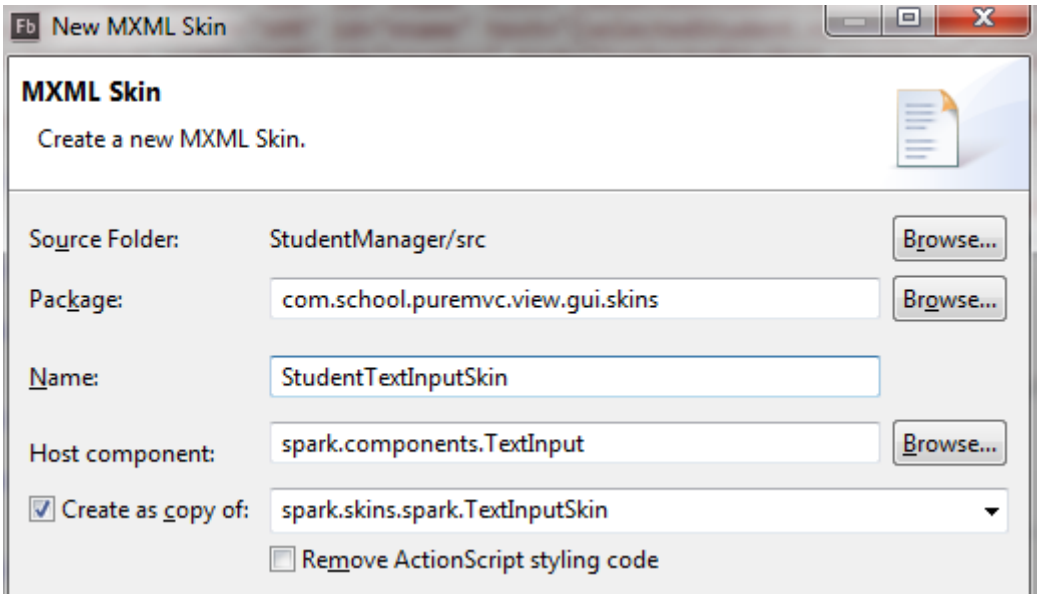
```
s|TextInput
{
	skinClass:ClassReference("com.school.puremvc.view.gui.skins.StudentTe
xtInputSkin");
	chromeColor:#fd7d22;
	focusColor:#fd7d22;
}
```

Skin klasa, se može kreirati u TextInput polje:



Slika 3.7.93. Kreiranje skin klase

Pritiskon na enter, dobija se dijalog box. U dijalogu definiše se paket u kojem se nalazi skin klasa, ime skin klase, roditeljska klasa, i odakle se kopira sadržaj u datu kreiranu klasu. Ime klase je StudentTextInputSkin.



Slika 3.7.94. Forma za definisanje mxml skin komponente StudentTextInputSkin

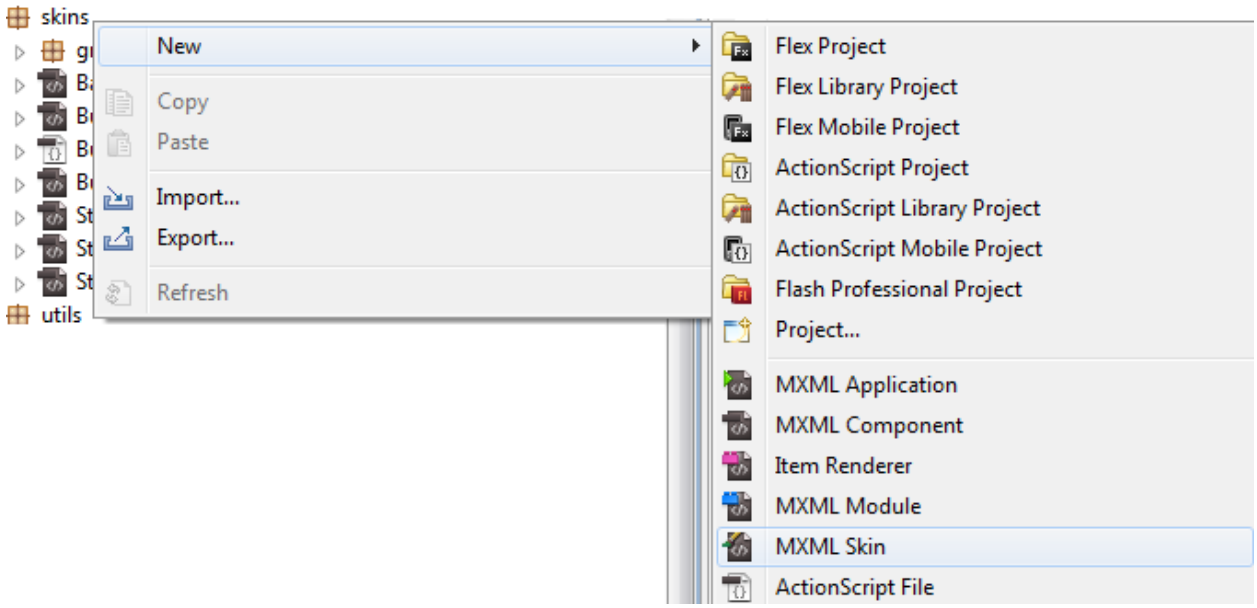
Kada se pritisne dugme finish, i kreira se klasa StudentTextInputSkin. Sada red skinClass="com.school.puremvc.view.gui.skins.StudentTextInputSkin" iz TextInput –a se premešta u Style.css fajlu da bi se primeno stil na sva TextInput polja u aplikaciji.

```
<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
<s:Label text="Email:" width="40%" styleName="labelText" color="0xff0000"/>
<s:TextInput width="60%" id="email" text="{selectedStudent.email}"
prompt="email"
skinClass="com.school.puremvc.view.gui.skins.StudentTextInputSkin"/>
</s:HGroup>
```

Sada TextInput polje ima oblik:

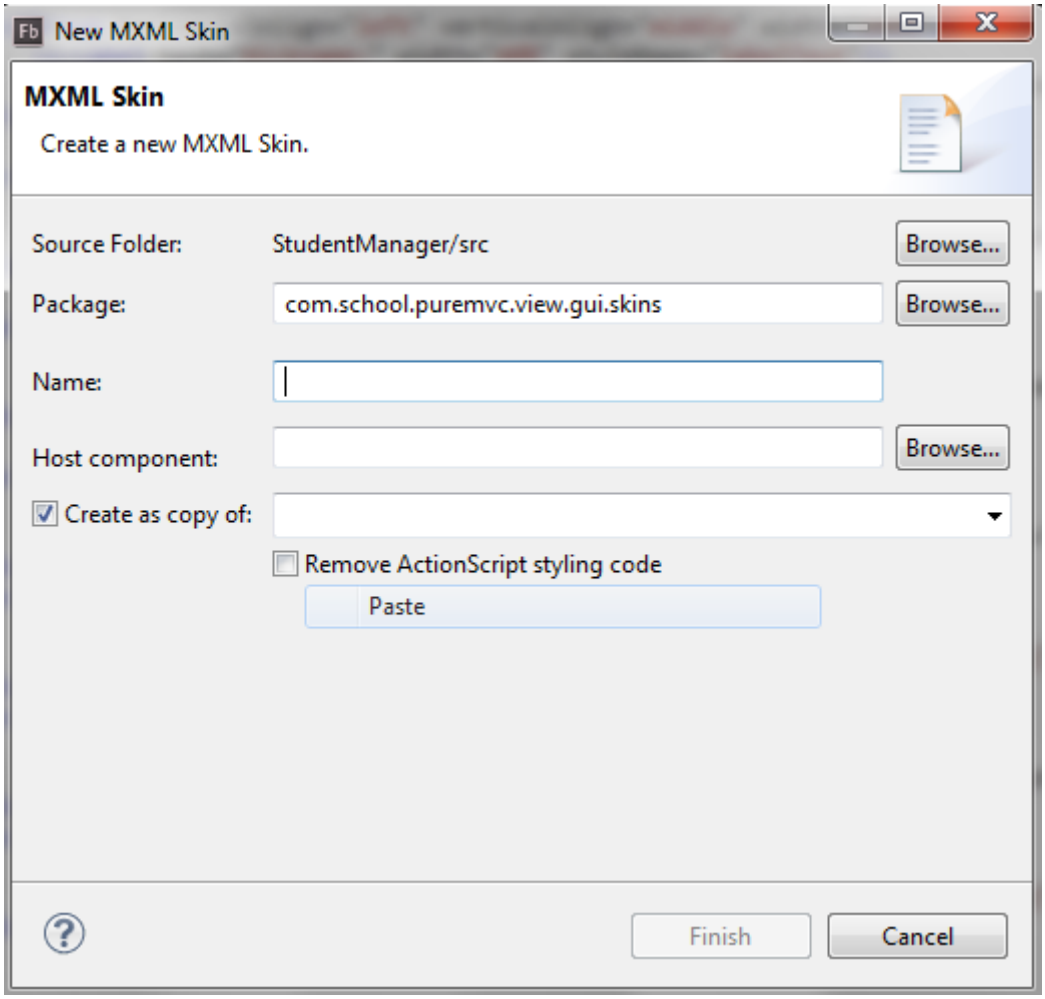
```
<s:HGroup horizontalAlign="left" verticalAlign="middle" width="100%">
<s:Label text="Email:" width="40%" styleName="labelText" color="0xff0000"/>
<s:TextInput width="60%" id="email" text="{selectedStudent.email}"
prompt="email"/>
</s:HGroup>
```

Klasa skinClass se koristi u TextInput da bi izgenerisali osnovni kod za klasu StudentTextInputSkin. Drugi način generisanja skin klase je da se u paketu skins kreira MXML Skin komponenta, kao na slici 3.7.95.



Slika 3.7.95. Prikazuje meni, pomoću kojeg se definiše mxml skin komponenta

Pritiskom na enter, dobija se dijalog box, gde je neophodno popuniti polja.



Slika 3.7.96. Forma za definisanje mxml skin komponente

Prethodni način je jednostavniji, jer je ponuđeno ime komponente koje se nasleđuje(Host component). Sada se u style.css definiše klasa s|TextInput:

```
s|TextInput
{
	skinClass:ClassReference("com.school.puremvc.view.gui.skins.StudentTe
	xtInputSkin")
}
```

Ovo znači, primeni sve što se nalazi u StudentTextInputSkin na skinClass, a skinClass, je skin klasa TextInput polja. Sada se vrši primena skina na sva TextInput polja. Delovi koda, iz klase StudentTextInputSkin su:

```
<s:RichEditableText id="textDisplay" verticalAlign="middle"
widthInChars="10"
```

```
              left="1" right="1" top="1" bottom="1"
styleName="buttonFonts"/>
<s:Label id="promptDisplay" maxDisplayedLines="1"
              verticalAlign="middle"
              mouseEnabled="false" mouseChildren="false"
              includeIn="normalWithPrompt,disabledWithPrompt"
              includeInLayout="false"
              />
</s:SparkSkin>
```

Prvo se definiše border. Border je jedan pravougaonik, koji ima samo jedan stroke, prozirnosti 0.6, debljine 1px i boje 0xd9e6f2. Margine su sa svih strana ravne nuli, što znači da je naslonjen na roditeljsku klasu, koja definiše pravougaonik.

```
<s:Rect left="0" right="0" top="0" bottom="0" id="border" alpha="0.6">
<s:stroke>
<!--- @private -->
<s:SolidColorStroke id="borderStroke" weight="1" color="0xd9e6f2"/>
</s:stroke>
</s:Rect>
```

U drugom koraku crta se novi pravougaonik, za background. Margine su po 1px sa leve desne strane, gore 1px, i dole 1px. To znači, ispod kontejnerskog pravougaonika crta se novi pravougaonik. Dakle, ovde postoji dva pravougaonika, jedan je border koji nije popunjen i drugi koji je popunjen belom bojom, je background.

```
<!-- fill -->
<s:Rect id="background" left="1" right="1" top="1" bottom="1">
<s:fill>
<s:SolidColor id="bgFill" color="0xd9e6f2" />
</s:fill>
</s:Rect>
```

Treća komponenta je RichEditableText klasa.

```
<!-- text -->
<s:RichEditableText id="textDisplay" verticalAlign="middle"
widthInChars="10" left="1" right="1" top="1" bottom="1"
styleName="buttonFonts"/>
```

Parametar verticalAlign, vrednosti "middle", ukazuje da se tekstualna komponenta nalazi na sredini po vertikali. Dok, osobina widthInChars="10" definiše podrazumevanu veličinu za tekst 10 em. Dakle, difoltna širina ove komponente merena u em jedinicu, po difoltu je 10 em karaktera. Parametri left="1" right="1" top="1" bottom="1" definišu mesto postavljanja komponente RichEditableText. Na komponentu RichEditableText definiše se nova komponenta Label. U ovoj komponenti osobina maxDisplayedLines="1" određuje maksimalan broj linija, to je jedna linija.

```
<s:Label id="promptDisplay" maxDisplayedLines="1"
                verticalAlign="middle"
                mouseEnabled="false" mouseChildren="false"
                includeIn="normalWithPrompt,disabledWithPrompt"
                includeInLayout="false"
                />
</s:SparkSkin>
```

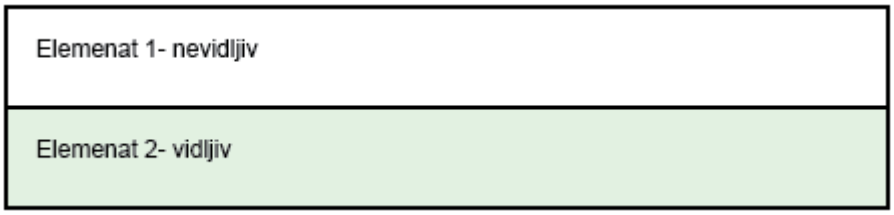
Postavljanjem vrednosti osobine mouseEnabled na "false", onemogućava se selekcija mišom. Dok, parameter mouseChildrensa vrednošću "false" isključuje mogućnost selekcije mišem njegove dece. Layout određuje raspored elemenata u komponenti. Osobina includeInLayout="false", znači da Label po difoltu nije uključen u raspored elemenata.

Tri stvari mogu uticati na vidljivost elemenata na ekranu:

1. Prva je osobina alpha, i ona služu samo za transparenciju.
2. Druga je osobina Visible, koja određuje da li je komponenta vidljiva ili ne.
3. I treća je includeInLayout. includeInLayout znači komponenta je uključena ili nije uključena u raspored. Dakle, ako je alpha nula, element postoji, samo je nevidljiv, transparentan je. Ako je Visible false, element opet postoji samo je potpuno nevidljiv. Osobina alpha može da uzme vrednost od 0 do 1, a visible ima vrednost false ili true, znači ili je nevidljiv ili vidljiv, nema nijansi kao kod alpha. Ali je tu, i utiče na druge elemente.

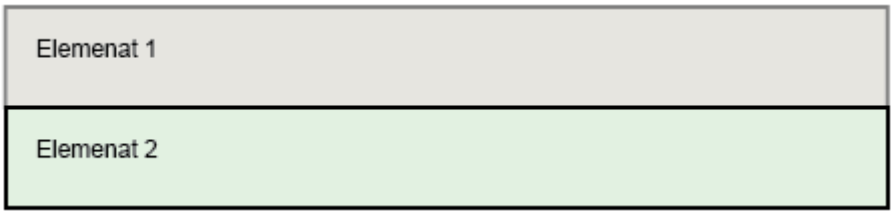
U narednom bloku koda definisan je VGroup kontejner, koji sadrži dva elementa. Osobina visible prvog elementa ima vrednost false a drugog true. Drugi element biće vidljiv, ali se nalazi ispod prvog elementa, i ako je on nevidljiv. On će u layout zauzimati neki prazan prostor, isto kao prazan prostor.

```
<VGroup>
<elemenat1 visible =false>
<elemenat2 visible =true>
</VGroup>
```



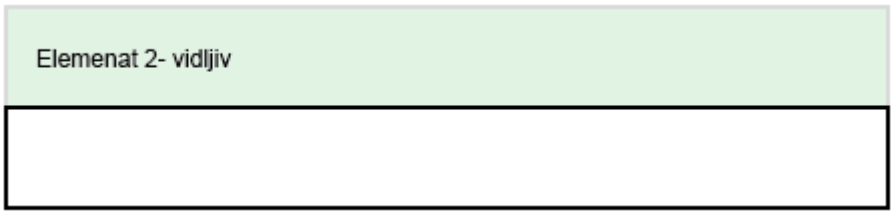
Slika 3.7.97. definiše dva elementa Element1 i Element2, kada osobina visible prvog elementa ima vrednost false a drugog true.

Ako je includeInLayout="false" za prvi elemenat, elemenat1 neće postojati, dok elemenat2 će se podići gore:



Slika 3.7.98. definiše formu od dva elementa Element1 i Element2.

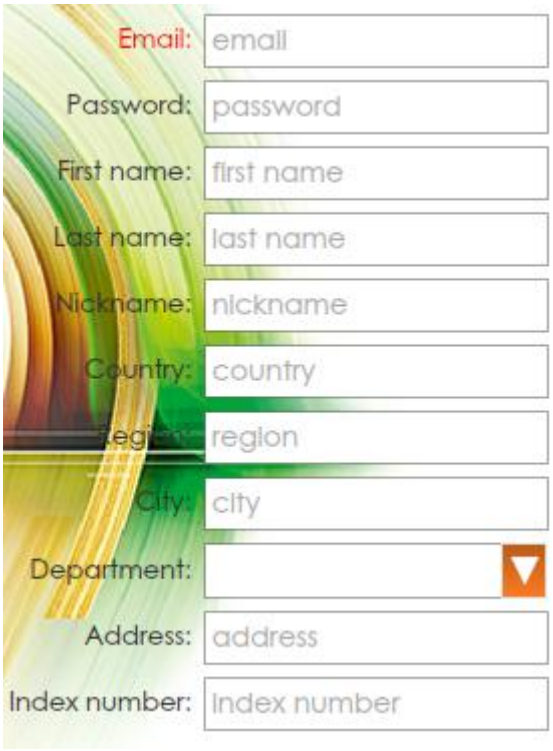
```
<VGroup>
<elemenat1 includeInLayout="false">
<elemenat2 includeInLayout="true">
</VGroup>
```



Slika 3.7.99. definiše izgled dva elementa Element1 i Element2, za slučaj da je osobina includeInLayout="false" za prvi elemenat.

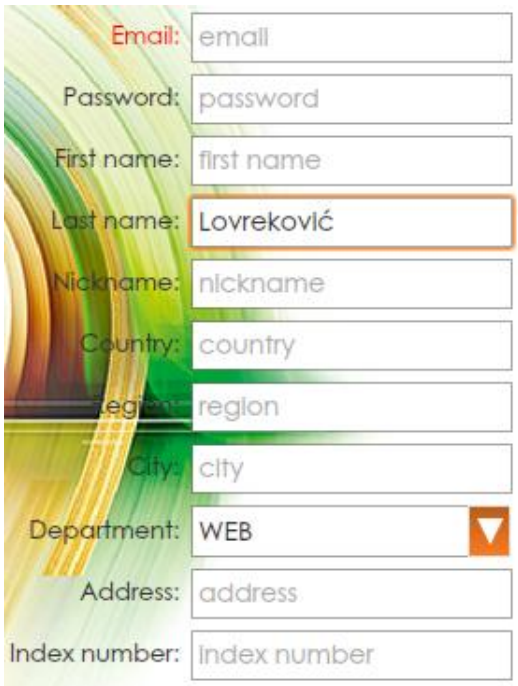
Osobina includeIn utiče na sva tri dela(alpha, visible i includeInLayout) u isto vreme. Naime, kada se kaže includeIn, to znači alpha=0, visible=false i includeInLayout=false. U ovoj klasi se nalazi i metoda updateDisplayList. updateDisplayList je jedna od poslednjih metoda, koja se poziva da se apdejtuje i prikaže komponenta. Ako je stanje normalno, inkluduje se u

"normalWithPrompt, disabledWithPrompt". U praksi prompt je ono što piše u TextInput polje, email, ili password itd, što je prikazano na slici3.7.100.



Slika 3.7.100. prikazuje izgled TextInput komponenti u aplikaciji.

A RichEditableText je ono što piše u TextInput polje:



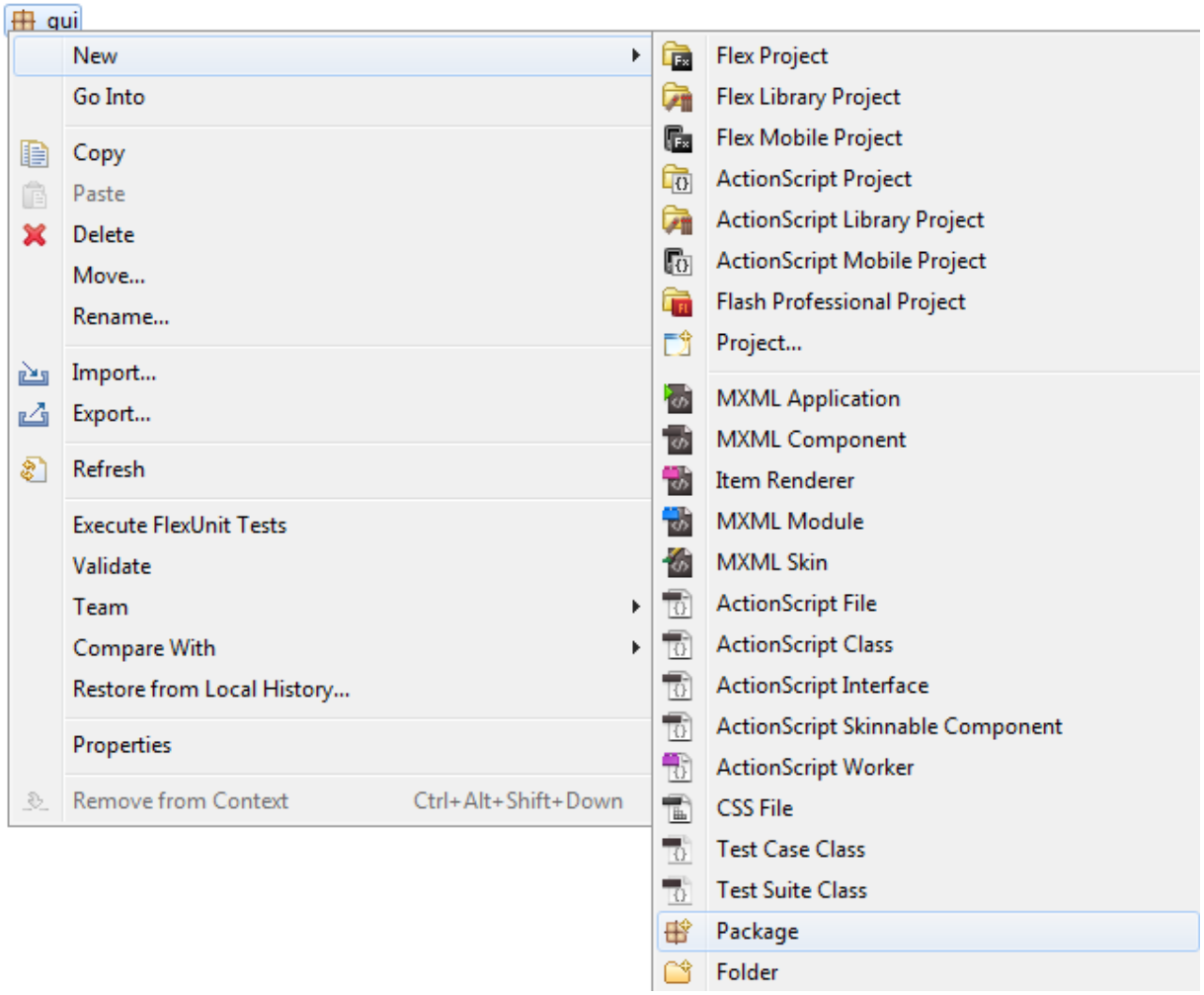
Slika 3.7.101. prikazuje izgled TextInput komponenti u aplikaciji, kada su dva polja popunjena.

Naime, ovde postoje dve vrste teksta, prvi je RichEditableText, to je ono što je upisano a iznad se nalazi prompt. Prompt je vidljiv samo ako se definiše, a definiše se u TextInput polju kao osobina:

```
<s:TextInput width="70%" id="pass" prompt="password"
displayAsPassword="true"/>
```

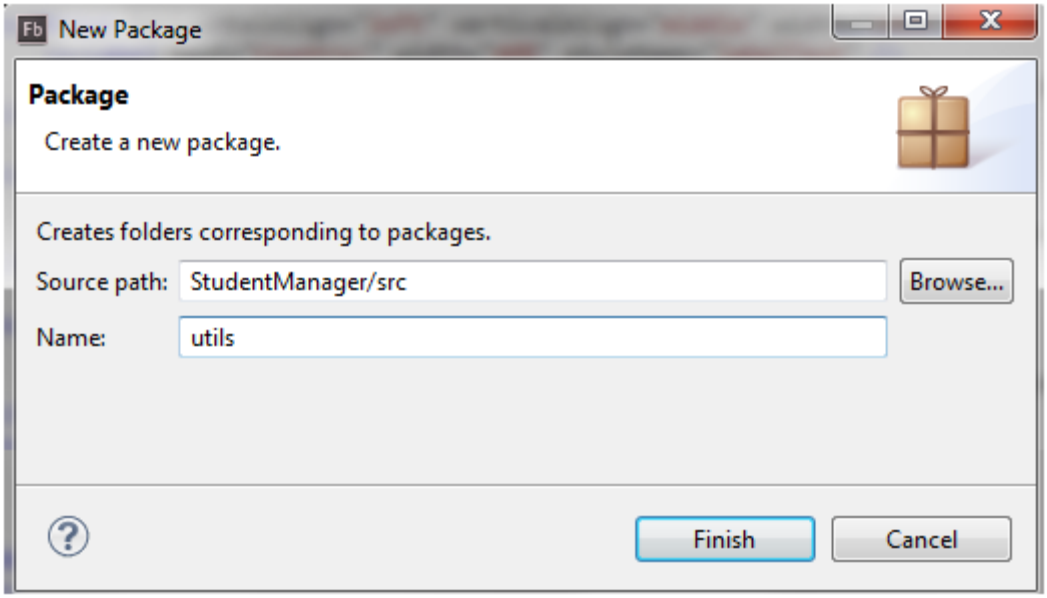
3.7.27 Pozadina aplikacije

Klasa ImageUtils koja definiše pozadinu aplikacije, nalazi se u paketu utils, koji je smešten unutar paketa gui. Način kreiranja paketa utils je prikazan narednom slikom.



Slika 3.7.102. prikazuje način kreiranja paketa utils.

Klikom na stavku Package, dobija se dijalog box:

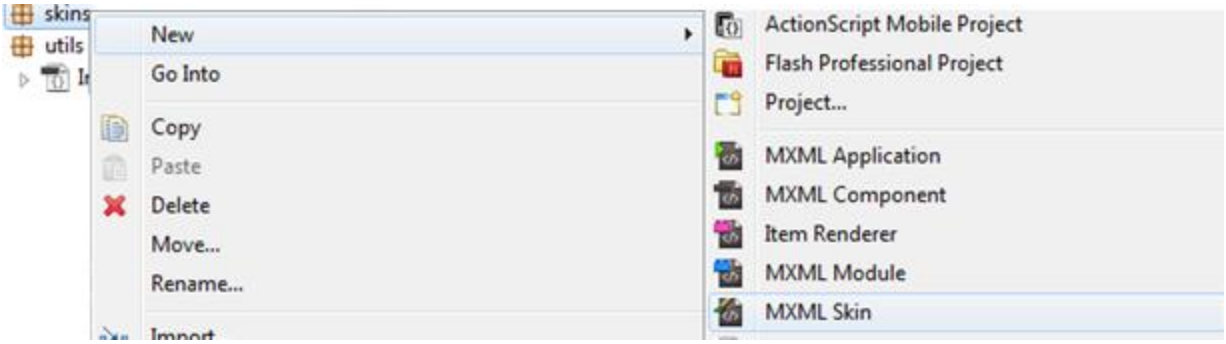


Slika 3.7.103. Dijalog za definisanje paketa.

U klasi ImageUtils se embeduje slika. U paketu assets, unutar foldera images, arhivira se slika background.jpg, koja predstavlja pozadinu aplikacije. Unutar koda, parametar source sadrži putanju slike za pozadinu. Dakle, u klasi ImageUtils, napravljena je javna konstanta backgroundImage tipa Class. Za konstantu backgroundImage, zalepljen je image fajl. Znači od ovog trenutka konstanta backgroundImage, postaje background.jpg. Class je je generiočka klasa u AS3 jeziku. Naime, Ovde je dodat fajl.

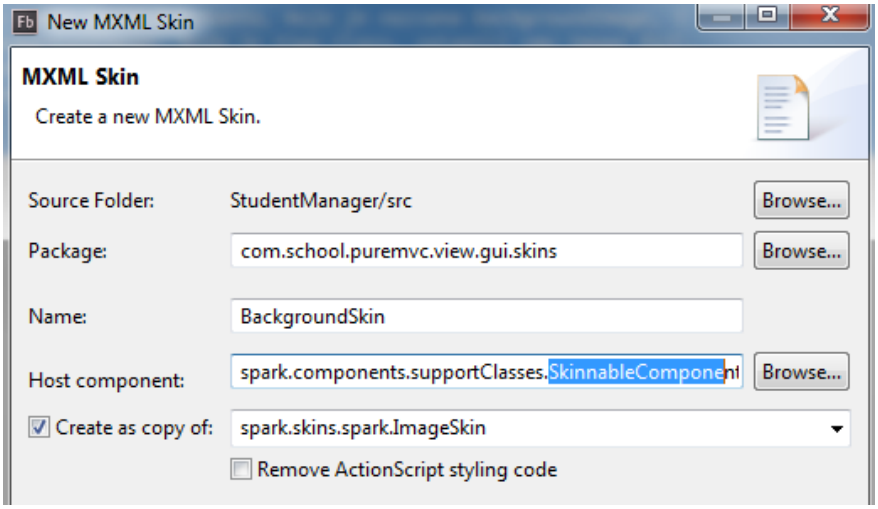
```
package com.school.puremvc.view.gui.utils
{
	public class ImageUtils
	{
		[Embed(source="assets/images/background.jpg")]
		public static const backgroundImage:Class;
		[Embed(source="assets/images/arrow.png")]
		public static const dropDown:Class;
	}
}
```

Sada se u paket skins pravi novi skin BackgroundSkin:



Slika 3.7.104. Prikazuje meni, za kreiranje skin komponente.

Za Host component, bira se SkinnableComponent[1]. To je najlakša klasa koja postoji.



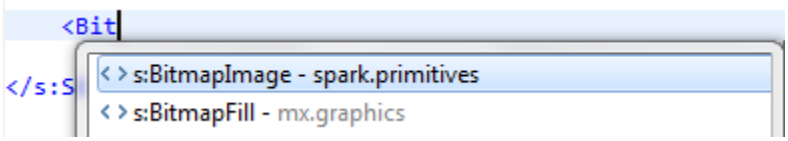
Slika 3.7.105. Dijalog za definisanje imena, paketa, osnovne klase, za klasu BackgroundSkin.

Kod ove klase je:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
            xmlns:s="library://ns.adobe.com/flex/spark"
            xmlns:mx="library://ns.adobe.com/flex/mx">
      <!-- host component -->
      <fx:Metadata>
      [HostComponent("spark.components.supportClasses.SkinnableComponent")]
      </fx:Metadata>
</s:Skin>
```

SkinnableComponent nema ništa u sebi. Naime, SkinnableComponent je prazan prostor koji može da se skinuje. U ovoj klasi pravi se BitmapImage. BitmapImage nasleđuje iz paketa spark

Image. Dakle, BitmapImage je kontejner gde se mogu prikazati image fajlovi t.j. slike, ali ne može ništa da se radi sa njima, primitivan je, samo služi za prikazivanje. Postoji da olakša, da se ne ubacuju unutar aplikacije nepotrebne stvari. Idealan je za background-e i za ikonice.



Slika 3.7.106. prikazuje meni, za odabir klase BitmapImage klase.

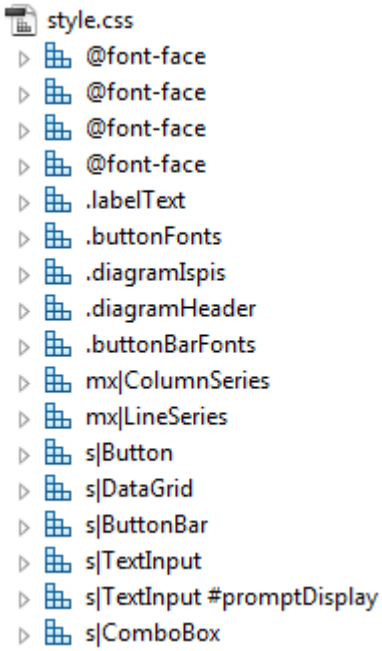
```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
            xmlns:s="library://ns.adobe.com/flex/spark"
            xmlns:mx="library://ns.adobe.com/flex/mx">
      <!-- host component -->
      <fx:Metadata>
      [HostComponent("spark.components.supportClasses.SkinnableComponent")]
      </fx:Metadata>
      <fx:Script>
            <![CDATA[
                  import com.school.puremvc.view.gui.utils.ImageUtils;
            ]]>
      </fx:Script>
      <s:BitmapImage source="{ImageUtils.backgroundImage}" width="100%"
      height="100%"/>
</s:Skin>
```

Iz klase ImageUtils pristupa se elementu koji sadrži sliku. Širina i visina elementa koji se učitava postavljena je na 100%. U fajl StudentAMF, na samom početku stavi se SkinnableComponent.

```
<s:SkinnableComponent
            width="100%" height="100%"
            skinClass="com.school.puremvc.view.gui.skins.BackgroundSkin"
            />
```

3.7.28 Style.css

U ovom fajlu se definišu stilovi za kontrole u aplikaciji. Struktura fajla je pikazana na slici3.7.122.



Slika 3.7.107. Prikaz svih korisničko definisanih stilova u fajlu Style.css

3.7.29 Klasa StudentComboSkin

ComboBox se kreira u fajlu ResultDiagrams.mxml. Da bi se kreirani stil primenio na sve ComboBox-ve, onda se u style.css fajlu kreira stil. ComboBox čine više elemenata, TextInput, dugme i jedna lista, kao padajući meni. U izgledu može da se menja TextInput. Ovde je primenjen skin klasa na dugmad i TextInput polja.

3.7.30Klasa ButtonBarCustomButton

Svaki ButtonBar ima svoj podatak. ButtonBar kontrola predstavlja grupu logički povezanih dugmadi sa zajedničkim izgledom i navigacijom. Pošto je to grupa, deo ButtonBar-a je jedan Button. Klasa ButtonBar ima data. U data su podaci za obradu ili editovanje. Ova osobina može da se koristi kao source za date binding. Kada se ova osobina modifikuje onda se dispečuje

dataChange ivent. Naime, data je deo DataProvider-a. Prema tome, ButtonBar ima svoj dataProvider, koji je array.

```
package com.school.puremvc.view.gui.skins
{
	import spark.components.ButtonBarButton;

	public class ButtonBarCustomButton extends ButtonBarButton
	{
		public function ButtonBarCustomButton() {
			super();
		}
```

Osobina data je ugrađena promenljiva koja je objekat i dobija se iz Array liste.

```
override public functionset data( value : Object ) : void {
	super.data = value;
```

U kodu se proverava da li objekat koji je stigao ima property, koja se zove "enabled". Naime, provera se ArrayList u klasi AplicationFacade.

```
			try {
				if ( value.hasOwnProperty( "enabled" ) ) {
					enabled = data.enabled;
				}
			}
			catch(err:Error){
				trace(err)
			}
		}
	}
}
```

Enabled na levoj strani, u izrazu enabled = data.enabled; je deo koji je nasleđen iz ButtonBarButton. hasOwnProperty je sistemska promenljiva, koja omogućava da se nešto disejbluje ili enebluje.

3.7.32 Klasa StudentiAPI.php

Ova klasa se nalazi na serverskoj strani, realizovana je u PHP-u i služi da opsluži klijentsku stranu kada se koristi format AMF za podatke, koji razmenjuju podatke između servera i klijenta. Klasa je smeštena u fajlu StudentiAPI.php, sadrži sedam metoda, koje su date u listi.

- *__construct*
- *getAllStudenti*
- *changeStudentiData*
- *deleteStudent*
- *addNewStudent*
- *filterStudents*
- *getDepartments*

Na početku fajla vrši se učitavanje fajlova, naredbom require_once[5]. Unutar ove funkcije navodi se staza fajla koji se učitava. U konstruktor funkciji __construct, vrši se povezivanje sa bazom podataka, sa kojom ova klasa komunicira.

```
<?php
     require_once("includes/Setup.php");
     require_once("vo/com/school/vo/StudentVO.php");
     require_once("vo/com/school/vo/DepartmentVO.php");
     require_once("vo/com/school/responses/StudentResponse.php");

     class StudentiAPI {
           public function __construct(){
                 $setup = new Setup();
           }
```

Sve metode u ovoj klasi moraju biti public ako se pozivaju iz Flex aplikacije. Svaka metoda koja se poziva mora vraćati neku vrednost.

Metod *getAllStudenti*se poziva iz Flex-a, poseduje dva parametra. Prvi parameter definiše početak čitanja podataka iz tabele baze podataka, dok drugi parameter oderđuje koliko se podataka učitava. PHP metod mysql_query na osnovu upita kreira resurs u memoriji[5,6,68]. Početak resursa se čuva u promenljivoj $result. Resurs sadrži podatke iz tabele.

```
public function getAllStudenti($start,$requested){
$result = mysql_query("SELECT users.id, users.email, users.password,
users.first_name,users.last_name,users.nickname,users.country,
users.region,users.city,users.department, users.address, users.created_at,
users.updated_at, users.index_num, department.name, department.id as depid
FROM users inner join department on department.id = users.department LIMIT
$start, $requested");
```

Naredba inner join, je ugrađena funkcija u php-u koja spaja dve tabele i to na tabelu users spaja department, *on* je uslov, department.id su brojevi i users.department su brojevi[45]. Deo koda *department.id as depid* znači da se department.id posmatra pod novim imenom a to je depid. Da nije ovako napisano, onda bi postojala dva id-a iz dve tabele usrs i department. Da se to ne bi desilo, onda je za id iz tabele department uvedeno drugo ime, a to je depid. Izraz FROM users inner join department on department.id = users.department, znači spoj tabelu users sa tabelom department pod uslovom(*on* je uslov) da je department.id = users.department, jer users.department su brojevi. Lokalna promenljiva *$resultArray* je tipa array. U while petlji, funkija mysql_fetch_object čita po jedan red iz resursa $result, pretvara ga u objekt i smešta ga u $row. Naredbom $medc = new StudentVO(); kreira se privremeni objekat $medc, koji sadrži osobine, sa istim imenom kao i polja u tabeli baze podataka. Polja objekta $medc se popunjavaju iz objekta $row. Popunjeni objekat $medc smešta se u niz $resultArray.

```
$resultArray = array();
$response = new StudentResponse();
     try {
           while ($row = mysql_fetch_object($result)){
                 $medc = new StudentVO();
                 $medc->address = $row->address;
                 $medc->city = $row->city;
                 $medc->country = $row->country;
                 $medc->created_at = $row->created_at;
                 $medc->department = $row->depid;
                 $medc->email = $row->email;
                 $medc->first_name = $row->first_name;
                 $medc->id = $row->id;
                 $medc->index_num = $row->index_num;
                 $medc->last_name = $row->last_name;
                 $medc->nickname = $row->nickname;
```

```
            $medc->password = $row->password;
            $medc->region = $row->region;
            $medc->updated_at = $row->updated_at;
            $medc->country = $row->country;
            $resultArray[] = $medc;
        }
        $result2 = mysql_query("SELECT count(id) as total FROM users");
        $response->result = $resultArray;
        $response->totalNumber = mysql_fetch_object($result2)->total;
```

Metod count(id) je ugrađena funkcija u mysql-u, koja vraća broj nečega. Naredbom SELECT count(id) traži se da vrati broj svih upisa u tabelu. Izraz SELECT count(id) as total, broj upisa se smešta u polje total, dakle kreira se polje total u svojoj privremenoj memoriji, i upisuje se u njega broj koji je izračunat. Naime, mysql_fetch_object($result2)->total formira objekat na osnovu rezultata result2. Dakle, naredbom SELECT count(id) as total FROM users, formira se objekat, a polje u tom objektu zove se total. Objektu nije dato ime. Deo koda se može realizovati i na drugi način, koristeći naredne naredbe:

```
$result2 = mysql_query("SELECT count(id) as total FROM users");
$response->result = $resultArray;
$object=mysql_fetch_object($result2);
$response->totalNumber =$object->total;
```

Ovde je napravljena promenljiva $object, prosleđena joj je vrednost mysql_fetch_object($result2) od upita $result2. $result2 je tipa resurs. Funkcija mysql_query vraća kao rezultat objekat tipa resurs. Naime, mysql_query šalje jedinstveni upit u bazu podataka. Kao rezultat za SELECT, SHOW, DESCRIBE, EXPLAIN i druge deklaracije vraća se resultset. Resurs je specijalan tip objekata u php-u, od slučaja do slučaja je drugačiji. Za upite SELECT, SHOW, DESCRIBE, EXPLAIN kao rezultat, vraća resurs. Dok, za upite kao što su INSERT, DELETE, DROP, mysql_query vraća true ili false. Resurs koji se dobije, zavisno od upita, mogu da se koriste funkcije kao što su mysql_fetch_array(), gde se resurs pretvora u array. Funkcija mysql_fetch_row() pretvara resurs u redove, koji su veoma slični kao array, dok mysql_fetch_assoc() resurs transformiše u asocijativni array. U ovom slučaju koristi se mysql_fetch_object() koja resurs pretvora u niz objekata. Može da se koristi mysql_num_rows() kako bi se dobio broj rezultata. Promenljiva $result2, je ništa drugo do tip resurs. Sa resursom ne može da se uradi puno toga, već se moraju koristiti neke od

ugrađenih mysql_fetch funkcija, da se pretvori potrebni resurs, kao na primer mysql_fetch_object($result2). Kada je izvršena mysql_fetch_object funkcija, prosledivši joj $result2 kao parametar, dobija se php $object.

$object- se zove standard class objekat u php-u, i na ovaj način se obraćamo osobini: $object->total

```
            }
            catch(Exception $e){
                  return $e->getMessage();
            }
                  //cilj je da rezultat bude array ili lista objekata
            return $response;
      }
```

Metod *changeStudentiData* vrši ažururanje baze podataka.

```
public function changeStudentiData($student){
$result = mysql_query("UPDATE users SET email='$student-
>email',password='$student->password', first_name='$student-
>first_name',last_name='$student-last_name',nickname='$student->nickname',
country='$student->country', region='$student->region',city='$student-
city', department='$student-> department',address='$student->address',
updated_at=now(), index_num='$student->index_num' WHERE id='$student-
>id'");
return $result;
}
```

Metod *deleteStudent*vrši brisanje sloga u bazi podataka.

```
public function deleteStudent($studentId){
      $result = mysql_query("delete from users where id = '$studentId'");
      return $result;
}
```

Metod *addNewStudent* vrši dodavanje novog sloga u bazu podataka.

```
public function addNewStudent($student){
      $result = mysql_query("INSERT INTO users(email, password, first_name,
last_name, nickname, country, region, city, department, address,
created_at, updated_at, index_num) VALUES ('$student-> email','$student-
>password','$student->first_name','$student->last_name','$student-
```

```
>nickname','$student->country','$student->region','$student-
>city','$student->department','$student->address', now(), now(),'$student-
>index_num')");
            if ($result)
                  return mysql_insert_id();
            else
                  return false;
      }
```

Funkcija filterStudents se koristi za filtriranje podataka, njoj se prosleđuje šest parametara $name, $address, $email, $start, $requested i $department. U prvom redu se postavlja string promenljiva $query="" na prazan string. U if naredbi if($name!= "") proverava se da li je promenljiva $name prazna, ako nije onda se formira izraz za $query. U ovom izrazu se proverava da li ono što je upisano u $name sadrži promenljiva first_name ili last_name na početku ili kraju ili negde nasredini, dakle bilo gde. Gde je first_name ili last_name kolone u tabeli baze podataka.

Ako se koristi Like i procenat, to je regularni izraz. Preoverava se da li se traženi string nalazi u tekst.Za konstrukciju,

```
$query.=”address like ‘$address%’ “;
```

znači početak mora biti takav a na kraju je sve jedno.

```
$query.=”address like ‘%$address’ “;  Ovde početak nije bitan a kraj mora da
se završava tako.
```

```
 public function
 filterStudents($name,$address,$email,$start,$requested,$department){
      $query = "";
      if ($name != ""){
            $query .= " (users.first_name like '%$name%' OR users.last_name
 like '%$name%') ";
            }
            if ($address != ""){
                  if ($query == ""){
                        $query .= " users.address like '%$address%' ";
                  }
                  else {
                        $query .= " and users.address like '%$address%' ";
                  }
```

```
        }
        if ($email != ""){
              if ($query == ""){
                    $query .= " users.email = '$email' ";
              }
              else {
                    $query .= " and users.email = '$email' ";
              }
        }

        if ($department != ""){
              if ($query == ""){
                    $query .= " users.department = '$department' ";
              }
              else {
                    $query .= " and users.department = '$department' ";
              }
        }
$result = mysql_query("SELECT users.id, users.email, users.password,
users.first_name, users. last_name, users.nickname,
users.country,users.region, users.city, users.department, users.address,
users.created_at, users.updated_at, users.index_num,department.name,
department.id as depid FROM users inner join department on department.id =
users.department WHERE $query LIMIT $start, $requested");
```

Ovde se vrši pretraga u bazi start(npr prvog) pa do maksimalno requested. Ali kada dođe do 500–totog rezultata, stane se sa pretragom. Ovde ako je start=1 a request =500, traži se pretraga od prvog pa dok se ne nađe max 500 komada i tu se stane, stim što zadovojavaju dati uslov. Ovde ako je start=1 a request =500, on ne traži dati kriterijum od 1 do 500, nego počinje od 1 a traži 500, ako u prvih pesto nađe 5 koji zadovoljavaju kriterijum, on se prebacuje na drugih 500 i traži koliko zadovoljava kriterijum, sve dok ne popuni kvotu od 500. Kada popuni ukupno 500 onda će ih vratiti.

```
$resultArray = array();
$response = new StudentResponse();
try {
      while ($row = mysql_fetch_object($result)){
            $medc = new StudentVO();
            $medc->address = $row->address;
            $medc->city = $row->city;
```

```
                $medc->country = $row->country;
                $medc->created_at = $row->created_at;
                $medc->department = $row->depid;
                $medc->email = $row->email;
                $medc->first_name = $row->first_name;
                $medc->id = $row->id;
                $medc->index_num = $row->index_num;
                $medc->last_name = $row->last_name;
                $medc->nickname = $row->nickname;
                $medc->password = $row->password;
                $medc->region = $row->region;
                $medc->updated_at = $row->updated_at;
                $medc->country = $row->country;
                $resultArray[] = $medc;
          }
$result2 = mysql_query("SELECT count(id) as total FROM users WHERE
$query");
          $response->result = $resultArray;
          $response->totalNumber = mysql_fetch_object($result2)->total;
     }
     catch(Exception $e){
          return $e->getMessage();
     }
     //cilj je da rezultat bude array ili lista objekata
     return $response;
}
```

Metod getDepartments čita podatke u tabeli department.

```
public function getDepartments(){
     $result = mysql_query("SELECT department.id, department.name from
department");
          $resultArray = array();
          while ($row = mysql_fetch_object($result)){
                $dep = new DepartmentVO();
                $dep->id = $row->id;
                $dep->name = $row->name;
                $resultArray[] = $dep;
          }
          return $resultArray;
     }
}
?>
```

3.7.33 Klasa Setup

Ova klasa uspostavlja konekciju sa bazom podataka[45]. Na početku fajla Setup.php definišu se konstante, koje se koriste u klasi.

```
<?php

	define('HOST', 'localhost');
	define('USERNAME', 'root');
	define('PASSWORD', '');
	define('DB', 'student');

	class Setup
	{
		public function __construct()
		{
			mysql_connect(HOST, USERNAME, PASSWORD);
			mysql_select_db(DB);
		}
	}
?>
```

3.7.34 Klasa StudentVO

Ova klasa prihvata vrednosti iz jednog reda u bazi podataka. Broj osobina odgovara broju kolona u tabeli users. Instance ove klase se koriste za transfer podataka od server do klijenta, stim što postoji identična klasa na serverskoj strain, tako da se vrši mapiranje podataka sa serverske strane na klijentsku stranu i obrnuto.

```
<?php
	class StudentVO
	{
		public $id;
		public $email;
		public $password;
		public $first_name;
		public $last_name;
		public $nickname;
		public $country;
		public $region;
```

```
            public $city;
            public $department;
            public $address;
            public $created_at;
            public $updated_at;
            public $index_num;
```

U narednoj liniji koda, definiše se jedinstveno ime klase, koje mora biti identično u Flex aplikacjiji

```
            var $_explicitType = "com.school.vo.StudentVO";
      }
?>
```

3.7.35 Klasa DepartmentVO

Ova klasa sadrži dve osobine, istog naziva kao i kolone u tabeli DepartmentVO. Instanca ove klase sadrži vrednosti sloga iz tabele DepartmentVO.

```
<?php
class DepartmentVO
{
      public $id;
      public $name;
```

Ovde se definiše jedinstveno ime klase, koje mora biti identično u Flex aplikacjiji

```
      var $_explicitType = "com.school.vo.DepartmentVO";
      }
?>
```

3.7.36 Klasa StudentResponse

Klasa StudentResponse sadrži dva polja. Prvo polje sadrži podatke iz baze koji se transportuju do klijenta i druga osobina sadrži ukupan broj pročitanih podataka iz baze.

```
<?php
      class StudentResponse {
            public $result;
            public $totalNumber;
            var $_explicitType = "com.school.responses.StudentResponse";
```

```
    }
?>
```

3.7.37 Klasa StudentiAPIJson

Klasa StudentiAPIJson sadrži sve potrebne metode za čitanje određenog broja slogova iz baze podataka, ažuriranje reda u tabeli, brisanje željenog reda, dodavanje novog sloga o studentu. U konstruktoru se vrši povezivanje sa bazom podataka.

```
<?php
    header('Access-Control-Allow-Origin: *');
    header('content-type: application/json; charset=utf-8');
    require_once("includes/Setup.php");

class StudentiAPIJson {
    public function __construct(){
    $setup = new Setup();
    }
```

Metod getAllStudenti čita podatke o studentima počevši od reda $start i čita naredni broj $requested slogova. Pročitane slogove vraća kao resurs, koji se obrađuju I kao niz objekata vraćaju klijentskoj strani.

```
public function getAllStudenti($start,$requested){
$result = mysql_query("SELECT id, email, password, first_name, last_name,
nickname, country, region, city, department,address, created_at,
updated_at, index_num FROM users LIMIT $start, $requested");
```

Ovde se pravi arrar za smeštanje rezultata.

```
    $resultArray = array();
```

U while petlji vrši se iteracija kroz rezultat.

```
    while ($row = mysql_fetch_assoc($result)){
          $resultArray[] = $row;
    }
```

Instanca $obj služi da rezultat posatne objekat sa dva elementa (trenutna struktura).

```
    $obj['result'] = $resultArray;
    $obj['status'] = true;
    $result2 = mysql_query("SELECT count(id) as total FROM users");
    $obj['totalNumber'] = mysql_fetch_object($result2)->total;
```

Ugrađena php funkcija json_encode je pravi json format od datog parametra echo json_encode($obj);

```
}
```

Metod addNewStudent dodaje novi element u bazi podataka.

```
public function addNewStudent($studentString){
    $student = json_decode($studentString);
    //json_decode je mysql funkcija, koja pretvara json string u objekat
$result = mysql_query("INSERT INTO users(email, password, first_name,
last_name, nickname,country, region, city, department, address,
created_at, updated_at, index_num) VALUES ('$student->email','$student-
>password',' $student->first_name','$student->last_name','$student-
>nickname',' $student->country','$student->region','$student-
>city','$student->department', '$student->address',now(),now(),'$student-
>index_num')");
```

Funkcija mysql_insert_id() je ugrađena u PHP koja vraća id poslednjeg unosa u bazu podataka[5,6].

```
if ($result){
        $obj['result'] = mysql_insert_id();
        $obj['status'] = true;
    }
    else {
        $obj['result'] = NULL;
        $obj['status'] = false;
    }
    echo json_encode($obj);
}

public function deleteStudent($studentId){
```

```
	$result = mysql_query("delete from users where id = '$studentId'");
	if ($result){
		$obj['result'] = "";
		$obj['status'] = true;
	}
	else {
		$obj['result'] = NULL;
		$obj['status'] = false;
		$obj['error'] = mysql_error();
	}
	echo json_encode($obj);
}
```

Metod changeStudentiData vrši izmenu sadržaja u bazi podataka.

```
public function changeStudentiData($studentString){
	$student = json_decode($studentString);
	$result = mysql_query("UPDATE users SET email='$student->email',
	password='$student->password',first_name='$student->first_name',
	last_name='$student->last_name',nickname='$student->nickname',
	country='$student->country',region='$student->region',city='$student-
	> city', department='$student-> department',address='$student-
	>address', updated_at=now(),index_num='$student->index_num' WHERE
	id='$student->id'");

	if ($result){
```

$obj['result'] je prazan String zato što ne treba nikakav rezultat u aplikaciji odakle je ovaj method pozvan, dovoljna je samo informacija da je 'status' true, tj. operacija uspešno izvršena, a zadržavamo ga zbog čuvanja postojeće strukture Json objekta.

```
		$obj['result'] = "";
		$obj['status'] = true;
	}
	else {
		$obj['result'] = NULL;
		$obj['status'] = false;
	}
	echo json_encode($obj);
}
	}
```

Ovaj deo je van klase.

```
    if(isset($_REQUEST['action'])){
        $action = $_REQUEST['action'];
        $student = new StudentiAPIJson();

        if ($action == "getAll"){
            $start = $_REQUEST['start'];
            $requested = $_REQUEST['requested'];
            $student->getAllStudenti($start,$requested);
        }
        else if($action == "add"){
            $studentFlex = $_REQUEST['studentFlex'];
            $student->addNewStudent($studentFlex);
        }
        else if ($action == "delete"){
            $studentId = $_REQUEST['studentId'];
            $student->deleteStudent($studentId);
        }
        else if($action == "change"){
            $studentFlex = $_REQUEST['studentFlex'];
            $student->changeStudentiData($studentFlex);
        }
    }
?>
```

3.7.38 Klasa StudentAPIXml

Klasa StudentAPIXml poseduje javne metode, koje opslužuju klijenta. Metode ove klase pristupaju bazi podataka, čitaju ppdatke iz baze, menjaju sadržaj u bazi, dodaju novi sadržaj i brišu postojeći sadržaj. U konstrukt funkciji ove klase uspostavlja se veza sa bazom podataka.

```
<?php
    header('Access-Control-Allow-Origin: *');
    header('content-type: text/xml; charset=utf-8');
    require_once("includes/Setup.php");

class StudentAPIXml {
```

```
public function __construct(){
      $setup = new Setup();
}
```

Metod getAllStudenti, pristupa bazi podataka, čita zahtevani broj slogova i vraća iz baze resurs, koji se formatira i šalje klijentu[5,6,67].

```
public function getAllStudenti($start,$requested){

      $result = mysql_query("SELECT id, email, password, first_name,
      last_name, nickname, country, region, city, department, address,
      created_at, updated_at, index_num FROM users LIMIT $start,
      $requested");
            //napravimo arrar za smestanje rezultata
            $xml = "<root>";
            $xml .= "<result>";
            //iteracija kroz rezultat
      while ($row = mysql_fetch_assoc($result)){
            $xml .= "<student>";
            $xml .= "<id><![CDATA[".$row['id']."]]></id>";
            $xml .= "<email><![CDATA[".$row['email']."]]></email>";
            $xml .= "<first_name><![CDATA[".$row['first_name']."]]>
 </first_name>";
            $xml .= "<last_name><![CDATA[".$row['last_name']."]]>
</last_name>";
            $xml .= "<nickname><![CDATA[".$row['nickname']."]]>
                  </nickname>";
            $xml .= "<country><![CDATA[".$row['country']."]]></country>";
            $xml .= "<region><![CDATA[".$row['region']."]]></region>";
            $xml .= "<city><![CDATA[".$row['city']."]]></city>";
            $xml .= "<department><![CDATA[".$row['department']."]]>
                  </department>";
            $xml .= "<address><![CDATA[".$row['address']."]]></address>";
            $xml .= "<created_at><![CDATA[".$row['region']."]]>
                   </created_at>";
            $xml .= "<updated_at><![CDATA[".$row['updated_at']."]]>
                  </updated_at>";
            $xml .= "<index_num><![CDATA[".$row['index_num']."]]>
                  </index_num>";
            $xml .= "</student>";
```

```
            }
            $xml .= "</result>";
            $result2 = mysql_query("SELECT count(id) as total FROM users");
            $xml .= "<totalNumber>".mysql_fetch_object($result2)->total.
            "</totalNumber>";
            $xml .= "<status>true</status>";
            $xml .= "</root>";
            echo $xml;
      }
```

Metod addNewStudent dodaje novi element u bazi.

```
public function
addNewStudent($email,$password,$first_name,$last_name,$nickname, $country,
$region, $city,$department,$address,$index_num){
      $result = mysql_query("INSERT INTO users(email, password, first_name,
      last_name, nickname, country, region, city, department, address,
      created_at, updated_at, index_num) VALUES
      ('$email','$password','$first_name','$last_name','$nickname','$countr
      y','$region', '$city', '$department','$address',now(),
      now(),'$index_num')");
                  $xml = "<root>";
                  if ($result == true){
                        $xml .= "<result>".mysql_insert_id()."</result>";
                        $xml .= "<status>true</status>";
                  }
                  else {
                        $xml .= "<result>null</result>";
                        $xml .= "<status>false</status>";
                  }
                  $xml .= "</root>";
                  echo $xml;
            }
```

Metod deleteStudent, briše slog u bazi podataka.

```
public function deleteStudent($studentId){
      $result = mysql_query("delete from users where id = '$studentId'");
      $xml = "<root>";
            if ($result == true){
                  $xml .= "<result></result>";
                  $xml .= "<status>true</status>";
```

```
            }
            else {
                  $xml .= "<result>null</result>";
                  $xml .= "<status>false</status>";
            }
            $xml .= "</root>";
            echo $xml;
      }
```

Metod koja menja sadržaj u bazi zove se changeStudentiData.

```
public function changeStudentiData
($email,$password,$first_name,$last_name, $nickname,
$country,$region,$city,$department,$address,$index_num,$id){
      $result = mysql_query("UPDATE users SET email='$email',
      password='$password',first_name='$first_name',
      last_name='$last_name', nickname='$nickname',country='$country',
      region='$region',city='$city',department='$department',address='$addr
      ess',updated_at=now(), index_num='$index_num' WHERE id='$id'");
            $xml = "<root>";
            if ($result == true){
                  $xml .= "<result></result>";
                  $xml .= "<status>true</status>";
            }
            else {
            $xml .= "<result>null</result>";
                  $xml .= "<status>false</status>";
            }
            $xml .= "</root>";
            echo $xml;
      }
      }
      //van klase
      if(isset($_REQUEST['action'])){
            $action = $_REQUEST['action'];
            $student = new StudentAPIXml();

      if ($action == "getAll"){
            $start = $_REQUEST['start'];
            $requested = $_REQUEST['requested'];
            $student->getAllStudenti($start,$requested);
      }
```

```
    else if($action == "add"){
          $email = $_REQUEST['email'];
          $password = $_REQUEST['password'];
          $first_name = $_REQUEST['first_name'];
          $last_name = $_REQUEST['last_name'];
          $nickname = $_REQUEST['nickname'];
          $country = $_REQUEST['country'];
          $region = $_REQUEST['region'];
          $city = $_REQUEST['city'];
          $department = $_REQUEST['department'];
          $address = $_REQUEST['address'];
          $index_num = $_REQUEST['index_num'];
    $student->addNewStudent ($email,$password,$first_name,$last_name,
    $nickname,$country,$region,$city,$department,$address,$index_num);
    }
    else if ($action == "delete"){
          $studentId = $_REQUEST['studentId'];
          $student->deleteStudent($studentId);
    }
    else if($action == "change"){
          $email = $_REQUEST['email'];
          $password = $_REQUEST['password'];
          $first_name = $_REQUEST['first_name'];
          $last_name = $_REQUEST['last_name'];
          $nickname = $_REQUEST['nickname'];
          $country = $_REQUEST['country'];
          $region = $_REQUEST['region'];
          $city = $_REQUEST['city'];
          $department = $_REQUEST['department'];
          $address = $_REQUEST['address'];
          $index_num = $_REQUEST['index_num'];
          $id = $_REQUEST['id'];
          $student->changeStudentiData($email,$password,$first_name,
$last_name, $nickname,$country,$region,$city,$department,
$address, $index_num,$id);
    }
}
?>
```

3.7.39 Klasična veza HTML-a i PHP-a

Ova aplikacija povezuje HTML i PHP[45]. Aplikacija pripada Web 1.0 standardu, gde ima ograničen broj kontrola[4]. Kod je prilično složen, skalabilnost praktično ne postoji, odnosno ako se želi proširiti postojeći kod, onda zahteva puno napora. Ovakva aplikacija se dosta brzo izvšava za to što se nalazi kompletna na serveru.

```
<html>
<head>
<body style="background-color:#F0E68C">
<h2 style="font-family:arial colo:blue">
Unos-azuriranje podataka o studentima.
</h2>
<form method="post" action="unosstud.php">
<?php
     header('Content-Type: text/html; charset=utf-8');
Ovde se startuje sesija.
     session_start();
```

Konektovanje na MySQL, server se zove localhost, korisničko ime je root, i šifre nema, t.j. šifra je ""

```
     if(!($DB=mysql_connect("localhost","root","")))
     die("Ne moze se izvrsiti konekcija na MYSQL server");

     //"otvaranje" baze podataka
     if(!mysql_select_db("student",$DB))
     die("Ne moze se otvoriti baza podataka");
```

Naredbom if(!mysql_select_db("student",$DB)) otvara se baza "student" na serveru $DB. U narednim linijama koda vrši se deklarisanje i inicijalizacija promenjivih[68,5].

```
     $index_num="";
     $last_name="";
     $first_name="";
     $department="";
     $password="";
     $email="";
```

```
$country="";
$nickname="";
$region="";
$city="";
$created_at="";
$updated_at="";
```

Pri deklarisanju treba izbegavati korišćenje velikih slova za varijable, jer se koriste za konstante. Kod PHP-a velika slova označavaju konstante, stim što će raditi i ako se promenljive definišu velikim slovima, i ako nije čitljivo kasnije.
Ako korisnik klikne na dugme "TRAZI Prez" u aplikaciji, php funkcija *isset* proverava da li postoji globalna promenljiva $_POST, $_POST postoji tek kada se klikne na dugme submit. Dakle, promenljiva $_POST se kreira klikom na dugme submit.

```
if(isset($_POST['TRAZI']))
{
```

Narednom linijom koda, kreira se upit kojim se selektuje cela tabela users[5,6].

```
$upit="SELECT * FROM  users WHERE last_name LIKE '%".$_POST['LAST_NAME'].
"%'";
```

LIKE je sintaksa SQL jezika. LIKE znači ‘kao’[5]. Ako se koristi upit:

```
 $upit="SELECT * FROM  users WHERE last_name '".$_POST['LAST_NAME']."'";
```

onda se selektuje sve iz users-a, gde je prezime jednako sadržaju koji se upiše u polje $_POST['LAST_NAME']. Ako se napiše npr string Mar, naćiće samo prezimena koja su jednaka sa Mar. Ali ako se koristi LIKE, onda se nalaze sva prezimena koja počinju sa Mar. Ako se stavi procenat ispred i iza, $_POST promenljive: LIKE '%".$_POST['LAST_NAME']."%'";, to znači ovo što se pošalje $_POST['LAST_NAME'], može da se nađe bilo gde u prezimenu. Znači, ako postoji Mar u prizimenu, bilo gde, u početku, na kraju, negde na sredini, vratiće taj sadržaj. Ako se upiše M, vratiće sve slogove iz baze, ako prezime sadrži bilo gde slovo M. Međutim, ako se upiše samo na početku procenat: LIKE '%".$_POST['LAST_NAME']."", onda će se proslediti svi elementi koji se završavaju sa M ili MAr, zavisno šta je upisano. Dakle, ako se stavi bez procenta: LIKE '".$_POST['LAST_NAME']."", i upiše se Mar, onda se uzimaju samo prezimena koja počinju

sa Mar. Naime, LIKE znači ako se klikne na dugme, a upisano je u polje m, onda se čitaju sva prezimena koja počinju sa m. U if bloku kopira se deo tabele, ili cela tabela na osnovu upita, u RAM memoriju, a identifikator tako kreirane tabele, ili dela tabele u RAM-u se upisuje u promenljivu $rezultat. Za ovaj upit, kopira se cela tabela u RAM-u sa hard diska.

```
        if(!($rezultat=mysql_query($upit,$DB))){
              print("Ne moze se izvrsiti upit!<br/>");
              die(mysql_error());
        }
```

Narednom linijom koda kopira se prvi red iz resursa $rezultat i smešta se u promenljivu $red. Podaci iz resursa $rezultat, smeštaju se u asocijativni niz[6].

```
        if( !($red=mysql_fetch_assoc($rezultat))){
              print("Nema trazeog  usersa!br/>");
              //die(mysql_error());
        }
        else
        {
        $index_num=$red['index_num'];
              $last_name=$red['last_name'];
              $first_name=$red['first_name'];
              $department=$red['department'];
              $password=$red['password'];
              $country=$red['country'];
              $email=$red['email'];
              $nickname=$red['nickname'];
              $region=$red['region'];
              $city=$red['city'];
              $created_at=$red['created_at'];
              $updated_at=$red['updated_at'];
        }
}
```

Telo if naredbe se izvršava, ako je korisnik kliknuo na dugme"TRAZI Prez"

```
else if(isset($_POST['Trazi500']))
{
```

```
       $upit="SELECT * FROM  users order by created_at desc limit 500";
           if(!($rezultat=mysql_query($upit,$DB))){
                 print("Ne moze se izvrsiti upit!<br/>");
                 die(mysql_error());
           }
           echo ("<table border='1'>");
           echo ("<tr><th>Index</th><th>Last Name</th><th>First Name</th>
           <th>Department</th><th>Email</th><th>Country</th>

<th>Nickname</th><th>Region</th><th>City</th><th>Created_at</th><th>Update
d_at</th>
</tr>" );
while ($red=mysql_fetch_assoc($rezultat)){
echo ("<tr>");
```

Narednom linijom koda se proverava, da li je email jednak null ili prazan. Ako je prazan dodaje se n/a u tom slučaju.

```
           $em = $red['email'];
           if ($em == NULL)
                 $em = "n/a";
           echo
("<td>".$red['index_num']."</td><td>".$red['last_name']."</td><td>".$red['
first_name']."</td><td>".$red['department']."</td><td>".$em.

     "</td><td>".$red['country']."</td><td>".$red['nickname']."</td><td>".
$r   ed['region']."</td><td>".$red['city']."</td><td>".$red['created_at'].
  "</td><td>".$red['updated_at']."</td>");
                 echo ("</tr>");
           }
           echo ("</table>");
           if (isset($_SESSION['request_time'])){
                 echo ($_SESSION['request_time']."<br />");
                 echo (microtime(true)."<br />");
                 echo (microtime(true) - $_SESSION['request_time']);
           }
     }
     else if (isset($_POST['TRAZI2']))
     {
```

Naredni upit, obrađuje slučaj ako korisnik klikne na dugme "RAŽI Br.Ind."

```
$upit="SELECT * FROM  users WHERE index_num LIKE
'".$_POST['index_num']."'";
          if(!($rezultat2=mysql_query($upit2,$DB))) {
                print("ne moze se izvrsiti upit!<br/>");
                die(mysql_error());
          }
```

Varijabla $red predstavlja naziv asocijativnog niza. Funkcija mysql_fetch_assoc uzima jedan red iz RAM-a[5,6], gde pokazuje $rezultat2. Ovaj red se pretvara u niz, tipa asocijativni niz. Polja niza $red su nazivi kolona u tabeli a vrednosti su vrednosti iz tabele, t.j. datog reda.

```
          if(!($red=mysql_fetch_assoc($rezultat2))){
                print("Nema trazenog  usersa! <br/>");
          }
          else
          {
                $index_num=$red['index_num'];
                $last_name=$red['last_name'];
                $first_name=$red['first_name'];
                $department=$red['department'];
                $password=$red['password'];
                $country=$red['country'];
                $email=$red['email'];
                $nickname=$red['nickname'];
                $region=$red['region'];
                $city=$red['city'];
                $created_at=$red['created_at'];
                $updated_at=$red['updated_at'];
          }
     }
```

Naredni blok koda se aktivira, ako korisnik klikne na dugme "DODAJ"

```
else if (isset($_POST['DODAJ']))
{
if ((!$_POST['LAST_NAME']) || (!$_POST['FIRST_NAME']) ||
(!$_POST['DEPARTMENT']) || (!$_POST['INDEX_NUM']) || (!$_POST['PASSWORD'])
```

```
|| (!$_POST['EMAIL']) || (!$_POST['COUNTRY']) || (!$_POST['NICKNAME']) ||
(!$_POST['REGION']) || (!$_POST['CITY']))
{
```
echo"Mora biti uneto prezime,ime,broj indeksa, department, sifra i region!";
```
     }
     else if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
     echo "This email address is considered valid.";
           }
           else
           {
                 //$created_at=date("d.m.Y",time());
                 //$updated_at=date("d.m.Y",time());

                 //
```
Metod mysql_real_escape_string sprečava sql inject u DB, uklanja html specijalne karaktere[6].

```
$email = mysql_real_escape_string(htmlspecialchars($_POST['EMAIL']));
$upitdod="INSERT INTO
users(last_name,first_name,index_num,department,password,email,country,
nickname,region,city,created_at,updated_at)
VALUES('".$_POST['LAST_NAME']."' ,'".$_POST['FIRST_NAME']."',
'".$_POST['INDEX_NUM']."','". $_POST['DEPARTMENT']."',
'".$_POST['PASSWORD']. "','".$_POST['EMAIL']."','"
.$_POST['COUNTRY']."','".$_POST['NICKNAME']."','". $_POST['REGION'].
"','".$_POST['CITY']."',now(),now())";
                 if(!($rezultatd= mysql_query($upitdod,$DB))){
                       print(" Ne moze se izvrsiti upit! < br/>");
                       die( mysql_error());
                 }
                 $MESSAGE="Slog dodat";
           }
     }
     else if (isset($_POST['AZURIRAJ']))
     {

            $updated_at=date("y-d-m H:m:s",time());
//ako je korisnik kliknuo na dugme " AZURIRAJ"
if ((!$_POST['LAST_NAME']) || (!$_POST['FIRST_NAME']) ||
(!$_POST['DEPARTMENT']) || (!$_POST['INDEX_NUM']) || (!$_POST['PASSWORD'])
```

```
|| (!$_POST['COUNTRY']) || (!$_POST['NICKNAME']) || (!$_POST['REGION']) ||
(!$_POST['CITY']) )
{
     echo "Mora biti uneto prezime,ime,broj indeksa, status i sifra!";
}
else if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
     echo "This email address is considered valid.";
     }
     else
     {
```

Funkcija, mysql_real_escape_string($value), ubacuje kosu crtu ispred specijalnog znaka, u tekstu[4].

```
$email = mysql_real_escape_string(htmlspecialchars($_POST['EMAIL']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['LAST_NAME']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['FIRST_NAME']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['PASSWORD']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['DEPARTMENT']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['COUNTRY']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['NICKNAME']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['CITY']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['UPDATED_AT']));
$email = mysql_real_escape_string(htmlspecialchars($_POST['INDEX_NUM']));
$upitaz="UPDATE  users SET last_name='".$_POST['LAST_NAME']."',
first_name='".$_POST['FIRST_NAME']."', password='".$_POST['PASSWORD'].
"'department='" .$_POST['DEPARTMENT']."',country='".$_POST['COUNTRY'].
"',nickname='".$_POST['NICKNAME']."',region='".$_POST['REGION']."',city='"
.$_POST['CITY']."',updated_at = now(),email='".$_POST['EMAIL']."' WHERE
broj_indeksa='".$_POST['INDEX_NUM']."'";

     if(!($rezultat=mysql_query($upitaz,$DB)))
       {
           print("Ne moze se izvrsiti AZURIRANJE u tabeli  users!<br/>");
           die(mysql_error());
        }
$MESSAGE="SLOG AZURIRAN";
    }
           $index_num=$red['index_num'];
           $last_name=$red['last_name'];
           $first_name=$red['first_name'];
```

```
        $department=$red['department'];
        $password=$red['password'];
        $country=$red['country'];
        $email=$red['email'];
        $nickname=$red['nickname'];
        $region=$red['region'];
        $city=$red['city'];
        $created_at=$red['created_at'];
        $updated_at=$red['updated_at'];
 }
else if(isset($_POST['OBRISI']))
{
```

Ovde se onemugućava brisanje, kada u polje INDEX_NUM nije uneto ništa ili je uneta nula. Kako u bazi puno polja nema INDEX_NUM, već je prazno, onda bi se obrisala gotovo cela baza. Dakle, ovde se proverava da li je prazan ili je nula, ne dozvoljava se takav slučaj.

```
if ($_POST['INDEX_NUM'] != "" || $_POST['INDEX_NUM'] != "0")
{
//ako je korisnik kliknuo na dugme"OBRISI"
$upitbris="DELETE FROM  users WHERE index_num= '".
      $_POST['INDEX_NUM']."'";
       if(!($rezultatbris=mysql_query($upitbris,$DB)))
       {
            print("Ne moze se izvrsiti brisanje!<br/>");
            die(mysql_error());
       }
```

Naredni deo koda briše selektovane podatke sa ekrana, za slog koji je obrisan.

```
        $index_num="";
        $last_name="";
        $first_name="";
        $department="";
        $password="";
        $country="";
        $email="";
        $nickname="";
        $region="";
        $city="";
        $created_at="";
        $updated_at="";
```

```
            $MESSAGE="SLOG OBRISAN";
     }
}
```

PHP metod trim vrši brisanje belih znakova sa leve i desne strane stringa.

```
     $last_name=trim($last_name);
     $first_name=trim($first_name);
     $password=trim($password);
     $department=trim($department);
     $index_num=trim($index_num);
     $country=trim($country);
     $email=trim($email);
     $nickname=trim($nickname);
     $region=trim($region);
     $city=trim($city);
     $created_at=trim($created_at);
     $updated_at=trim($updated_at);
```

Kontrole se kreiraju u HTML-u. Ako se ne stavi kolika je dužina polja u input name kontroli, u osobini size="5", onda se dužina definiše po difoltu.

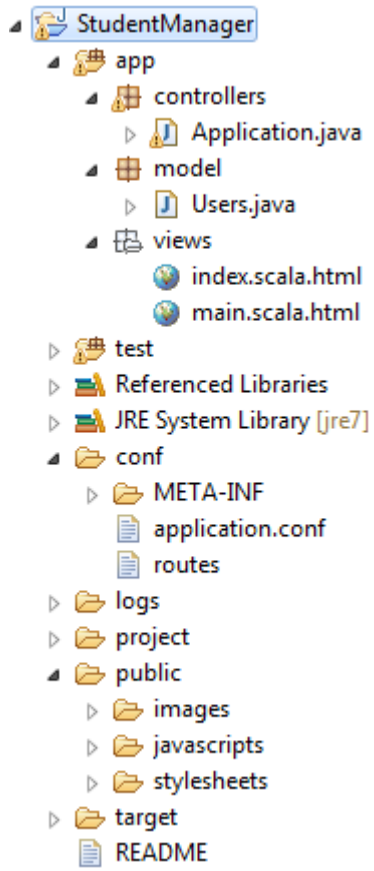
```
?>
<table>
<colgroup>
           <col span="3" style="color:#F00">
</colgroup>
     <tr>
     <td align="right" style="color:#F00">Br.indeksa:</td>
<td><input name="INDEX_NUM" type="text" placeholder="number of index"
value="<?php echo $index_num?>"/></td>

     <td align="right" style="color:#F00">Prezime:</td>
     <td><input name="LAST_NAME" type="text" placeholder="last name"
        value="<?php echo $last_name?>"/></td>

     <td align="right" style="color:#F00">Ime:</td>
     <td><input name="FIRST_NAME" type="text" placeholder="first name"
         value="<?php echo $first_name?>"/></td>
</tr>
<tr>
     <td align="right">Password:</td>
```

```
     <td><input name="PASSWORD" type="password" placeholder="password"
      value="<?php echo $password?>"/></td>
<td align="right">Department:</td>
     <td><input name="DEPARTMENT" type="text" placeholder="department"
       value="<?php echo $department?>"/></td>
<td align="right">Email:</td>
     <td><input name="EMAIL" type="text" placeholder="email" value="<?php
       echo $email?>" /></td>
</tr>
<tr>
<td align="right">Country:</td>
<td><input name="COUNTRY" type="text" placeholder="country" value="<?php
  echo $country?>"/></td>
<td align="right">Nickname:</td>
<td><input name="NICKNAME" type="text" placeholder="nickname"
   value="<?php echo $nickname?>"/></td>
<td align="right">Region:</td>
<td><input name="REGION" type="text" placeholder="region" value="<?php
echo
   $region?>"/></td>
</tr>
<tr>
<td align="right">City:</td>
<td><input name="CITY" type="text" placeholder="city"  value="<?php echo
$city?>"/></td>
<td align="right">Created_at:</td>
<td><input name="CREATED_AT" type="text" placeholder="created at"
value="<?php echo $created_at?>"/></td>
<td align="right">Updated_at:</td>
<td><input name="UPDATED_AT" type="text" placeholder="created up"
value="<?php echo $updated_at?>"/></td>
</tr>
</table>
     <br>
     <input type="submit" name="DODAJ" value="DODAJ" style="background-
color:blue; color:yellow;font-weight:bold"/>
<input type="submit" name="AZURIRAJ" value="A&#381;URIRAJ"
style="background-color:blue; color:yellow;font-weight:bold"/>
<input type="submit" name="OBRISI" value="OBRISI" style="background-
color:blue; color:yellow;font-weight:bold"/>
<br><br>
```

```
<input type="submit" name="TRAZI" value="TRA&#381;I Prez."
style="background-color:#666; color:#FFF;font-weight:bold"/>
<input type="submit" name="TRAZI2" value="TRA&#381;I BR.Ind."
style="background-color:blue; color:yellow;font-weight:bold"/>
<input type="submit" name="Trazi500" value="Trazi500" style="background-
color:blue; color:yellow;font-weight:bold"
onclick="<?php  $_SESSION['request_time'] = microtime(true);  ?>"/>
     <br>
<?php
     if (isset($MESSAGE))
     {
           echo "<BR><BR>$MESSAGE";
     }
?>
</form>
</body>
</head>
</html>
```

3.7.40 Java server

Kada se rade Java servisi, postoje dva načina. Prvi način je da se pišu čisti servleti. Servleti su Java klase koje omogućavaju rad sa HTTP requestom. HTTP request sadrži request i response(zahtev i odgovor). Drugi način je da se primeni jedan od frejmova, koji u sebi ima ugrađene servlete. Najpoznatiji je Spring. U radu se koristi najnoviji Framework play. Play ima ugrađene klase na bazi servleta. Struktura foldera i fajlova na strani Java servera, koji vrše serijalizaciju i deserijalizaciju podataka u XML i JSON formtu, je data na slici 3.7.108. U app se nalazi aplikacija, gde su smešteni svi fajlovi aplikacije, na serverskoj strani. Pre početka rada, mora se uraditi konfiguracija aplikacije. Kada se otvori projekat, ima tri fajla za konfiguraciju *build.properties, build.scale i plugins.sbt.*



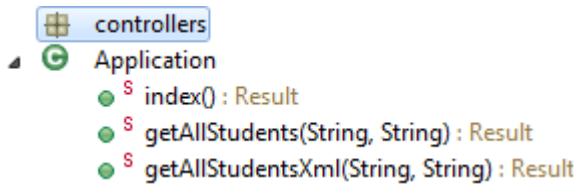
Slika 3.7.108. Struktura foldera i fajlova na strani Java servera

3.7.41 Xml u Javi

Na severskom delu postoje dve Java klase, koje komuniciraju sa klijentskom aplikacijom, Aplication.java i Users.java. Struktura foldera i fajlova odgovara MVC strukturi. Klasa Users.java opisuje tabelu u bazi podataka users.

3.7.42 Klasa Aplication.java

Ovo je glavna aplikacija na serverskom delu, sa kojom klijentska aplikacija komunicira, obraća se direktno metodama iz ove klase. Ova klasa sadrži tri statičke javne metode index(), getAllStudents i getAllStudentsXml. Struktura klase je prikazana na slici 3.7.109.



Slika 3.7.109. Prikaz osobina i metoda za klasu Application

3.7.43 Fajl routs

U ovom fajlu se definiše metod i adrese do fajlova na serveru. Struktura fajla je:

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~
# Home page
GET     / controllers.Application.index()
GET     /students-all/:start/:requested
controllers.Application.getAllStudents(start:String,requested:String)
GET     /students-all-xml/:start/:requested
controllers.Application.getAllStudentsXml(start:String,requested:String)

# Map static resources from the /public folder to the /assets URL path
GET     /assets/*file  controllers.Assets.at(path="/public", file)
```

Ovde se definiše interfejs. Naime, definiše se metod koji se koristi za transport podataka od klijenta do servera, ovde je GET metod. Sa students-all imenuje se adresa do metode getAllStudents. Dakle, students-all-xml je javni pristup metodu getAllStudentsXml, odnosno, students-all-xml predstavlja url adresu do metode getAllStudentsXml. localhost:9000 je aplikacija, tu se nalazi url adresa.

Desni deo u liniji koda,

```
controllers.Application.getAllStudentsXml(start:String,requested:String)
```

označava funkciju u kontroleru. Levi deo /students-all-xml/:start/:requested je public dela controllers.Application.getAllStudentsXml(start:String,requested:String) ili public adresa do funkcije getAllStudentsXml. Korisnik daje ime intefejsu, ovde smo nazvali smo students-all-xml, t.j. /students-all-xml/:start/:requested. Kada se stave dve tačke u izrazu /students-all-xml/:start/:requested, onda se taj deo odnosi na promenljivu iz funkcije start:String i requested:String. Naime, :start se odnosi na promenljivu start u metodi getAllStudentsXml kroz(/) dve tačke t.j :requested promenljiva requested (requested:String). To znači da se šalje ovakav upit: http:localhost:9000/students-all/0/5, ovde je start nula, a requested je 5, traži da se vrati 5 rezultata.

Ispis: http:localhost:9000/students-all-xml/0/5 je praktično isti kao ispis: http:localhost/studenti_munir/services/xml/StudentAPIXml.php?action=getAll&start=0&requested=5, samo u drugom jeziku i drugi je interfejs.

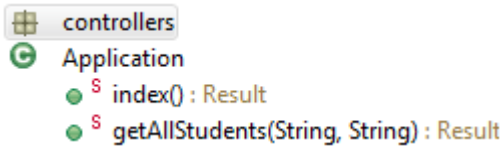
U adresi http:localhost:9000/students-all-xml/0/5 port za Java server je broj 9000, ovaj fajl se nalazi u virtuelnoj mašini. Kada se pokrene aplikacija, adresa students-all-xml referencira na funkciju getAllStudentsXml.Takođe je neophodno navesti naredni deo koda:

```
GET /assets/*file  controllers.Assets.at(path="/public", file)
```

U ovoj liniji koda, assets su potrepštine, html fajlovi, slike. Folderu public pristupa se putem assets-a. Ako su potrebne slike, dokumenati, snimaju se u folder public, a onda im se pristupa kroz assets

3.7.44 Json u Javi

Glavna aplikacija na Java serveru koja komunicira sa klijentskom aplikacijom nosi naziv Aplication.java. Klasa Aplication.java sadrži dve metode index i getAllStudents. Struktura ove klase je data na slici3.7.110.



Slika 3.7.110. Prikaz osobina i metoda za klasu Application

Algoritam metoda u Java klasi Aplication dat je u narednom kodu:

```
package controllers;
public class Application extends Controller{
    public static Result index(){
        return ok(index.render("Your new application is ready."));
    }
```

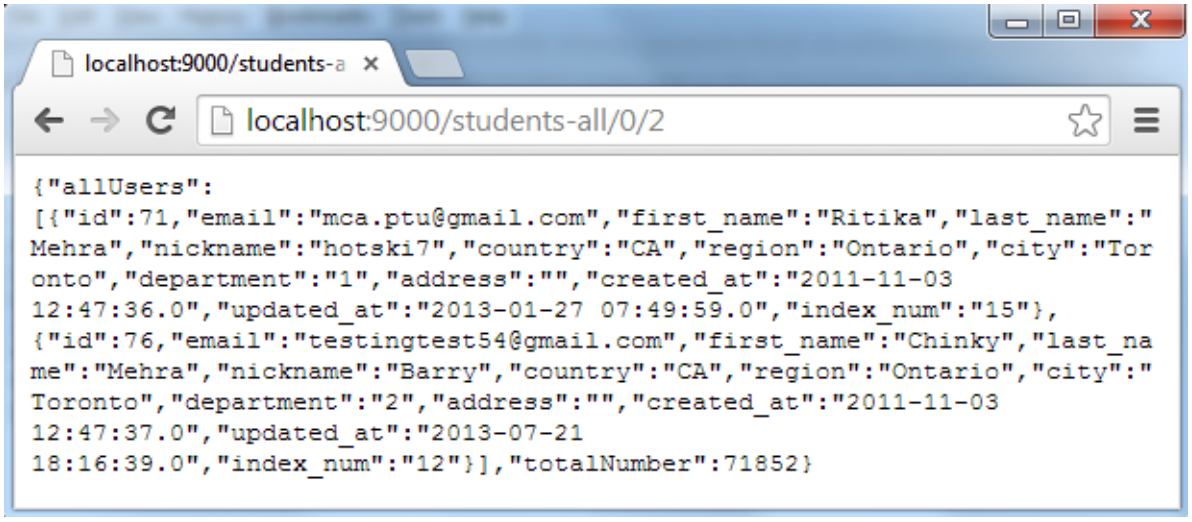
Metoda getAllStudents se poziva sa klijentske strane, pomoću koje se učitavaju podaci iz baze podataka. Ovoj metodi je neophodno proslediti vrednosti za dva parametra start i requsted. Prva vrednost se odnosi na početak učitavanja u bazi podataka, a druga vrednost definiše koliko podataka treba učitati iz baze podataka. Naredba @Transactional ispred medode, ukazuje da se ovde vrši tranzicija. Json document se sastoji od više elemenata ili nodova. U prvoj liniji koda unutar metode, deklariše se objekat result tipa ObjectNode. Instanca result se kreira statičkom metodom newObject, koja pripada klasi Json. U drugoj liniji koda deklariše se niz nodova arrayUsers, koji u sebi smešta jedan objekat result, koji poseduje jedno polje koje nosi naziv allUsers. U telu metode se deklariše lista users, čiji su elementi objekti tipa Users. Lista users se popunjava, pozivom statičke metode getAllStudents, klase Users. Statičkoj metodi

getAllStudents, klase Users, se prosleđuju dva cela broja. For petljom se vrši iteracija kroz sve elemente niza users. Dakle, svaki element pojedinačno se kopira iz niza users u promenljivu user tipa Users, unutar for petlje, onda se kreira lokalni objekat row tipa ObjectNode. Naredbom row.put("id", user.id); u kreirani object row, kreira se polje id i u njega se upisuje vrednost id t.j. vrednost iz baze podataka user.id. Na isti način se generišu druge osobine u dinamičkom objektu row email, first_name, last_name, nickname, country, region, city, department, address, created_at, updated_at i index_num i popunjavaju vrednostima respektivno user.email, user.first_name, user.last_name, user.nickname, user.country, user.region, user.city, user.department, user.address, user.created_at.toString(), user.updated_at.toString(), user.index_num. Kada se kreiraju sva polja i popune vrednostima u privremenu instancu row, onda se objekat row, koristi za popunjavanje elementa niza, naredbom arrayUsers.add(row);

```
    @Transactional
    public static Result getAllStudents(String start,String requsted){
      ObjectNode result = Json.newObject();
      ArrayNode arrayUsers = result.putArray("allUsers");

List<Users> users =
Users.getAllStudents(Integer.parseInt(start),Integer.parseInt(requsted));
      if (users != null){
            for (Users user:users){
                  ObjectNode row = Json.newObject();
                  row.put("id", user.id);
                  row.put("email", user.email);
                  row.put("first_name", user.first_name);
                  row.put("last_name", user.last_name);
                  row.put("nickname", user.nickname);
                  row.put("country", user.country);
                  row.put("region", user.region);
                  row.put("city", user.city);
                  row.put("department", user.department);
                  row.put("address", user.address);
                  row.put("created_at", user.created_at.toString());
                  row.put("updated_at", user.updated_at.toString());
                  row.put("index_num", user.index_num);
                  arrayUsers.add(row);
            }
      }
```

```
        result.put("totalNumber", Users.getCountUsers());
        return ok(result);
    }
}
```

Izlaz u Internet pretraživač, kada se upiše url adresa, ima oblik:



Slika 3.7.111. JSON fajl

Kada se popuni niz arrayUsers, onda se ubacuje još jedan objekat u resul, koji sadrži ukupan broj slogova u bazi, naredbom result.put("totalNumber", Users.getCountUsers());

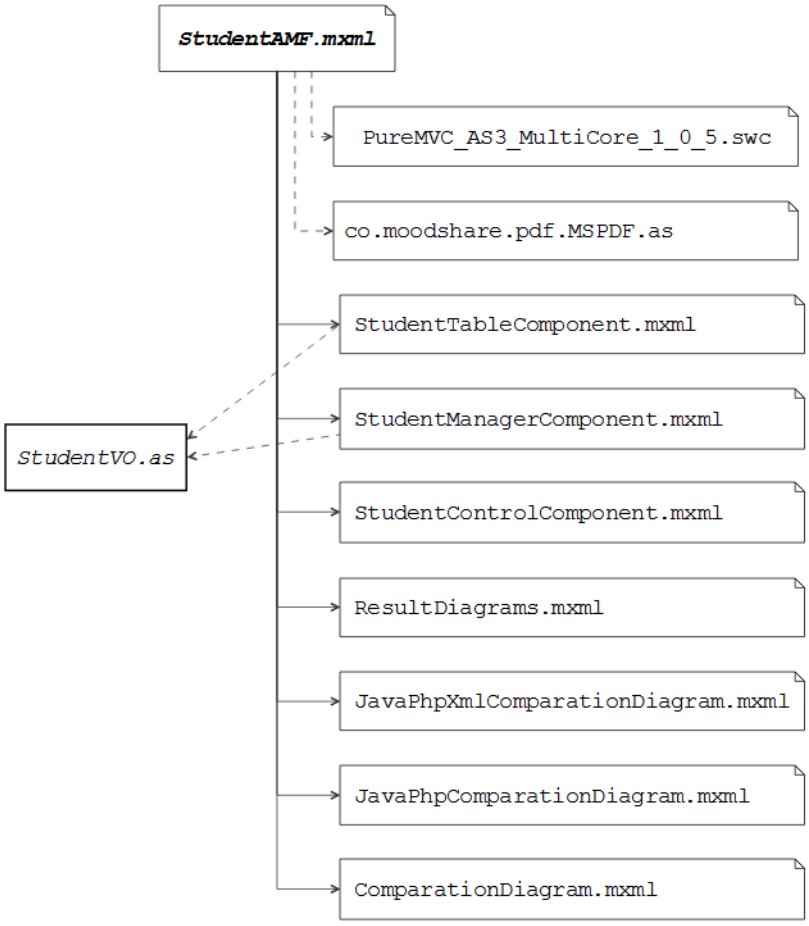
3.7.45 Modelovanje radne aplikacije u UML-u.

Cela aplikacija je objektno zasnovana, pa je stoga prirodno da se uml primeni kao jezik za modelovanje aplikacije. U radu se koristi dijagram klasa, objektni dijagram i dijagram komponenti. Dijagram klasa, se najčešće koristi u uml-u. Ovaj dijagram se sastoji od klasa, interfejsa, njihovih veza i saradnje, u osnovi predstavlja objektno orijentisani pogled sistema. Klasa može predstaviti konkurentnost.

3.7.46 Komponentni dijagram

Komponentni dijagram opisuje skup komponenti i njihove veze. Komponente se sastoje od klasa, interfejsa i veza. Ovaj dijagram predstavlja implementaciju vizuelnog dela. Tokom faze izrade softvera (klasa, interfejsa itd), sistem je organizovan u različite grupe, u zavisnosti od njihovih veza. Te grupe su poznate kao komponente. Ovaj dijagram uključuje sve fajlove

koji se koriste u aplikaciji, i druge bitne elemente, biblioteke kao i veze između ovih komponenti. Glavna komponenta u aplikaciji je StudentAMF.mxml

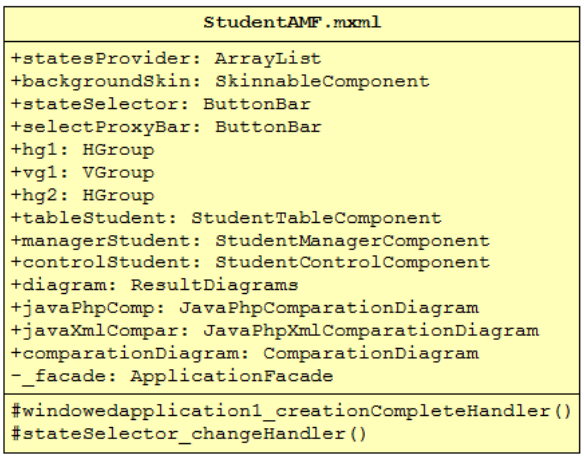


Slika 3.7.112. Komponentni dijagram autorske aplikacije

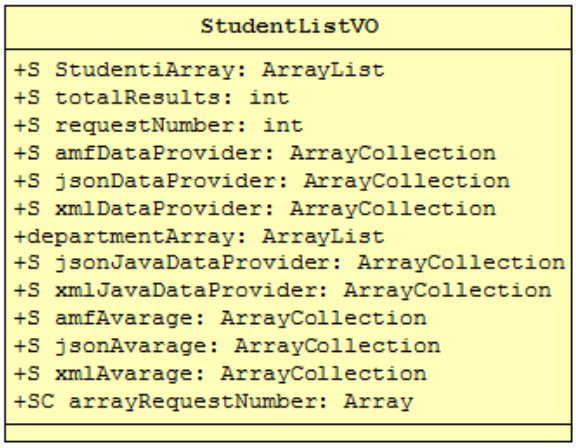
3.7.47 Dijagram klasa

Sve klase u aplikaciji, su predstavljene u dijagramu klasa, pri čemu ime svake klase govori šta klasa radi. Atributi i metode svake klase su jasno definisane. Za svaku klasu definisan je potreban broj osobina, zato što nepotrebne osobine čini dijagram komplikovanim.

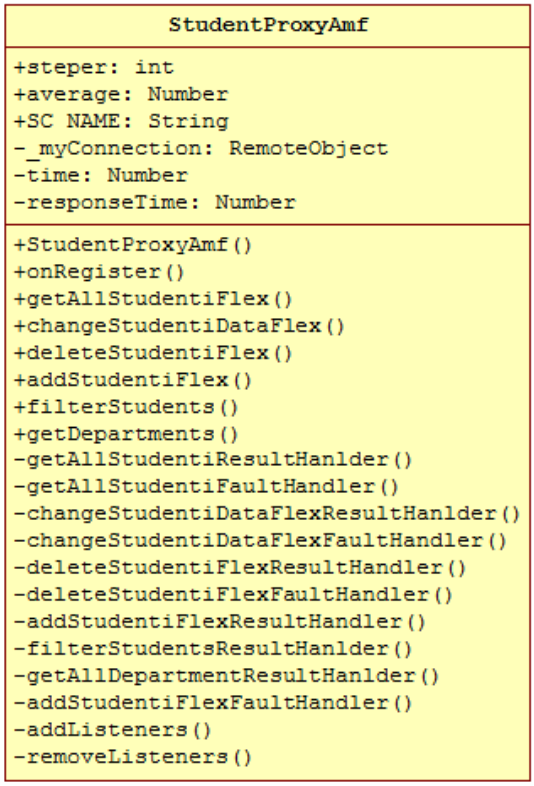
Klasa *StudentAMF.mxml*



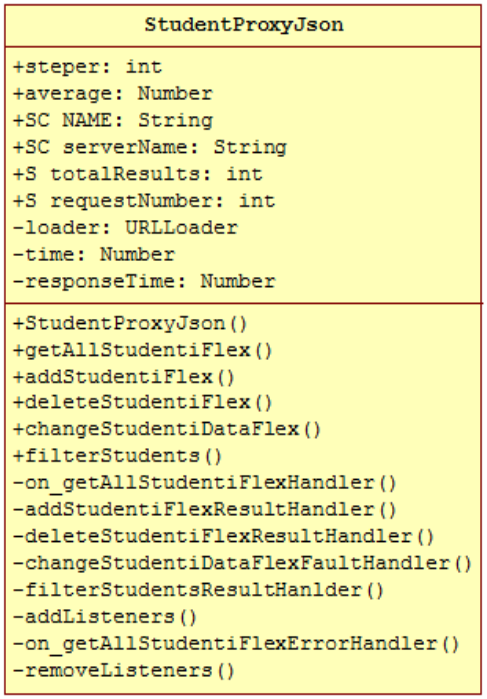
Klasa *StudentListVO*



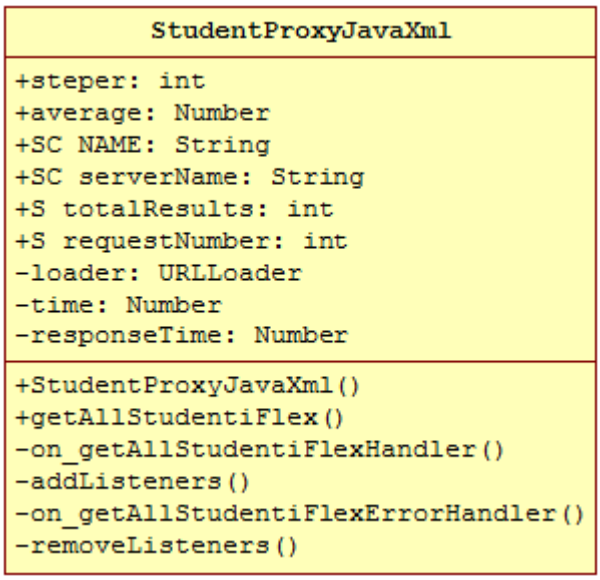
Klasa *StudentProxyAmf*



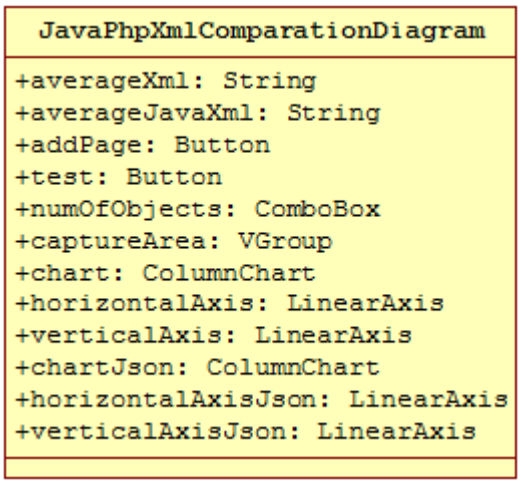
Klasa *StudentProxyJson*



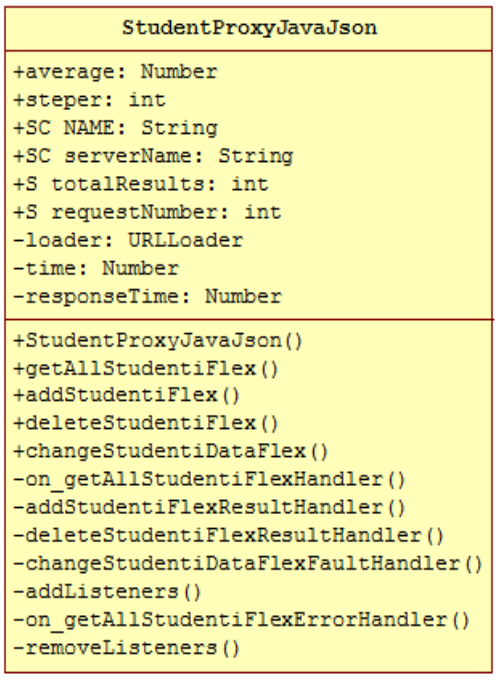
Klasa*StudentProxyJavaXml*



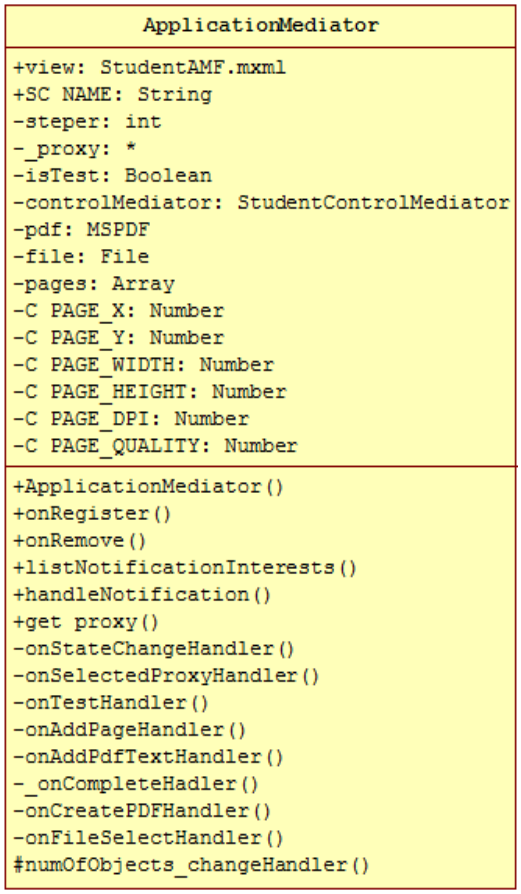
Klasa *JavaPhpXmlComparationDiagram*



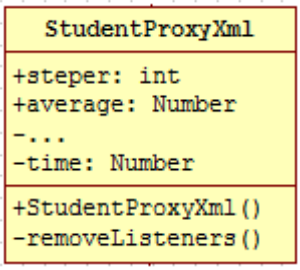
Klasa *StudentProxyJavaJson*



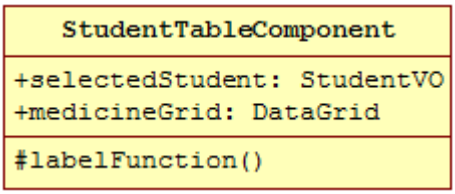
Klasa *ApplicationMediator*



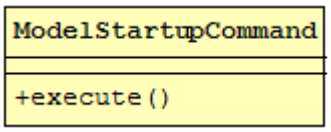
StudentProxyXml



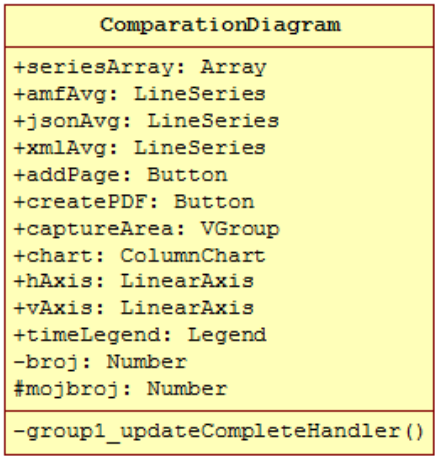
Klasa *StudentTableComponent*



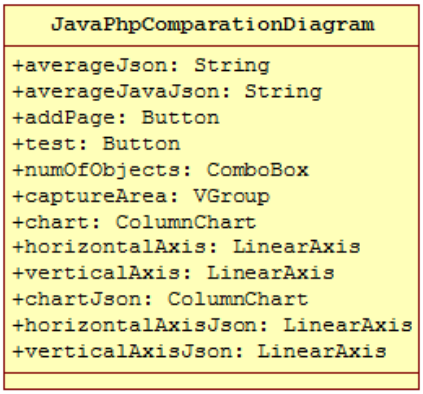
Klasa *ModelStartupCommand*



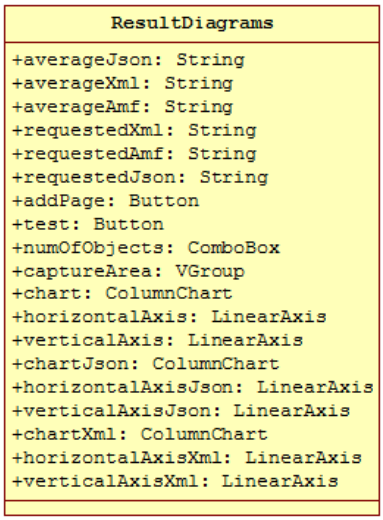
Klasa *ComparationDiagram*



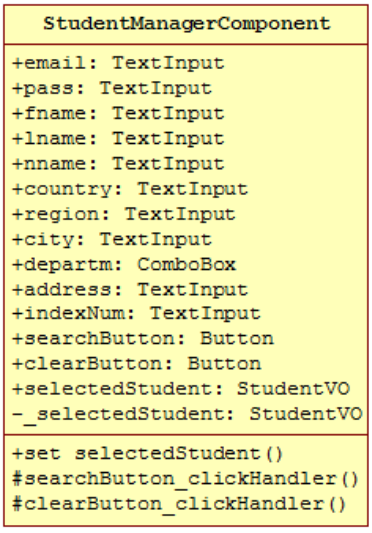
Klasa *JavaPhpComparationDiagram*



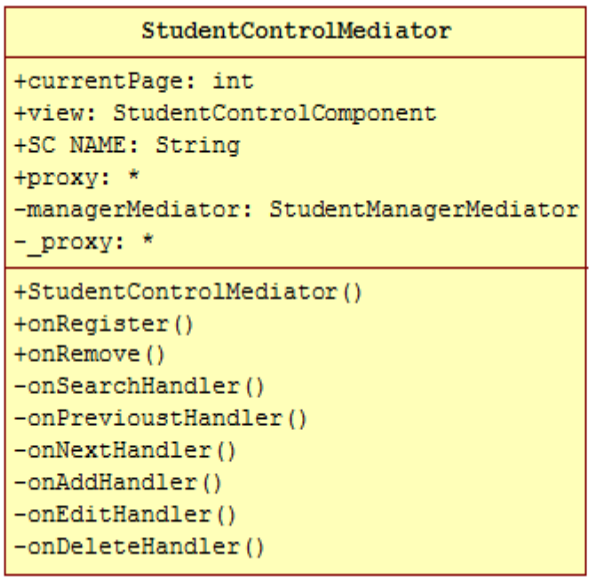
Klasa *ResultDiagrams*



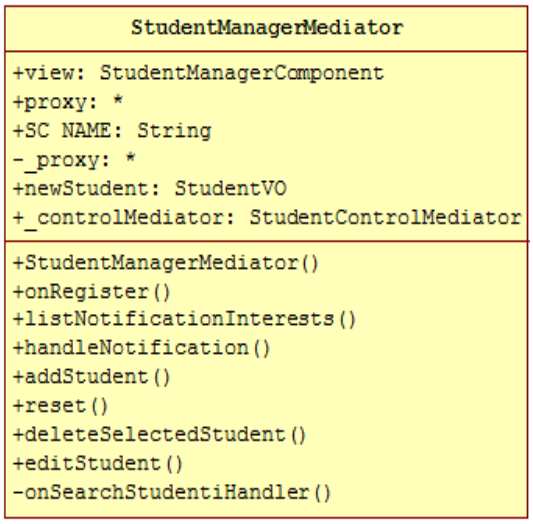
Klasa *StudentManagerComponent*



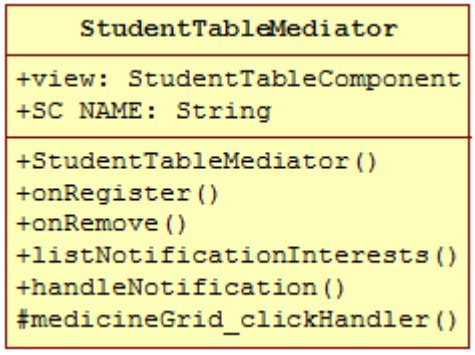
Klasa *StudentControlMediator*



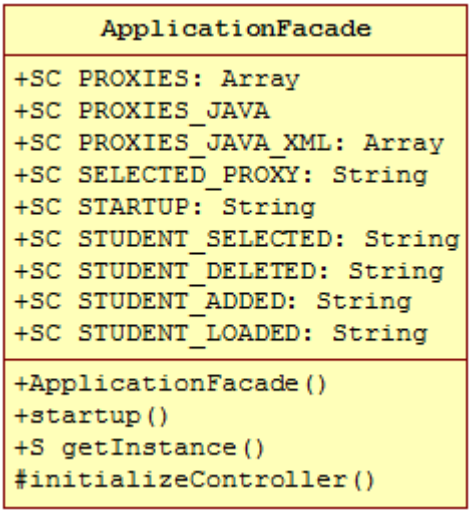
Klasa *StudentManagerMediator*



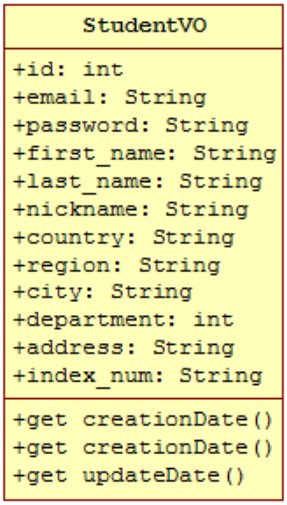
Klasa *StudentTableMediator*



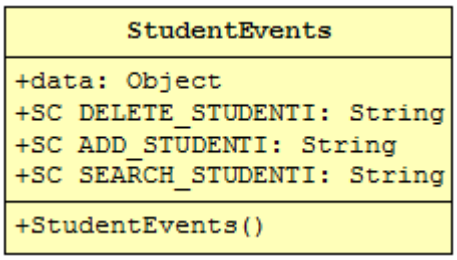
Klasa *ApplicationFacade*



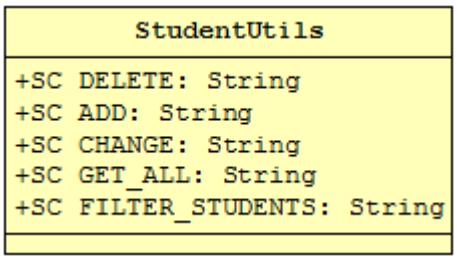
Klasa *StudentVO*



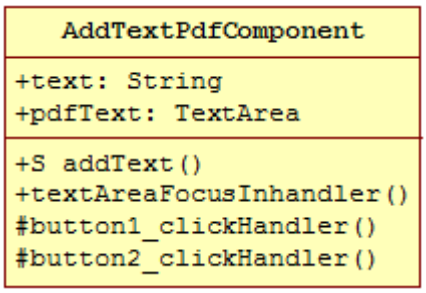
Klasa *StudentEvents*



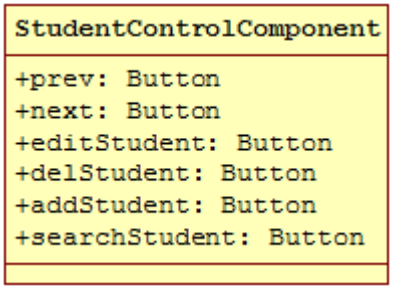
Klasa *StudentUtils*



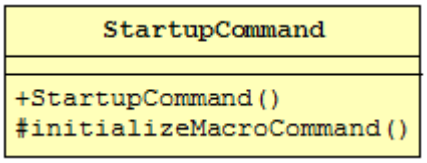
Klasa *AddTextPdfComponent*



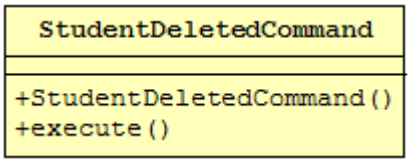
Klasa *StudentControlComponent*



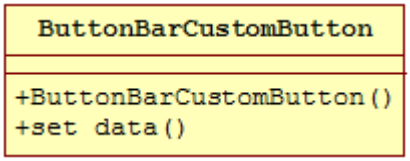
Klasa *StartupCommand*



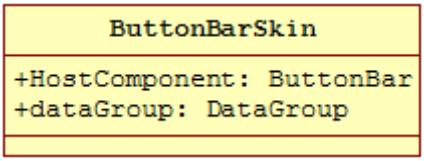
Klasa *StudentDeletedCommand*



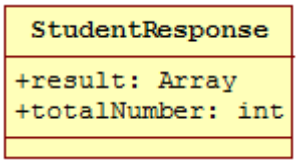
Klasa *ButtonBarCustomButton*



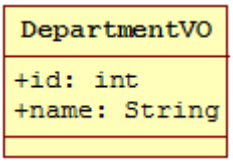
Klasa *ButtonBarSkin*



Klasa *StudentResponse*

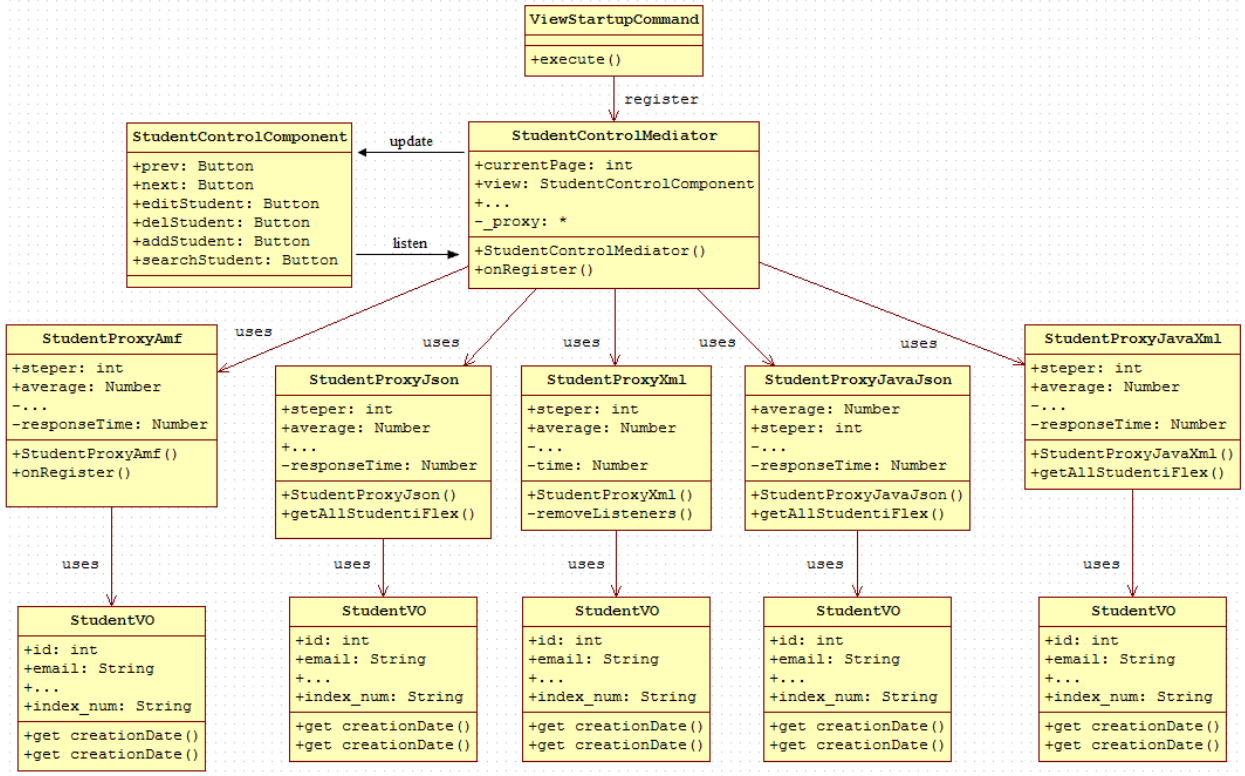


Klasa *DepartmentVO*

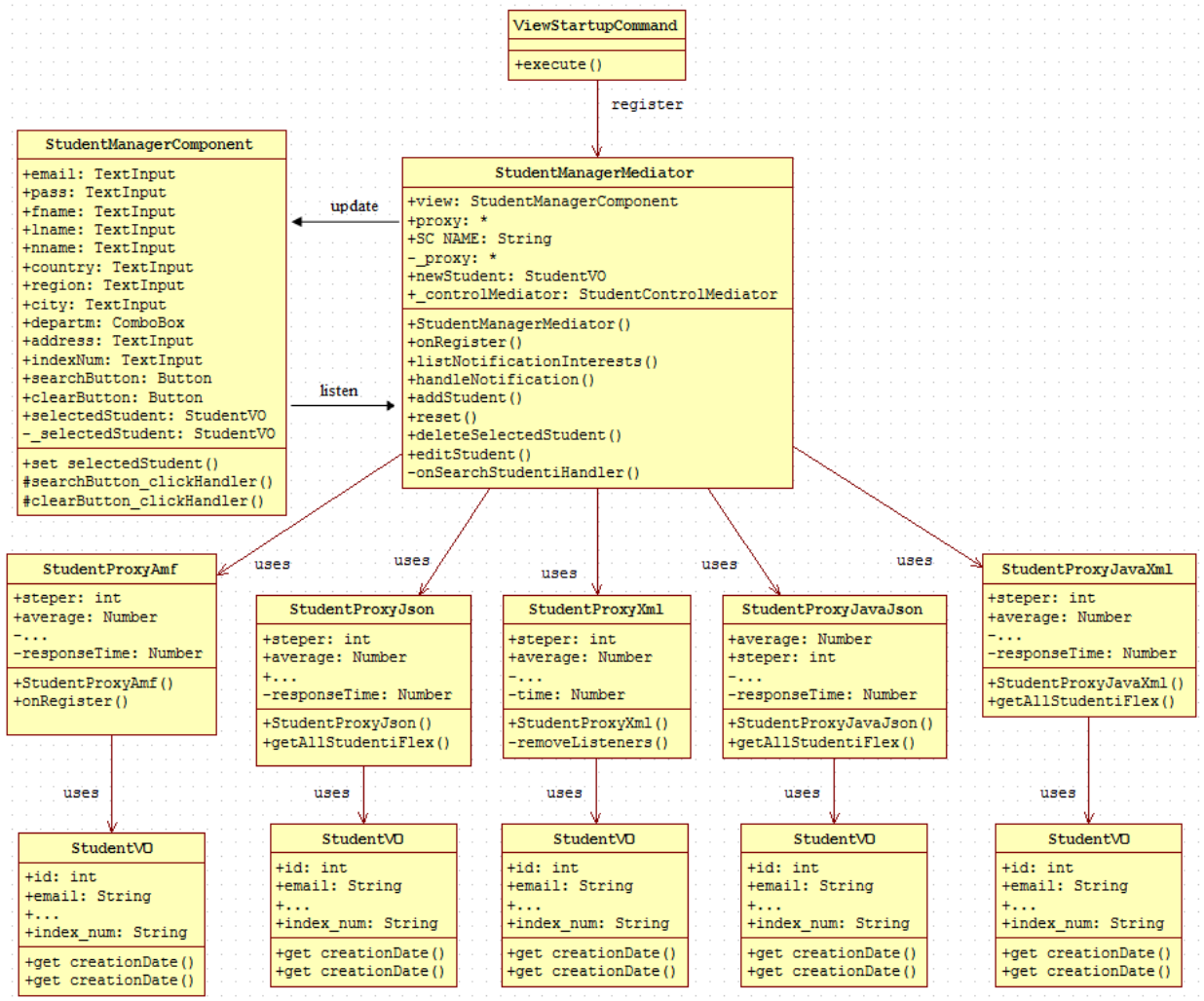


3.7.48 Objektni dijagram

Objektni dijagram predstavlja instancu klasnog dijagrama, opisuje statičko stanje sistema, koje predstavlja jedan snapshot u konkretnom trenutku. Na slici 3.7.113, prikazan je objektni dijagram, gde se komandom ViewStartupCommand registruje kontroler StudentControlMediator. Ovaj kontroler je pridružen grafičkoj komponenti StudentControlComponent, koju apdejtuje, a grafička komponenta osluškuje metode kontrolera. Kontroler StudentControlMediator koristi proksije StudentProxyAmf, StudentProxyJson, StudentProxyXml, StudentProxyJavaJson i StudentProxyJavaXml za pripremu podataka. Svi proksiji koriste instance klase StudentVO.



Slika 3.7.113. Objektni dijagram autorske aplikacije, za objekte klase ViewStartupCommand, StudentControlMediator, StudentControlComponenti proksije



Slika 3.7.114 Objektni dijagram autorske aplikacije, za objekte klase ViewStartupCommand, StudentManagerMediator, StudentManagerComponent i proksije

4. EKSPERIMENTALNO ISTRAŽIVANJE

U ovom poglavlju su analizirane web, web2.0, iznad web2.0[4,8,52], desktop aplikacije u pogledu složenosti pisanja koda, efikasnosti, bezbednosti, korisničkog interfejsa.

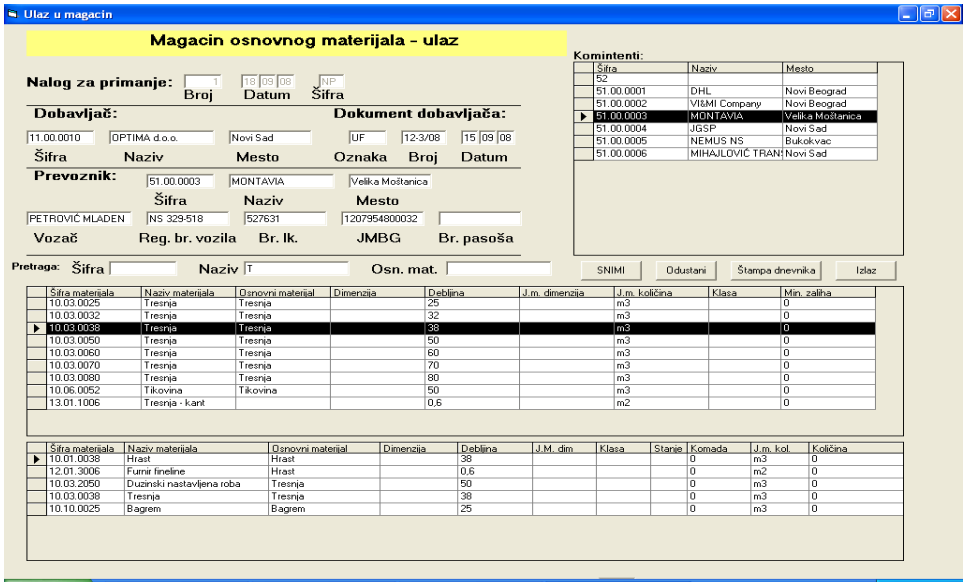
4.1 Rezultati eksperimentalnog istraživanja

U cilju provere koliko Web aplikacije mogu podržati karakteristike kompleksnog korisničkog grafičkog interfejsa svojstvenog desktop aplikacijama, analizirana je jedna tipična poslovna desktop aplikacija-softverski paket za praćenje magacinskog poslovanja, i to konkretno ulaz materijala u magacin, i odgovarajuće verzije ove aplikacije urađene kao klasična web i web2.0 aplikacija. Poređenjem osobina ovih aplikacija, moguće je utvrditi koje su teškoće u izradi korisničkog interfejsa web zasnovanih poslovnih aplikacija.

U primeru su korištene tehnologije otvorenog koda za izradu web i web2.0 aplikacija: -Javascript, PHP, MySql, i Adobe Flex, ali dobijeni rezultati važe i za aplikacije urađene i primenom bilo kojih drugih tehnologija za razvoj web, odnosno web2.0 aplikacija.

4.1.1 Desktop aplikacija

Izgled korisničkog interfejsa ove aplikacije, urađene u programskom jeziku Visual Basic 6 (Microsoft Visual Studio 6), prikazan je na slici 4.1.1:



Slika 4.1.1: Izgled tipičnog korisničkog grafičkog interfejsa desktop aplikacije

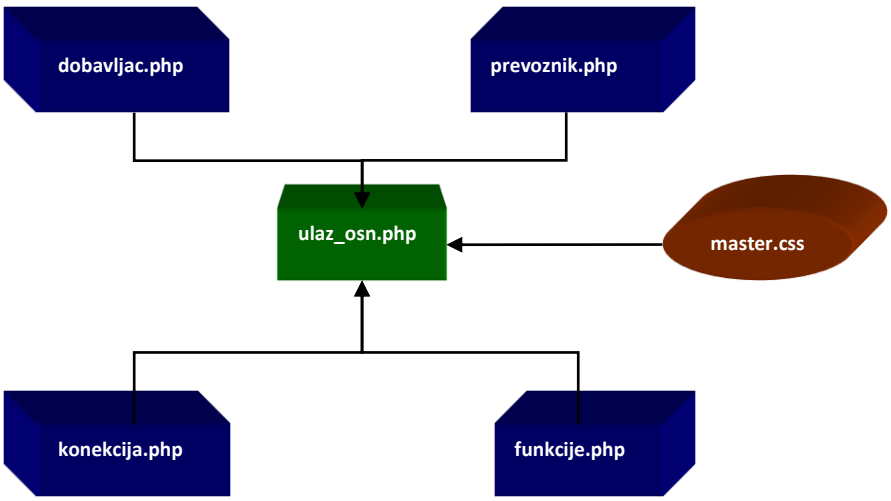
Aplikacija poseduje sve standardne mogućnosti korisničkog interfejsa, tako da ovde neće biti dalje detaljno opisana.

4.1.2 Web aplikacija

Za realizaciju Web aplikacije za magacinsko poslovanje, korišten je skriptni jezik na strani servera, konkretno PHP 5.2.0. Baza podataka je preuzeta iz postojeće desktop aplikacije, i ona je realizovana kao MySql baza (u Visual Basicu korišten je ODBC drajver, a u Web aplikaciji PHP-ov ugrađen API).

Struktura aplikacije prikazana je na slici 4.1.2. Web verzija aplikacije se sastoji od šest fajlova u radnom folderu na serveru. Jezgro aplikacije je ujedno i indeks stranica aplikacije pod imenom ***ulaz_osn.php***. Fajlovi ***dobavljac.php*** i ***prevoznik.php*** su moduli aplikacije, ***funkcije.php*** i ***konekcija.php*** su klase a ***master.css*** je CSS (CascadingStyleSheet) fajl koji je zadužen za grafički izgled stranice. Ovih šest fajlova su deo aplikacije koji služi za prijem tj. ulaz materijala u magacin.

Interfejs aplikacije je napravljen u HTML jeziku uz razmeštanje elemenata na ekranu pomoću CSS-a. Radi se o određenom broju input polja i tabela iz razloga smeštanja velikog broja informacija u okviru jednog ekrana.



Slika 4.1.2: Struktura Web aplikacije "Ulaz u magacin"

JavaScript funkcije korištene su za osvežavanje stranice aplikacije kako bi se potrebni podaci upisali na server. Naime, pošto PHP programski jezik izvršava se na serveru, potrebno je

osvežiti stranicu svaki put kada želite setovati promenjivu, tj. smestiti podatke u privremenu memoriju servera. U ovom slučaju, za to su zadužene ugrađeni rukovaoc događajem JavaScript jezika *onBlur* (na promenu) i korisnička funkcija *mySubmit().*

```
function mySubmit() {
document.formName.submit();
}
```

Izgled tabela, pozadine, veličinu, boju i tip fonta određujemo, kao što je napomenuto, ***master.css*** fajlom.

Unutar fajla ***konekcija.php*** nalazi se klasa *SystemComponent* koja definiše parametre konekcije sa bazom podataka.

Klasa *Konekcija* nalazi se unutar fajla ***funkcije.php*** i definiše osobine i metode potrebne za rad aplikacije. Zamišljena je kao kombinacija ugrađenih funkcija PHP jezika i korisničkih funkcija prilagođenih potrebama aplikacije.

Ulaz u magacin (fajl ulaz_osn.php) je centralni deo aplikacije. Pomoću sesija je omogućeno praćenje korisnika tokom cele njegove sesije na Web lokacije. Ovo omogućuje prikazivaje odgovarajućeg sadržaja, koji će zavisiti od želja pojedinačnog korisnika. Ovim je obezbeđeno čuvanje potrebnih podataka u obliku promenljivih sesija. Kao što je već ranije napomenuto, da bi se dodelila vrednost promenljive i upisala na server potrebno je izvršiti osveženje stranice. U tom slučaju, ako vrednosti ostalih promenljivih nisu sačuvane kao promenljive sesije izgubiće se svi podaci koje smo želeli da sačuvamo tj. nova vrednost svih promenljivih će biti NULL. Pošto je zamišljeno da se u okviru aplikacije dozvoli korisniku popunjavanje svakog pojedinačnog polja, a JavaScript funkcijom smo omogućili osvežavanje stranice napuštanjem svakog tog polja, moralo se voditi računa kako o dodeljivanju novih vrednosti promenljivama, tako i čuvanju vrednosti promenljivih koje nismo menjali.

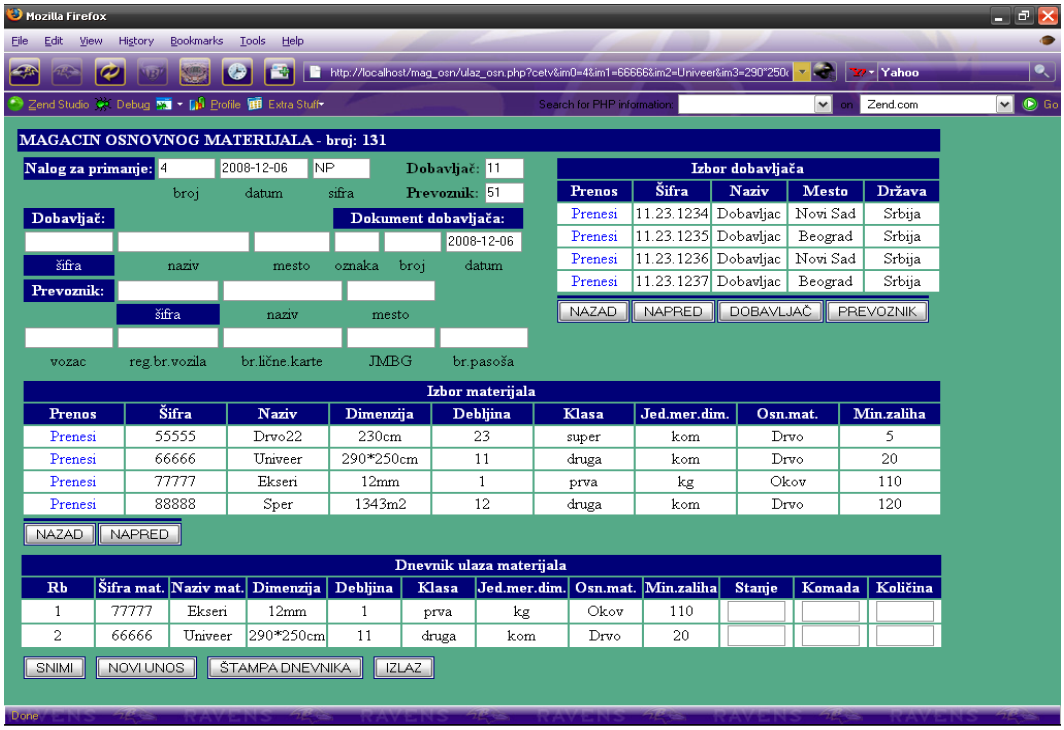
```
if (isset($_REQUEST['datum'])) $datpri=$_REQUEST['datum'];
else {$datpri=$_SESSION['datpri'];}
if (!isset($_SESSION['datpri'])) $datpri=date('Y-d-m');
if (isset($_REQUEST['sifra'])) $sifpri=$_REQUEST['sifra'];
else {$sifpri=$_SESSION['sifpri'];}
```

Uslovnom strukturom *if-else* se proverava da li želimo promeniti vrednost promenljive ili zadržati staru. Svakim novim prolaskom parsera PHP koda proverava se da li je promenljiva setovana (*isset*), ako jeste dodeljujemo joj novu vrednost a ako nije (*else*) zadržavamo joj staru vrednost u obliku promenljive sesije. U ovoj aplikaciji su praktično svi podaci dodeljivani

promenljivima sesije, pa čak i sadržaji čitavih tabela u obliku multidimenzionalnih vektora (nizova).

$_SESSION['vekt'][$_SESSION['idv']][$i]=$_REQUEST["im$s"];

Moduli u obliku fajlova ***dobavljac.php*** i ***prevoznik.php*** služe za selektivno prikazivanje podataka. Prikazivanje se vrši putem tabele za izbor dobavljača ili prevoznika u gornjem desnom uglu aplikacije (slika 3.).



Slika 4.1.3: Web verzija aplikacije "Ulaz u magacin"

Selektovanje jednog ili drugog modula se vrši jednostavnom promenom upita bazi podataka. Modul dobavljača vrši upite bazi podataka poljima tabele namenjene za dobavljače, i dobijene rezultate prikazuje u vidu tabele na ekranu. Modul prevoznika vrši identičan posao ali poljima tabele namenjenima za prevoznike.

Za razliku od ostalih tabela, donja, tj. ***dnevnik ulaza materijala*** (Slika 4.1.4..) poseduje specifičnu osobinu da pored toga što ispisuje prenesene podatke dozvoljava korisniku da i sam upiše određene vrednosti poput ***Stanje***, ***Komada*** i ***Količina***. U isto vreme, ova tabela nije unapred određena količinom podataka koji definišu broj kolona i redova, već se mora formirati u zavisnosti od korisnika, tj. broja redova materijala koje korisnik selektuje. Ovaj problem je rešen uz pomoć multidimenzionih vektora i korisničke funkcije koja iscrtava redove i kolone tabele, i u isto vreme ih popunjava podacima dozvoljavajući korisniku da dopunjava polja koja su ostala prazna[58]

Slika 4.1.4. Tabela Dnevnik ulaza materijala

```
if (isset($_SESSION['idv']))
{
        for ($i=0; $i <= 7; $i++)
        {
                $s=$i+1;
                $_SESSION['vekt'][$_SESSION['idv']][$i]=$_REQUEST["im$s"];
        }
        $_SESSION['idmat'][$_SESSION['idv']]=$_REQUEST['im0'];
        ++$_SESSION['idv'];
        echo $mag -> Tabela($_SESSION['vekt'], $_SESSION['df'],
        $_SESSION['dk'], $_SESSION['dm']);
}
```

Navedeni primer koda služi da napravi tabelu Dnevnik ulaza materijala. Promenljive sesije čuvaju podatke o prenesenim redovima tabele. Primeti se da su multidimenzioni vektori jer pored kolona koje su u vidu prvog indeksa imaju i drugi indeks gde se čuvaju podaci o redovima. Podaci koje korisnik upisuje čuvaju se u promenljivama sesije pod imenima *$_SESSION['df']*, *$_SESSION['dk']*, *$_SESSION['dm']*. Funkcija *Tabela()* koja prihvata četiri parametra je zadužena da napravi tabelu i ispiše podatke onako kako ih je korisnik prosledio. Funkcija se nalazi u okviru klase *Konekcija* i kao takva je metoda instanciranog objekta *$mag*.

Aplikacija na kraju ima četiri mogućnosti koje pruža korisniku. Da snimi, traži novi unos, štampa dnevnik ili izađe iz aplikacije.

4.1.3 Ograničenja grafičkog interfesja web aplikacije

Grafički interfejs rađen u HTML-u ima svoja ograničenja kako u funkcionalnosti tako i izgledu aplikacije, a naročito u optimizaciji aplikacije za rad u različitim browserima.

Česta je upotreba osvežavanja stranice iz razloga unošenja podataka, što dosta usporava rad aplikacije i izaziva utisak isprekidanosti, ali je to jedini način u ovakvoj varijanti. Bilo kakvo štampanje izveštaja se može izvršiti samo na serveru, bilo udaljenom ili lokalnom.

Nije moguće "skrolovanje" podataka u rešetkama a manipulacija podacima tipa promene pozicije ili veličine reštke, promene redosleda kolona, širine pojedinačnih kolona, i slično, praktično su nemoguće.Zbog potrebe za osvežavanjem ekrana prilikom unosa podataka u neko od input polja jer se podaci moraju proslediti na server sa korisničkog računara da bi bili ažurirani, nije moguće pritiskom na određeni taster tastature (na primer tab, ili enter), preći sa polja u polje radi unosa.

Unos vrednosti direktno u polja rešetke je ostvaren umetanjem input objekata tipa tekst u pozicije rešetke gde se dozvoljava unos, a prosleđivanje se vrši uz pomoć promenjive sesije tipa višedimenzionog niza.Kod velikog broja slogova u tabeli, odnosno data rešetci, kretanje po redovima rešetke obavlja se klikom na dugmad "napred" ili "nazad".

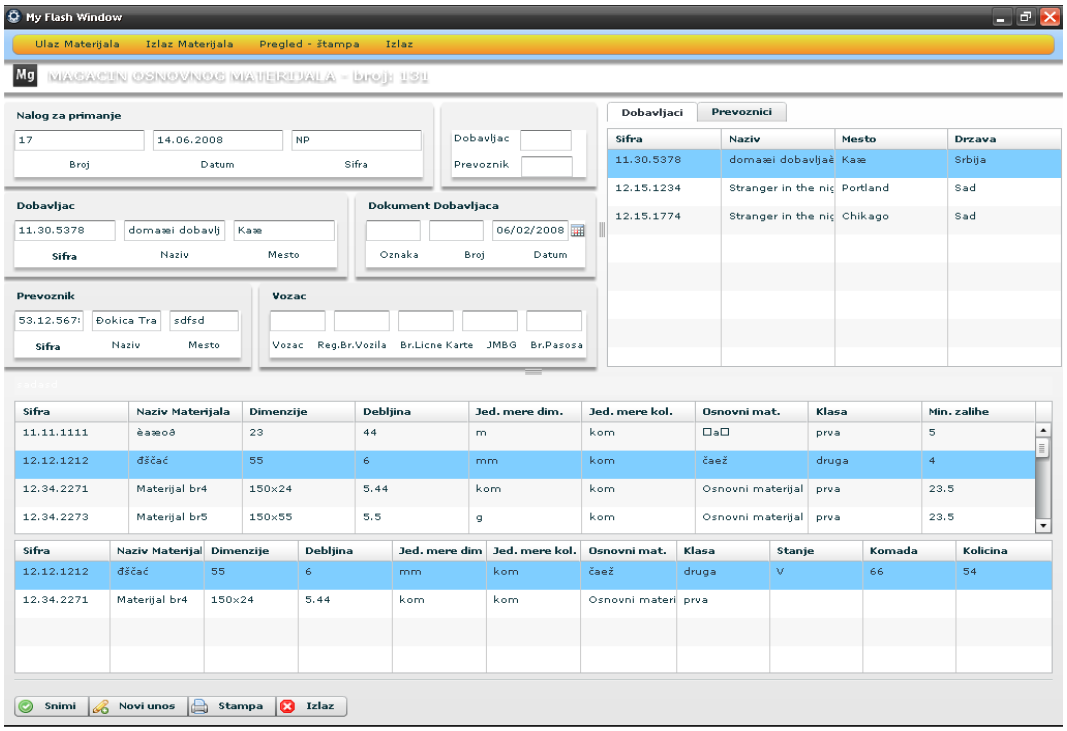
4.1.4 Web2.0 aplikacija -RIA grafički interfejs

Grafički interfejs Web2.0 aplikacije je napravljen celokupno u Adobe Fleks tehnologiji. Aplikacija podržava MVC (Model-View-Controler) arhitekturu, pri čemu se kompletan View (grafički korisnički interfejs) izvršava na strani klijenta, za razliku od prethodno opisane web verzije. Sam kontroler može se izvršavati na serverskoj ili klijentskoj strani, ili kombinovano, pri čemu svaka od ove tri varijante ima svojih prednosti i mana.

Vizuelni raspored elemenata grafičkog korisničkog interfejsa se ne razlikuje mnogo od prethodne dve verzije, ali grafički, kao što se i vidi na slici 4.1.5., pruža daleko više mogućnosti korisniku. Primenom Fleks-a je omogućena i promena veličine komponenata od strane korisnika direktnim prevlačenjem graničnika komponenti, kao i podešavanje rasporeda podataka po želji. Dobijene podatke u data rešetkama je moguće razvrstavati po abecedi, vrednostima i dr., zatim je moguće menjati redosled kolona i sve to na veoma jednostavan način. Pošto se korisnički interfejs izvršava na klijentskoj strani, nema post ili get prosleđivanja podataka ka serveru u cilju osvešavanja ekrana, pruža se mogućnost kretanja po poljima interfejsa i rada samo putem tastature, bez upotrebe miša.

U klasičnoj web verziji prilikom napuštanja input polja ekran se osvežavao zbog JavaScript funkcije i kursor bi uvek ponovo bio postavljen na početak, čime je nemoguće bilo kretanje putem tastature. Svi podaci u tekstualna polja i tabele se učitavaju sa servera putem XML dokumenta koje generiše PHP skript na serverskoj strani.

Logički deo aplikacije (kontroler) koji radi na serverskoj strani, bilo na udaljenom ili lokalnom serveru, zadužen je za komunikaciju sa bazom podataka, matematičke proračune, selekciju podataka na osnovu zadatih kriterijuma, obezbeđivanje inicijalnih podataka aplikacije, rad sa datumima i vremenom i mnoštvo drugih radnji potrebnih za normalno funkcionisanje aplikacije. Svi fajlovi zaduženi za ove zadatke su napisani u PHP programskom jeziku.Osnovni problem pri pravljenju logičkog dela programa je komunikacija sa korisničkim interfjsom koja se odvija putem XML dokumenata. Naime, PHP programski jezik ne poseduje klase i metode za jednostavno parsiranje XML dokumenata. Klase koje su namenjene za parsiranje nisu pogodne za novi E4X standard XML-a koji koristi AS3. Ono što je potrebno je nova klasa za parsiranje XML-a koja pristupa vrednostima tagova i atributa po njihovim imenima. Iz tog razloga, za potrebe ove aplikacije razvijena je takva PHP klasa. Napisana klasa pod nazivom *XMLParser* je rezultat traženih osobina i metoda koja će čitati XML dokumente prosleđene na server iz korisničkog interfejsa.

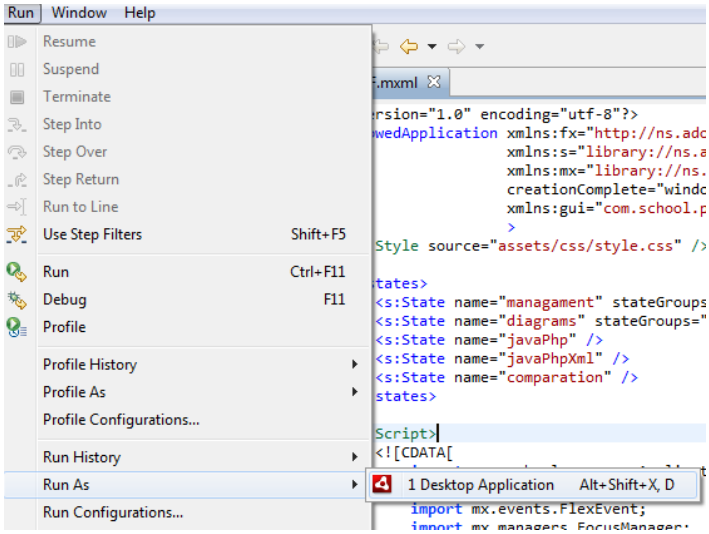


Slika 4.1.5. RIA (Web2.0) grafički interfejs aplikacije "Ulaz u magacin"

4.1.5 Analiza sistemskih resursa: procesor i memorija

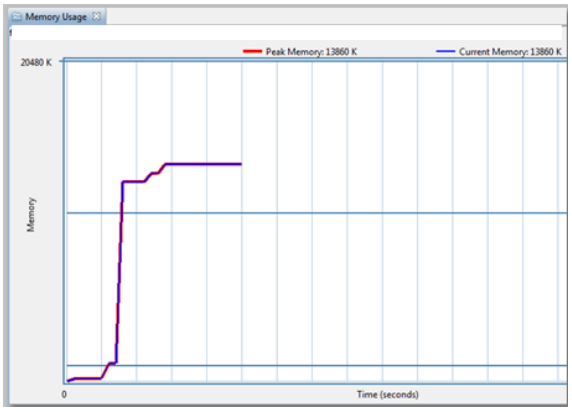
U aplikaciji se analizira broj živih instanci, procenat zauzetosti memorije instanci od različitih klasa koje egzistiraju u programu, instance koje zauzimaju mnogo memorije,

prosečno vreme koje zauzimaju motode tokom svih sekcija angažovanosti, vreme po sekcijama. Na osnovu ovih parametara, može se optimizovati kod. Na grafikonu se mogu pratiti promene zauzetosti memorije na osnovu aktivnosti pojedinih komponenti u aplikaciji. Ovo se postiže pomoću alata Profile as, koji omogućava praćenje svakog segmenta aplikacije, daje podatke o broju instanciranih objekata, zauzeću memorije, identifikuje performanse unutar aplikacije. Stavka za pokretanje alata profile as, adobe FleshBuildera, nalazi se u meni Run, slika 4.1.6.



Slika 4.1.6. Pokretanje analizatora aplikacije iz menija Run.

Kada se pokrene aplikacija, instancira se objekat klase StudentAMF.mxml. Nagli skok memorije desio se na grafiku, u trenutku učitavanja 500 objekata sa servera u klijentsku aplikaciju. Cela aplikacija nije veća od 20 MB, što se vidi na slici 4.1.7. Trenutno zauzeće memorije je 13860Kb, što je prikazano stavkom Current Memory: 13860Kb. Maksimalno zauzeta memorija za aplikaciju je bila 13860Kb, crvena linija na grafiku.



Slika 4.1.7. Prikazuje maksimalno zauzeće memorije, crvena linija, trenutno zauzeće memorije, plava linija, i promenu zauzetosti memorije tokom vremena, prikazuje memoriski prostor koji zauzima aplikacija, 20480 Kb.

Klasa StudentVO ima 500 živih instanci, (slika 4.1.8), i zauzima 67,7% instanci od ukupnog broja instanciranih objekata u memoriji od različitih klasa.

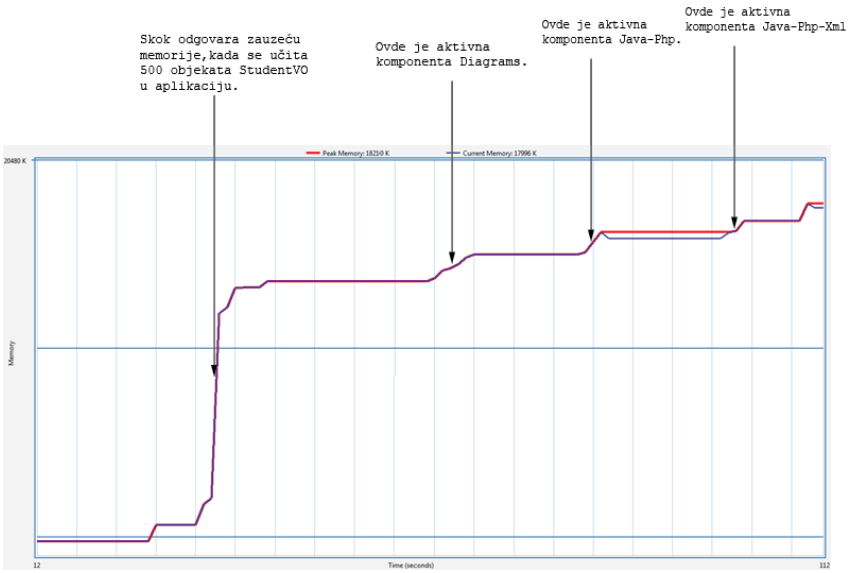
Na slici 4.1.8. je data lista klasa u aplikaciji, koje imaju instancirane objekte u memoriji, paket u kome se nalaze klase, ukupan broj instanciranih objekata za svaku klasu, broj živih instanci svake klase, veličinu memorije koju zauzimaju objekti svake klase izražene u bajtima i procentualno u odnosu na sve instance koje su egzistirale, trenutno zauzeće memorije koje zauzimaju živi objekti date klase. U ovoj tabeli se vidi da najviše memorije zauzimaju instance klase StudentVO, kojih ima 500 smeštenih u radnoj memoriji. Grafičke komponente zauzimaju dosta memorije, jer su grafičke klase glomazne, te stoga neophodno je svesti broj ovakvih klasa na minimum, kako bi ukupno zauzeće memorije bilo manje, a pri iscrtavanju dosta zauzimaju procesorskog vremena.

Class	Package (Filtered)	Cumulative Instances	Instances	Cumulative Memory	Memory
StudentVO	com.school.vo	1000 (23.33%)	1000 (57.97%)	76000 (25.32%)	76000 (39.55%)
Vector.<*>	__AS3__.vec	1244 (29.02%)	226 (13.1%)	59384 (19.79%)	18176 (9.46%)
MethodClosure	builtin.as$0	1253 (29.23%)	0 (0.0%)	50144 (16.71%)	0 (0.0%)
StudentTextInputSkin	com.school.puremvc.view.gui.skins	11 (0.26%)	11 (0.64%)	14740 (4.91%)	14740 (7.67%)
ButtonBarButtonSkin	com.school.puremvc.view.gui.skins	8 (0.19%)	8 (0.46%)	11200 (3.73%)	11200 (5.83%)
StudentButtonSkin	com.school.puremvc.view.gui.skins	7 (0.16%)	7 (0.41%)	9828 (3.27%)	9828 (5.11%)
StudentDataGridSkinInnerClass5	com.school.puremvc.view.gui.skins.grids	6 (0.14%)	6 (0.35%)	8376 (2.79%)	8376 (4.36%)
StudentDataGridSkinInnerClass9	com.school.puremvc.view.gui.skins.grids	16 (0.37%)	16 (0.93%)	6464 (2.15%)	6464 (3.36%)
HLayoutElementFlexChildInfo	HorizontalLayout.as$125	67 (1.56%)	0 (0.0%)	6432 (2.14%)	0 (0.0%)
ButtonBarSkinInnerClass1	com.school.puremvc.view.gui.skins	4 (0.09%)	4 (0.23%)	5856 (1.95%)	5856 (3.05%)
Property	flashx.textLayout.property	88 (2.05%)	88 (5.1%)	3872 (1.29%)	3872 (2.02%)
Vector.<int>	__AS3__.vec	46 (1.07%)	18 (1.04%)	3384 (1.13%)	1624 (0.85%)
ButtonBarSkinInnerClass2	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2928 (0.98%)	2928 (1.52%)
ButtonBarSkinInnerClass0	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2928 (0.98%)	2928 (1.52%)
ButtonBarSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2640 (0.88%)	2640 (1.37%)
PriorityBin	PriorityQueue.as$518	88 (2.05%)	88 (5.1%)	2112 (0.7%)	2112 (1.1%)
StudentAMF		1 (0.02%)	1 (0.06%)	2104 (0.7%)	2104 (1.09%)
StudentDataGridSkinInnerClass4	com.school.puremvc.view.gui.skins.grids	5 (0.12%)	5 (0.29%)	2020 (0.67%)	2020 (1.05%)
StudentDataGridSkinInnerClass2	com.school.puremvc.view.gui.skins.grids	5 (0.12%)	5 (0.29%)	2020 (0.67%)	2020 (1.05%)
LayoutElementFlexChildInfo	VerticalLayout.as$128	21 (0.49%)	0 (0.0%)	2016 (0.67%)	0 (0.0%)
StudentManagerComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1484 (0.49%)	1484 (0.77%)
StudentDataGridSkin	com.school.puremvc.view.gui.skins.grids	1 (0.02%)	1 (0.06%)	1468 (0.49%)	1468 (0.76%)
StudentComboSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	1 (0.06%)	1428 (0.48%)	1388 (0.72%)
BackgroundSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	1 (0.06%)	1364 (0.45%)	1324 (0.69%)
Vector.<Number>	__AS3__.vec	12 (0.28%)	8 (0.46%)	1360 (0.45%)	992 (0.52%)
StudentTableComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1324 (0.44%)	1324 (0.69%)
StudentControlComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1324 (0.44%)	1324 (0.69%)
EnumPropertyHandler	flashx.textLayout.property	50 (1.17%)	50 (2.9%)	1000 (0.33%)	1000 (0.52%)
SizesAndLimit	HorizontalLayout.as$125	31 (0.72%)	0 (0.0%)	992 (0.33%)	0 (0.0%)
NumberPropertyHandler	flashx.textLayout.property	23 (0.54%)	23 (1.33%)	920 (0.31%)	920 (0.48%)
StudentDataGridSkinInnerClass1	com.school.puremvc.view.gui.skins.grids	2 (0.05%)	2 (0.12%)	912 (0.3%)	912 (0.47%)
_StudentAMF_mx_managers_System...		1 (0.02%)	1 (0.06%)	708 (0.24%)	708 (0.37%)
Edge	Parcel.as$90	22 (0.51%)	2 (0.12%)	704 (0.23%)	64 (0.03%)
TextLayoutFormat	flashx.textLayout.formats	29 (0.68%)	24 (1.39%)	696 (0.23%)	576 (0.3%)
SizesAndLimit	VerticalLayout.as$128	20 (0.47%)	0 (0.0%)	640 (0.21%)	0 (0.0%)
CompositionCompleteEvent	flashx.textLayout.events	11 (0.26%)	0 (0.0%)	616 (0.21%)	0 (0.0%)
StandardFlowComposer	flashx.textLayout.compose	11 (0.26%)	0 (0.0%)	616 (0.21%)	0 (0.0%)
StudentDataGridSkinInnerClass6	com.school.puremvc.view.gui.skins.grids	1 (0.02%)	1 (0.06%)	456 (0.15%)	456 (0.24%)
MethodQueueElement	UIComponent.as$131	18 (0.42%)	0 (0.0%)	432 (0.14%)	0 (0.0%)
SimpleCompose	flashx.textLayout.compose	1 (0.02%)	1 (0.06%)	392 (0.13%)	392 (0.2%)
LineSegment	Path.as$200	12 (0.28%)	12 (0.7%)	384 (0.13%)	384 (0.2%)
TB_VJHelper	VerticalJustifier.as$93	11 (0.26%)	0 (0.0%)	352 (0.12%)	0 (0.0%)
StudentProxyJson	com.school.puremvc.model	1 (0.03%)	1 (0.08%)	72 (0.03%)	72 (0.05%)

Slika 4.1.8. Prikazuje listu svih objekata koje trenutno egzistiraju u aplikaciji

Pritiskom na dugme Next u aplikaciji, slika5.3., poziva se 500 novih instanci iz baze podataka. Sada se u tabeli prikazuje da je ukupno učitano 1000 objekata, i da postoji 1000 živih instanci. Međutim, logika profil as-a pripremila je 500 instanci za uništavanje, čeka neki moment kada će ih uništiti, međutim, ako se pritisne na dugme Gabrage Collector, onda se ručno pokreće logika za uklanjanje svih referenci na objekte koji se više ne koriste, i time se eleminiše curenje memorije. Dakle, pri učitavanju novih 500 instanci iz baze, memorija se povećala na 9710Kb, a onda se spustila na 7079kb, jer je garbage collector očistio 500 instanci iz memorije, koje se više ne koriste. U tabeli, objekti klase StudentVO zauzimaju 57,97% memorije, skinovi za tekstualna polja od klase StudentTextInputSkin, zauzimaju 7,67% memorije. Međutim, proksi StudentProxyJson zauzima 0.05% memorije. To su klase koje nemaju teških objekata, i zato zauzimaju malo memorije. Postoji jedna instanca kontrolera StudentControlComponent, koja zauzima 0,69% memorije. Za ButtonBarSkin dugme ukupno je instancirano 7 instanci, trenutno u memoriji egzistira 7, zauzimaju 1.37% memorije. Kada bi se izračunala memorija za sve skinove, StudentComboSkin 0,91%, ButtonBarSkinInnerClass1 1,93%, itd, vidi se da zauzimaju dosta memorije.

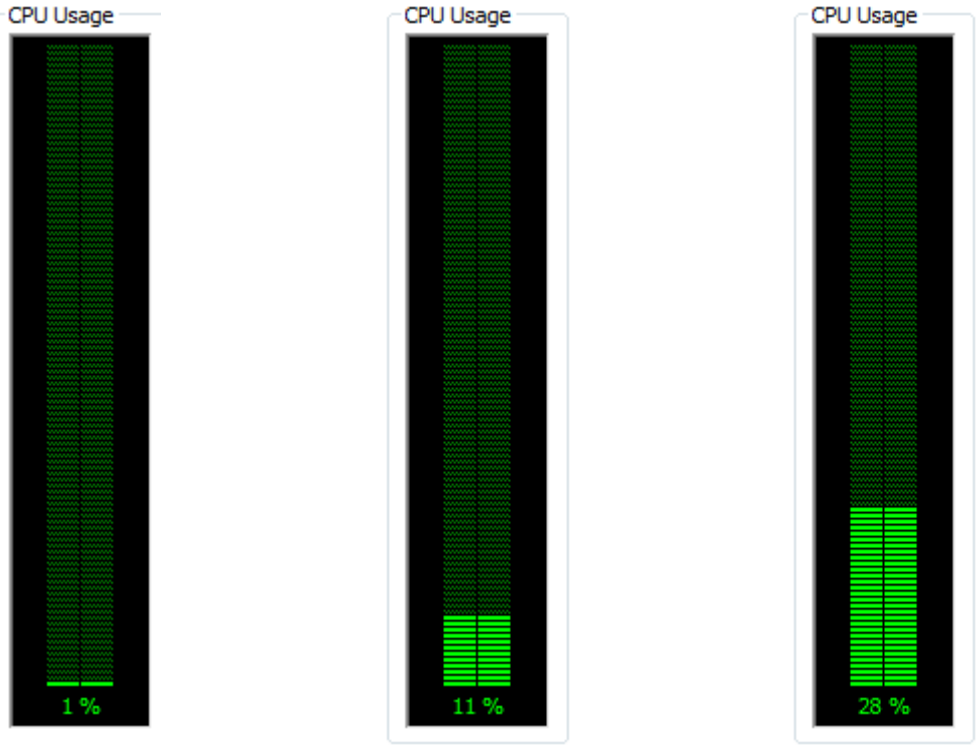
Klikom na dugme Diagrams u ButtonBaru, slika 4.1.11., dolazi do skoka memorije, zato što je uključena još jedna komponenta. Ako se klikne na dugme Java-Php dolazi do ponovnog skoka grafika, koji pokazje zauzeće memorije, isti slučaj je i kada se uključuju druge komponente.



Slika 4.1.9. Slika Prikazuje memoriski prostor koji zauzima aplikacija, kada korisnik vrši interakciju sa aplikacijom.

Fleks ne smanjuje memoriju kada Diagrams ili druga komponenta ima vrednost null, jer virtuelna mašina procenjuje koliko se troši memorije i drži rezervisanu memoriju. Skupljač smeća nije obavio posao brisanja zato što je procenio da će trebati memorija.

Kada je u pitanju procesorsko vreme, najviše procesorskog vremena se troši pri učitavanju podataka sa servera u aplikaciju. U slobodnom režimu, zauzetost procesora, je mala, slika 4.1.10a.

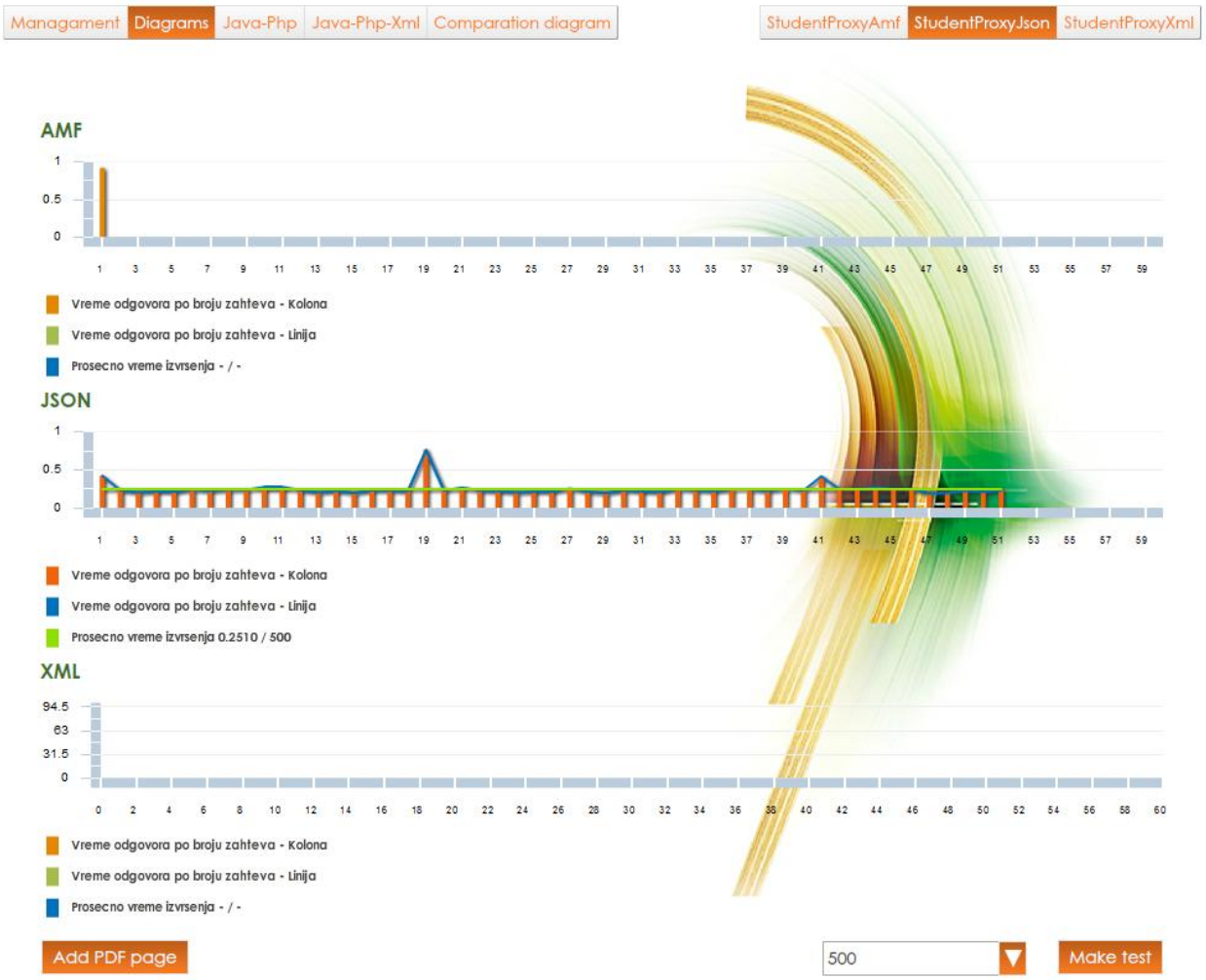


a) b) c)

Slika 4.1.10. Opterećenost procesora,a) u slobodnom režimu, b) kada se učitava 500 objekata, klase StudentVO,c) tokom kreiranja grafikona, za bilo koju tehnologiju prenosa amf, JSON ili xml, kada je komponenta Diagrams aktivna.

Kada se pritisne dugme Next,na radnoj aplikaciji, i pri tome se učita novih 500 objekata klase StudentVO, onda se procesor znatno optereti, opslužujući aplikaciju, što se vidi na slici 4.1.10. b. Upoređujući zaposlenost procesora na slici 4.1.10 a i 4.1.10 b, vidi se da je opterećenost u drugom slučaju za 10% veća.

Ako je aktivna komponenta Diagrams, slika 4.1.11. i pri tome se meri vreme učitavanja podataka u aplikaciju, s tim što iteracija se ponavlja 50 puta, onda se procesorsko vreme znatno više troši, što se vidi i na slici 4.1.10.c



Slika 4.1.11. Prikazuje učitavanje podataka sa server, u JSON format, i pri tome se meri opterećenost procesora.

Testiranje je urađeno na računaru sledećih performansi: CPU - QuadCore AMD Phenom X4 9550, 2200 MHz (11 x 200), L1 64 KB per core, L2 512 KB per core (On-Die, ECC, Full-Speed), L3 2 MB (On-Die, ECC, NB-Speed), RAM Memory - 3 Gb, Graphic card - nVIDIA GeForce 9400 GT. Za testiranje je korišćeno okruženje autorske aplikacije.

5. KOMPARATIVNA ANALIZA DESKTOP I WEB APLIKACIJA

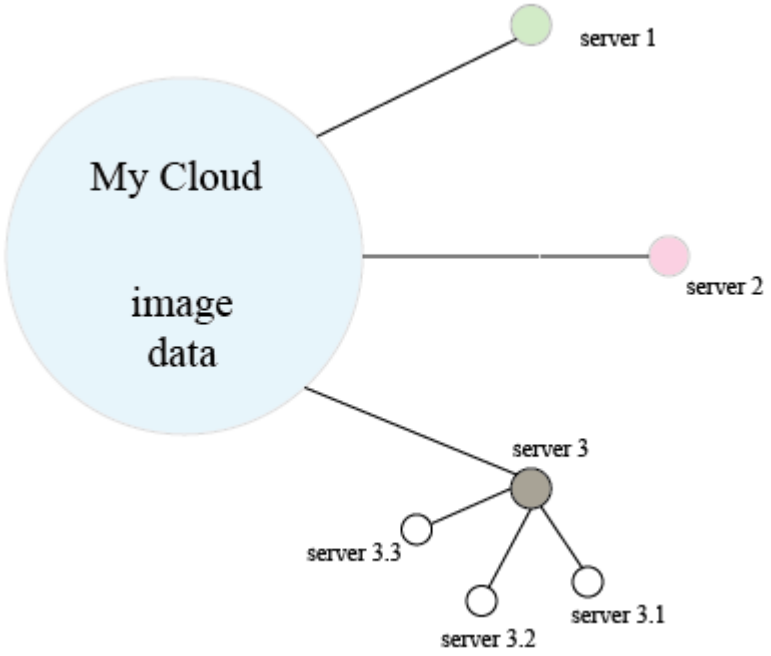
Praveći komparaciju između desktop i web aplikacija, može se zaključiti, desktop i Web aplikacije se razlikuju po izgledu i funkcionalnosti. Desktop aplikacija koristi sve resurse računara, dok Web aplikacija ne koristi resurse računara, već resurse browser-a, i sa druge strane koriste se resursi servera za neki proračun, ali prikaz podataka korisniku ograničen je samo na browser. Browser je desktop aplikacija na kojoj se prikazuje sadržaj sa Web-a t.j. sa nekog udaljenog servera. Na serveru može da bude bilo kakva aplikacija, a browser je ograničen veličinom interne memorije, u koju se može učitati limitirana količina podataka. Naime, prvi browser-i su imali memorijske limite, nisu mogli učitati aplikaciju od nekoliko stotina megabajta. Ovo ograničenje browser-a, predstavljalo je problem. Danas browser-i predstavljaju kompleksne aplikacije, sa ugrađenom komunikacijom sa računarskim sistemom, kroz razne plug-in-ove, i mogu da koriste resurse računara, kao što je grafičko ubrzanje, što omogućuje prikaz 3D sadržaja na browser-u, jer on koristi grafičku akseleraciju sa grafičke kartice direktno. OpenGL je tehnologija koja dozvoljava da se mogu koristiti 3D efekti za prikaz na Web-u, preko browser-a. OpenGL direktno se oslanja na procesor sa grafičke kartice. Dakle, sa servera se dovlači aplikacija, koja se prikazuje na korisnikovom računaru u browser-u.

Kada je u pitanju Web 1.0, Web 2.0 i doba posle Web 2.0, može se reći da je najbitnija osobina Wb 2.0 iskoristljivost[8], naime, sve što je bilo na ekranu moglo je da bude korisno za nešto, pri čemu je sadržaj na ekranu nudio mogućnost interaktivnosti sa korisnikom, ili obaveštenje korisniku. Stoga, Web2.0 predstavlja Web, gde korisnik učestvuje, a to je Read/Write Web, omogućava interakciju između korisnika i Web-a[4], slika 4.1.13, uvodi alate poput bloga, wikis-a, tagove, widget-e i RSS gde se nalaze podaci u bilo kakvoj formi. Web2.0 koristi tehnologiju posle statičkih stranica, prethodnih web sajtova. Web 1.0 je prezentacionog karaktera, gde se vrši prezentacija firmi, korisnik je mogao samo da čita na webu, zvao se i Read Web, slika 4.1.13. Web 2.0 je bio bogatiji izgledom u poređenju sa Web 1.0, sa znatno boljim korisničkim interfejsom. Tehnologija JavaScript-a je omogućila interaktivnost, asinhroni prenos podataka je ubrzao komunikaciju između korisnika i servera, gde se pojedini delovi aplikacije izvršavaju paralelno, bez usaglašavanja vremenskih tajminga[5]. Na ovaj način, je značajno poboljšana raspodela posla, kao i mogućnost da se blok koda može raščlaniti

i dodeliti različitim nitima na izvršavaje. Po nekima kraj Web2.0 se smatra 2010-om godinom, a od te godine je period posle Web 2.0.

Period posle Web2.0, na Web-u karakteriše personalizacija, semantički web, silosi, mikroformati, lokalizacija[4,8]. Ovo doba još nije definisano, nema ime. Personalizacija je prilagođavanje interneta jednoj osobi[4]. Danas za personalizaciju se koristi popularni servis *Cloud computing*, a računar služi kao pristupna tačka na internetu**.** Cloud computing je prostor na web-u, gde korisnik može da pretražuje i skladišti svoje podatke. Podaci, razni dokumenti, slike, se personalizuju, a nalaze se na raznim serverima, ali korisniku, servis Cloud computing, prikazuje sve to na jednom mesto, što je prisutan visok nivo apstrakcije.

Na slici 5.1. je prikazan My Cloud, koji sadrži podatke, slike, dokumenta i drugo. Ovi podaci se u virtuelni prostor prikazuju sa raznih servera, server1, server2 i server 3. Na serveru 3 su povezani serveri: server 3.1, server 3.2 i server 3.3.

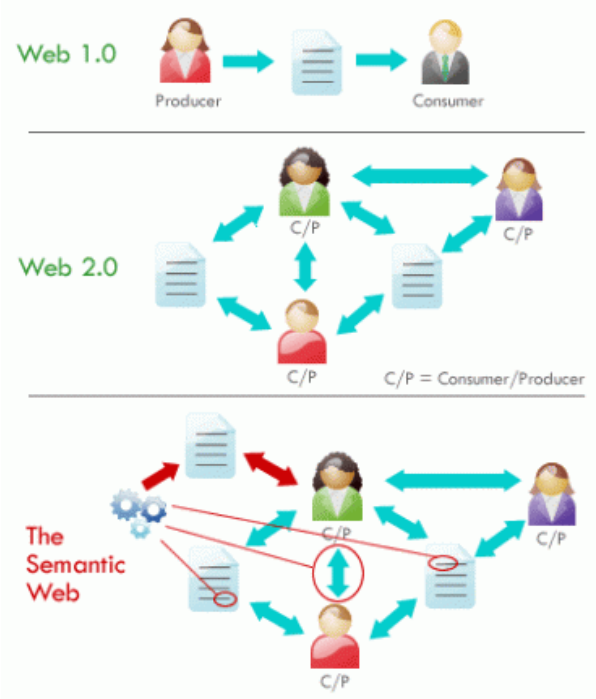


Slika 5.1. Cloud servis

Kada korisnik traži neki podataka, ne traži ga na nekom serveru, već u servisu Cloud computing, a nije mu bitno na kom serveru je on smešten.

Druga karakteristika ovog doba, je semantička pretraga na web-u. Semantički web je mesto gde mašina može da čita web stranice kao što čita čovek, mesto gde pretraživači i softver program može biti bolji skupljač podataka na internetu(pasivni program koji prikuplja podatke sa svih strana na jedno mesto), koji nalazi ono što korisnik traži. Prema tome, semantika predstavlja mogućnost softvera da prepozna potrebu korisnika, kada se upiše ključna reč na

web-u, i ne samo što se traži po ključnoj reči, već se traži koren ključne reči, provlačeći je kroz vremena, padeže, prati se celokupna istorija koju je korisnik proveo na webu, i pokušava da shvati šta korisnik hoće da dobije na webu, a ne neku listu sajtova koju pretraživač misli da treba prikazati. Takođe, ovaj period karakteriše i lokalizovana pretraga, naime, kada korisnik traži nešto, softver, pomoću korisnikovih podataka, a najvažniji podatak je gde se korisnik nalazi, GPS tehnologija, da rezultate. Dakle, period posle Web 2.0 karakterišu i mikroformati, silosi. Mikroformat se odnosi na kvantum podataka, koji se nalaze na računaru a dostupni su svima na mreži. Silosi su delovi Web farme, namenjeni za skladištenje podataka[8]. Web farmu sačinjavaju na primer 100 servera, a jedan server je silos. Dakle, silosi su mesta gde se skupljaju podaci. Flickr je jedan servis, koji sadrži slike. Korisnik može imati listu prijatelja na facebook-u, drugu listu na googl-e, i treću na yahoo. Ove tri liste su generalno različite. Ideja je spojiti sve te servise u jednoj tački, a za to se koriste silosi. Na slici 5.2. dat je grafički prikaz za Web1.0, Web 2.0, i semantički Web.



Slika 5.2. Web1.0, Web 2.0 i Semantički Web.

Kada se analizira MVC arhitektura i zadatak kontrolera, imajući u vidu autorsku aplikaciju, treba znati da se MVC struktura uvek nalazi samo na jednoj strani, nije raspodeljena između klijenta i servera, sastoji se iz tri dela M, V i C, ali sve zajedno čini klijenta ili servera. MVC pristup se može istovremeno koristiti i na serveru i na klijentu, kao što je slučaj sa autorskom aplikacijom, gde je na strani Java servera korišćen MVC play Framework. M je model, radi sa podacima, može biti baza podataka ili web servis. Kontroler može biti na strani klijenta i na

strani servera, izrađen u različitim ili istim tehnologijama, kao što je slučaj sa autorskom aplikacijom. On kontroliše, izdaje naredbe, vodi računa da podaci budu nađeni i prikazani u view. U autorskoj aplikaciji, kontroler ima ograničenu ulogu, služi za trigerovanje događaja, slanje naredbi i obavlja neke logičke pojave, dok model priprema podatke, obrađuje podatke koju će view da prikaže, pristupa serveru a server mu vraća podatke. Autorska aplikacija predstavlja RIA aplikaciju, koja ima serverski i klijentski deo. Na klijentskom delu je korišćen MVC, i na serverskom delu je korišćen MVC ali kada je korišćen Java server.

Kada je pitanje, da li obradu podataka vršiti na serveru ili klijentu, preporuka je, da sve komplikovane operacije treba vršiti na serveru, zato što su serveri po pravilu mnogo jači računari od klijentskih računara. Softver koji se pravi, ne pravi se za klijentov računar, jer se nezna kakav klijent ima računar. Ono o čemu treba voditi računa, da softver koji se pravi nema veliku potrošnju memorije. Prednost RIA aplikacija, je što se na serveru nalaze podaci, i tamo se može vršiti složena obrada. Na klijentu obrada podataka treba da bude svedena na minimum, kao što je priprema podataka za prikaz korisniku, a ne neke velike računske operacije.

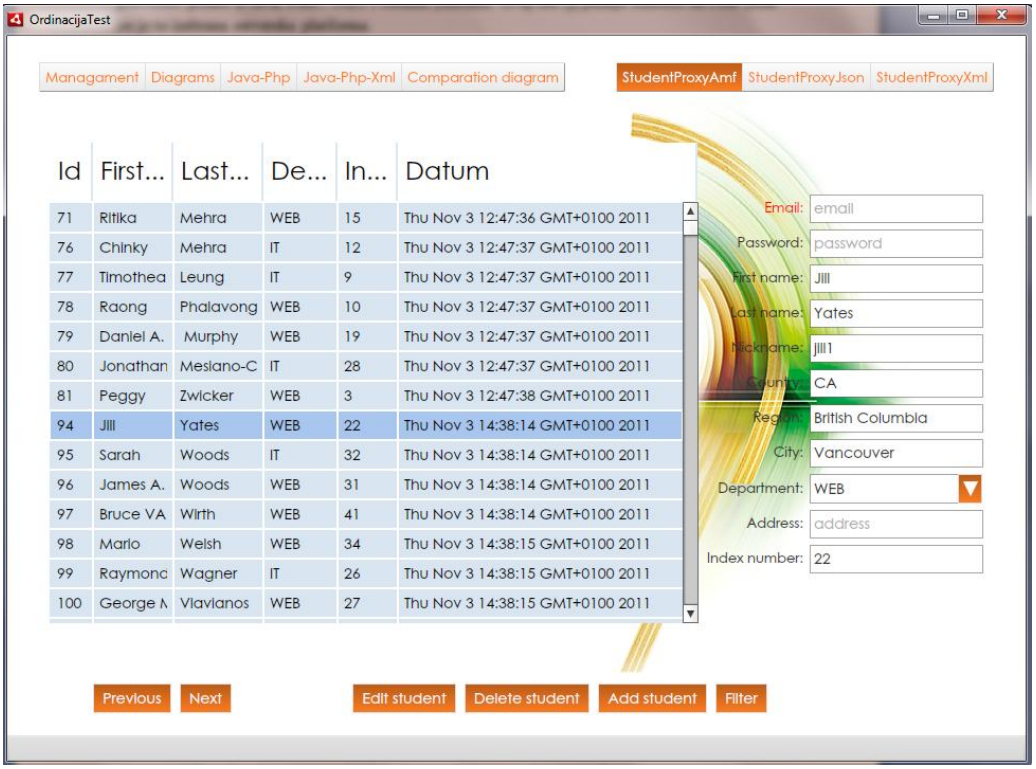
Takođe, na programeru je da odluči, da li treba slati sve podatke sa servera klijentu, ili te podatke podeliti po segmentima, jer bi u slučaju slanja svih podataka, fajl u xml-u bio veliki, što bi usporilo komunikaciju između servera i klijenta. Brzina izvršenja se može dobiti na serveru, ali se mnogo gubi na transportu. Vreme transporta je zanemarljivo, ali se povećava sa veličinom podataka.

Prema tome, po pitanju izbora desktop, Web ili Web2.0 odnosno Web koji predstavlja doba iznad Web2.0, ne postoji oštra granica, što zavisi i od zahteva klijenta. Ako se pravi aplikacija koja je zahtevna i kompleksna, onda se preporučuje da bude desktop aplikacija, jer će potpuno koristiti resurse sistema. Međutim, ako se radi neki okvir, gde će se prikazati neki sadržaj sa servera, onda je dovoljno da bude Web aplikacija. I desktop aplikacije se dele na dve vrste. Postoje desktop aplikacije koje su potpuno nezavisne od interneta, i RIA desktop aplikacije, čiji se kontejner nalazi na desktop, ali informacije i sadržaj se nalaze negde na internetu ili intranetu, što znači da nije na mašini, već negde na serveru. Kod Web aplikacije uvek je sadržaj negde na internetu ali i Web2.0 aplikacije su RIA aplikacije po svojoj definiciji. Rich ne znači da je bogata izgledom apikacija, već zato što omogućuje interakciju sa korisnikom.

Kada je u pitanju način komunikacije između servera i klijenta, sve se svodi na komplikovanost zapisa podataka, json, xml i amf struktura, i Web servisa.

Web servisi su SOAP i REST. SOAP je protokol koji prikazuje podatke u XML-u, stim da ima i deklarativni deo, gde XML tagovi nose i deklaraciju o klasama. REST servisi su klasični, get,

put, post, delete itd. To su protokoli koje koristi REST servis. Na slici 5.3. prikazano je radno okruženje aplikacije.



Slika 5.3. Radno okruženje aplikacije.

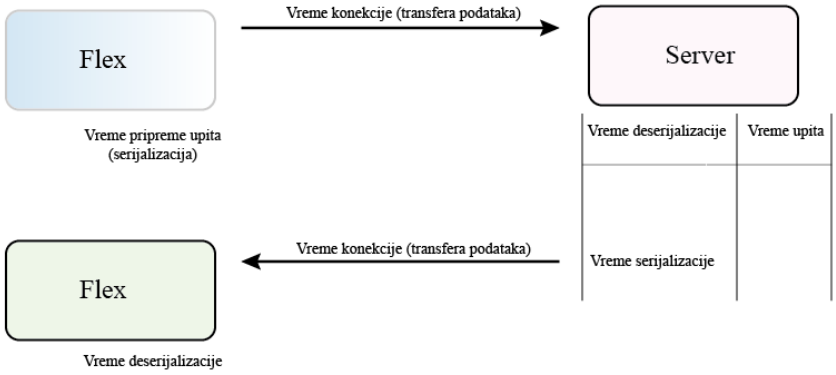
U aplikaciji se meri vreme transfera podataka od servera do klijenta za tri tehnologije formatiranja podataka, AMF, JSON i XML. Vreme izvršenja zahteva se meri od trenutka slanja zahteva serveru, sa klijentske strane, do trenutka učitavanja podataka sa servera u Flex aplikaciju. JSON, XML, AMF su protokoli koji se koriste za komunikaciju između klijenta i servera. Između njih postoji razlika u brzini pakovanju podataka, načinu formatiranja podataka, složenosti kreiranja aplikacije, veličine fajla koji se transportuje, kao i kompleksnosti u pogledu parsiranja podataka iz jednog u drugi format.

AMF radi sa binarnim podacima, na serverskoj strani prevodi neki jezik u binarne podatke, dok se na strani klijenta vrši inverzni postupak, deserijalizacija, koja prevodi nule i jedinice u vidni kod, koji se prikazuje unutar grafičkog interfejsa.

Kod JSON tehnologije, vrši se prevođenje php objekata u JSON objekte, i na kraju se JSON objekti, šalju kao tekstualni fajl klijentu, preko HTTP konekcije. Isto se radi i sa XML-om.

Radna aplikacija koristi AMFPHP paket, za komunikaciju klijentskog ActionScript3 i serverskog php jezika. Pored AMF-a, ActionScript3 ima podršku i za tehnologije XML i JSON, vršeći parsiranje podataka koji se transportuju u XML i JSON formatu.

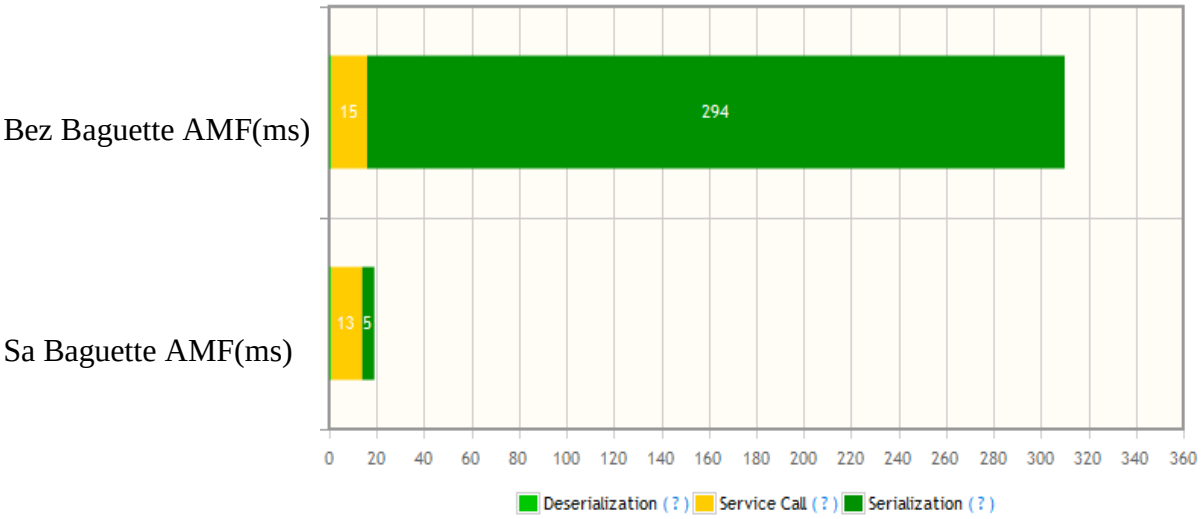
Pri transferu podataka između klijenta i servera, postoji više faza, pri čemu svaka faza zahteva određeno vreme. Na strani klijenta(Flex-a) postoji vreme prireme upita(serijalizacija) i vreme deserijalizacije, dok na strani servera postoji vreme deserijalizacije, vreme izvršenja upita u bazu podataka, i vreme serijalizacije. Postoji i vreme transfera podataka. Ova vremena su prikazana na slici 5.4.



Slika 5.4. Prikaz vremena, pri transportu podataka od servera do klijenta.

Upiti su ekstremno brzi za bilo koju tehnologiju, reda su mikrosekunde i to vreme se ne uzima u obzir. Vreme deserijalizacije na serverskoj strani obuhvata pretvaranje binarnih podataka u objekte nekog jezika. Vreme upita, se troši na čitanju podataku u bazi, ti podaci se potom serijalizuju, i onda se transferuju prema klijentu. I na kraju se podaci na strani fleksa deserijalizuju za određeno vreme. Serijalizacija ima globalno značenje, naime, u kompjuterskoj nauci, u kontekstu smeštanja podataka, serijalizacija predstavlja process prevođenja strukture podataka ili objekata u format koji može biti sačuvan(u fajlu ili bafer memoriji, ili može biti poslan kroz mrežu) i rekonstruisan, kasnije u istom ili drugačijem kompjuterskom okruženju [10].

AMF je od verzije 2.0, preuzela da proizvodi druga kompanija, i znatno je postao sporiji od JSON-a i XML-a. Sporost unosi serijalizacija, gde se troši puno vremena na pretvaranju objekata nekog jezika u binarne podatke, što se vidi na slici 5.5. Ovaj problem se rešava, dodavanjem plug in-a, Baguette AMF, osnovnoj verziji AMFPHP-a.

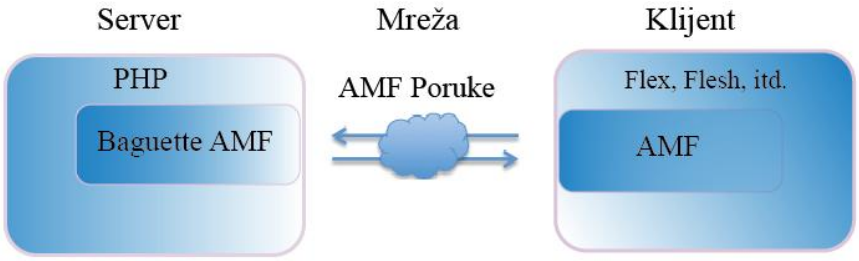


Slika 5.5. – Vremena serijalizacije, deserijalizacije i uspostave konekcije, kada se ne koristi dodak Baguette AMF i kada se koristi.

U aplikaciji se analiziraju tri bitna vremena, vreme serijalizacije, vreme deserijalizacije na strani klijenta i servera, vreme uspostave konekcije između servera i klijenta. Srednja linija, definiše vreme izvršenja servisnog poziva.

Vreme uspostave veze kod AMF tehnologije je približno isto sa i bez dodatka Baguette AMF. Vremena deserijalizacije se neznatno razlikuju u jednom i drugom slučaju. Na slici 5.5. se vidi, vreme serijalizacije podataka se drastično razlikuje sa i bez Baguette AMF-a. Kraće vreme serijalizacije sa Baguette AMF, predstavlja ubrzanje AMF-a.

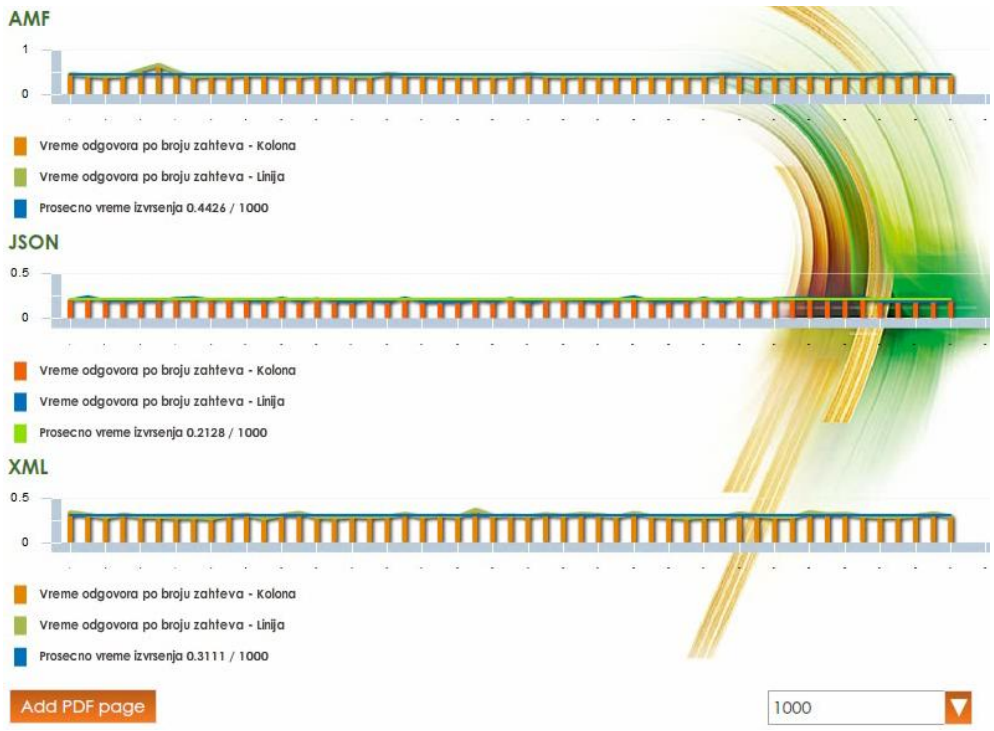
Dodavanjem plug-in-a Baguette AMF, osnovnoj verziji AMFPHP, čini AMF bržim od JSON-a i XML-a, pri transportu veće količine podataka, i time se dobija Baguette AMF. Baguette AMF sa AMFPHP je preporučljiv za kompanijske aplikacije, koje razmenjuju veliku količinu podataka. U radu se vrši prevođenje php objekata u AMF, JSON ili XML format[11,12,16,17], koji je čitljiv Flex klijentu. Flex može serijalizovati podatke u AMF, JSON ili XML formatu i deserijalizovati podatke iz ovih formata. Na slici 5.6. je prikazana komunikacija između klijenta i servera, gde se kao serverski jezik koristi php, a serijalizaciju, i deserijalizaciju podataka vrši BaguetteAMF.



Slika 5.6. Flex – PHP komunikacija

Klijent u sebi, ima ugrađen AMF format, što se vidi na slici 5.6. AMF, XML i JSON tehnologije imaju široku primenu. Spadaju u cross tehnologije, što znači da ne zavise od operativnog sistema, niti od kompjuterskog okruženja.

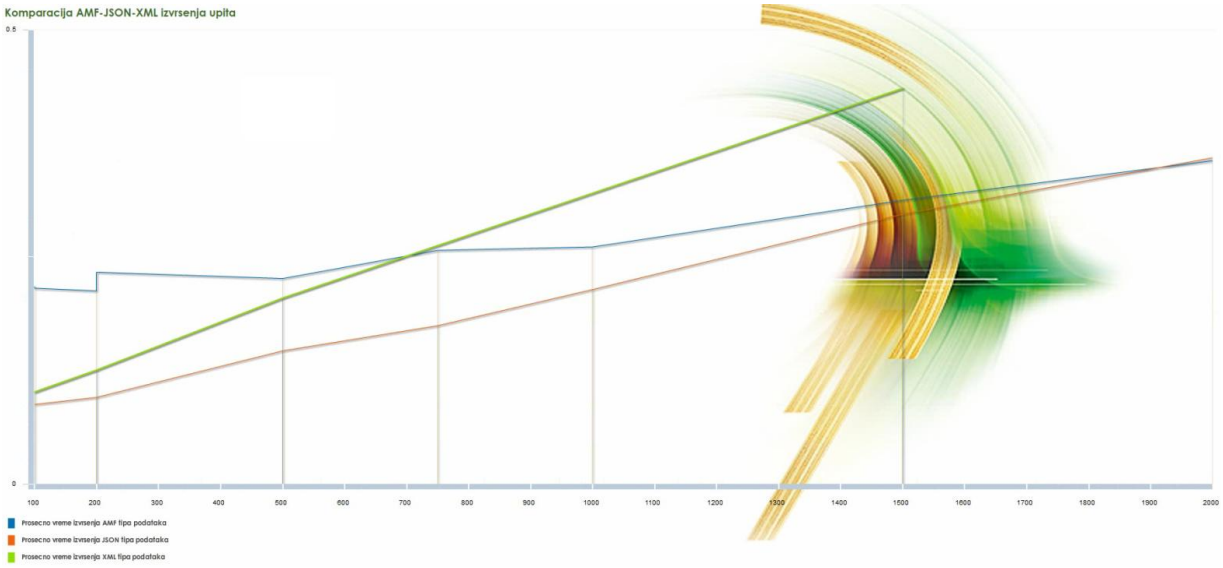
U aplikaciji, kada je aktivna komponenta Diagrams, slika 5.3, prikazuju se grafikoni za AMF, JSON i XML tehnologiju. Uporedni grafik brzine transporta podataka, od servera klijentu, koji ubuhvata vreme serijalizacije, deserijalizacije, vreme uspostave konekcije, je dato na slici 5.7. U ComboBiox-u, vrši se izbor broja podataka koji se učitavaju iz baze podataka na serveru u klijentsku aplikaciju. Izbor tehnologije prenosa vrši se pomoću button bar dugmeta selectProxyBar. Za svaku tehnologiju, vrši se pedeset iteracija učitavanja, pri čemu se u svakoj iteraciji učitava isti broj podataka, meri se vreme, i na kraju iteracije vrši se izračunavanje srednjeg vremena.



Slika 5.7. Grafikoni zavisnosti vremena od broja učitanih podataka za AMF, JSON i XML tehologiju

Testiranje je urađeno na računaru sledećih performansi: CPU - QuadCore AMD Phenom X4 9550, 2200 MHz (11 x 200), L1 64 KB per core, L2 512 KB per core (On-Die, ECC, Full-Speed), L3 2 MB (On-Die, ECC, NB-Speed), RAM Memory - 3 Gb, Graphic card - nVIDIA GeForce 9400 GT. Za testiranje je korišćeno okruženje autorske aplikacije.

Na grafikonu svaki stubić, predstavlja vreme učitavanja podataka. Horizontalna kriva linija, povezuje vrhove stubića a prava horizontalna linija na grafikonu opisuje srednje vreme učitavanja podataka. Grafikoni reprezentuju vreme, kada se učitava 1000 podataka iz baze.
Na velikom broju podataka, AMF je brži od JSON-a. Na slici 5.8, je na osnovu merenja u ovom modelu, prikazankomparacijski dijagram zavisnosti vremena od broja podataka. Na slici 5.8 se vidi, do 2000 podataka JSON je brži od AMF, a iznad ovog broja, AMF postaje brži. Pa stoga, je preporučljivo iznad ovog broja učitanih podataka koristiti AMF tehnologiju. Na dijagramu se vidi, zavisnost vremena od broja učitanih podataka, kod JSON-a i XML-a je znatno veća u odnosu na AMF. Kriva koja opisuje AMF tehnologije, treća linija, je znatno manjeg nagiba, dok kod JSON-a, prva linija, je znatno strmija, što znači da vreme brže raste, sa povećanjem broja učitanih podataka. Srednja linija predstavlja XML format, gde se vidi da XML mnogo više zavisi od JSON-a kada se povećava broj prenesenih podataka.



Slika 5.8. Komparacijski dijagram brzine učitavanja podataka za AMF, JSON i XML tehnologiju, za opseg od 100 do 2000 podataka, sa korakom 100.

AMF-a ne radi sa HTTP konekcijom. Prednost AMF-a je otvaranje soket konekcije, kroz koju se klijentu prosleđuju binarni podaci. Kod Flex-a, u AMF tehnologiji, koristi se RemoteObject klasa za uspostavljanje veze sa serverom, koja nasleđuje Proxy klasu. AMF obavlja asinhronu komunikaciju, gde se podaci šalju i primaju bez dogovorenog tajminga slanja i primanja podataka.

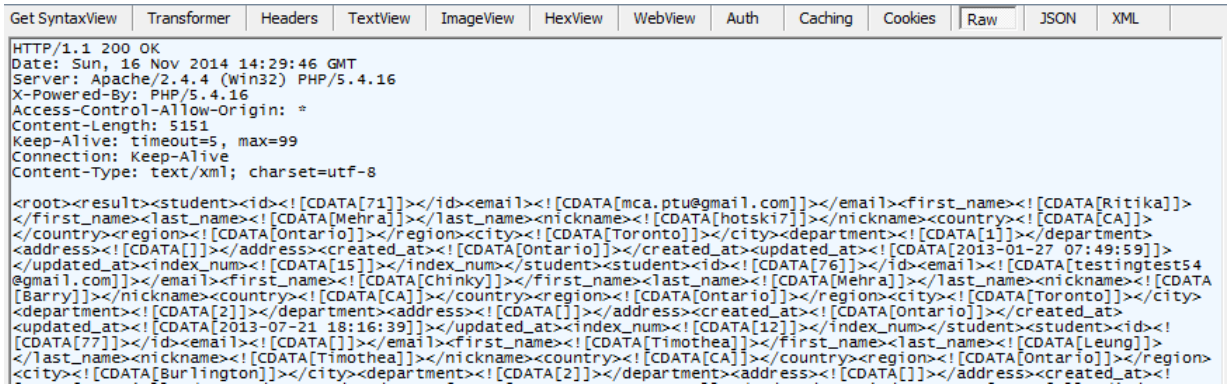
Kod AMF-a vreme transfera podataka ne zavisi od količine podataka, jer je binarni prenos, i uvek je praktično isto. Kod JSON-a i XML-a, vreme konekcije zavisi od količine podataka, podaci se pakuju u tekstalnom fajlu, i vreme izvršenja konekcije zavisi od veličine

fajla. Prema tome, što ima više podataka, vreme konekcije se povećava kod JSON-a i XML-a, s tim što je vreme konekcije kod XML-a veće od JSON-a za istu količinu podataka, zbog same strukture XML-a, što rezultuje veću veličinu izlaznog fajla, a time i veće vreme prenosa podataka.

Na slici 5.9. prikazan je sadržaj JSON fajla, kada se učitava 10 podataka sa servera u Flex aplikaciju.

Slika 5.9. Struktura JSON fajla, koji je učitan u klijentsku Flex aplikaciju.

Na slici 5.10prikazan je sadržaj XML fajla, kada se učitava 10 podataka sa servera u klijentsku aplikaciju. Upoređujući isti sadržaj, na slici 5.9. i 5.10, može se zaključiti da je XML fajl kompleksniji od JSON fajla, većeg je kapaciteta, i ako su oba čitljiva. Pa stoga, za prenos XML fajla, zbog većeg kapaciteta, potrebno je više vremena, zbog kompleksnosti same strukture XML-a, na strani servera i klijenta, koriste se složenije klase za obradu ovakve strukture u odnosu na JSON i AMF sadržaj, a sve to povećava vreme serijalizacije i deserijalizacije.



Slika 5.10. Struktura XML fajla, koji je učitan u klijentsku Flex aplikaciju

Prema tome, i ako nosi istu količinu podataka, zbog same strukture XML-a, XML fajl je veći od JSON-a. U tabeli 1, dati su podaci o veličini izlaznog fajla u zavisnosti od broja učitanih podataka iz baze, za JSON i XML fajl. Podaci su mereni pri učitavanju u aplikaciju.

Tabela 5.1.: Lista vrednosti za veličinu fajla u JSON i XML formatu, za N učitanih podataka

N	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Veličina JSON fajla (Kb)	2,73	5,41	8	10,448	13,05	15,7	18,36	21,16	23,97	26,43
Veličina XML fajla (Kb)	5.03	9,97	14,746	19,36	24,196	29,03	33,886	38,89	43,9	48,538
JSON fajl je veći od XML fajla za N%	84,2%	84,2%	84,3%	85,3%	85,4%	84,9%	84,56%	83,8%	83,14%	83,6%

Na osnovu uzorka iz tabele 1, od 10 do 100 učitanih podataka, razlika u veličini fajla između JSON i XML fajla je približno 84%. Prema tome, XML fajl je za 84% veći od JSON fajla, stoga je i kraći prenos iste količine podataka u JSON formatu.

Na strani servera i klijenta, struktura XML podataka je kompleksnija, pa je takvu strukturu teže obraditi, jer ima više nodova. Dakle, ako se ima kompleksna struktura objekta, kao što je XML, nju je u bilo kojem jeziku teže obraditi, zato što se mora proći kroz više nodova, kroz veću dubinu. Prema tome, kada je u pitanju serijalizacija i deserijalizacija, ne postoji razlika između jezika. Svaki jezik će obraditi brže, manje kompleksnu strukturu podataka. U ovom modelu, php-u, je lakše obraditi JSON objekte od XML objekata. Funkcije PHP-a za parsiranje XML objekta su kompleksnije, nego kada se obrađuje jednostavniji objekat.

Na brzinu transfera podataka utiče vreme serijalizacije i deserijalizacije, za bilo koju tehnologiju, što ima više podataka, vreme serijalizacije i deserijalizacije je veće. Serijalizacija je for petlja, koja prolazi, kroz ogroman broj objekata, pretvarajući ih u neku tehnologiju. Za sve tri tehnologije, postupak je isti. Za JSON, radi se upit u bazu, onda se odgovor iz baze pretvara u JSON, šalje se klijentu. Klijent čita JSON, pretvara ga u objekte, i dalje se izvršava komanda[11,12,16,17]. Isto tako se radi za XML i AMF.

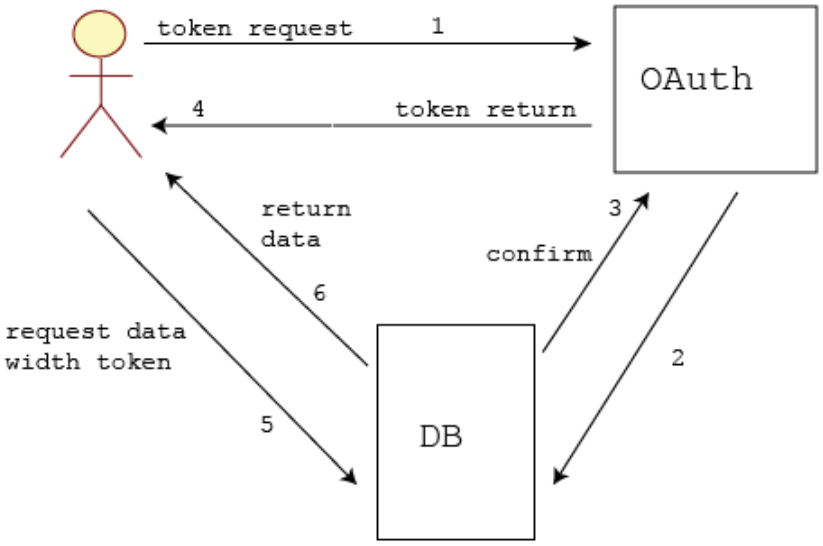
Vreme serijalizacije na strani servera zavisi od broja objekata, i ako ima više objekata koje treba serijalizovati, onda se troši više vremena, što je posebno izraženo kod JSON-a i XML-a, jer je složenost veća. Takođe što ima više podataka onda je fajl u JSON-u i XML-u teži i više vremena je potrebno za transport podataka.

Dakle vremenska razlika se najviše ogleda u vremenu transporta podataka kod AMF-a, JSON-a i XML-a i raste sa veličinom fajla.

Na 100 podataka, kod AMF-a, prosečno vreme učitavanja, iznosi 0,21s, a na 1000 podataka 0,31s. Kod JSON-a, na 100 podataka je 0,08s a na 1000 je 0,30s, razlika je 3,75 puta. To znači da transport podataka ima veliku ulogu.

Različiti resursi, koji se koriste u aplikaciji, mogu uticati na izbor načina pakovanja podataka između servera i klijenta. Na primer, ako radna aplikacija, koristi više servisa, koji koriste XML, a potrebno je da se neki servis napiše, onda je logično da se izabere XML, da bi se imao svuda isti način pripreme i obrade podataka, da bi se koristile iste klase u celokupnoj aplikaciji, radi bržeg pisanja koda i konzistentnosti koda. Prema tome, ako se koristi više izvora podataka, a na programeru je da napravi jedan izvor. Ako svi oni koriste jednu tehnologiju, na primer XML, onda je odluka na programeru da odabere jezik u onom delu koji obrađuje, a to je XML, jer će se na taj način prilagoditi servis koji se pravi sa drugim servisima, kako bi se iskoristio deo klasa iz drugih servisa, i omogućilo da se dva objekta saberu. Jer ako je jedan objekat u XML-u a drugi u JSON-u, onda se ta dva objekta ne mogu sabrati. U ovom slučaju, morao bi se prevesti jedan i drugi objekat u čitljivi deo, kako bi se spojili, što bi usporilo učitavanje podataka. Ali ako su oba objekta u XML-u ili u JSON-u, onda mogu se spojiti, jer su oni prirodno spojivi, postoje metode za njih, i to je jedna od prednosti koje se mogu koristiti. Kada je u pitanju bezbednost podataka, može se koristiti SSL ili enkriptovan način komunikacije. HTTP protokol nije bezbedan, kada se koristi tehnologija prenosa JSON, XML ili web servisi. Protokol HTTP može se koristiti, gde sigurnost podataka nije toliko bitna, jer je sadržaj JSON i XML fajla čitljiv. SSL je protokol za enkripciju saobraćaja. Ovde se saobraćaj između tačke A i B enkriptuje, kako se ne bi video sadržaj koji se šalje. Drugi način je autorizacija na servisima, koja se može napraviti ili kupiti i iskoristiti. Autorizacija je funkcija, navodeći prava pristupa nekim resursima, povezana sa zaštitom podataka. Za autorizaciju može se koristiti standard OAuth, koji spada u open standard za autorizaciju. OAuth pruža klijentskoj aplikaciji siguran dodeljen pristup resursima servera u ime vlasnika resursa. To znači, klijent pošalje serveru zahtev, da li može da vidi njegove podatke. Server odgovara tako, što daje potvrdan odgovor, u smislu broja ili zaštitne šifre, koju će korisnik koristiti kroz svaki naredni upit. Taj broj u svakom upitu, govori severu kao vlasniku podataka,

da je server dozvolio pristup podacima klijentu. I svaki put klijent dobija drugu šifru kada se loguje. Generalno, klijent se registruje na serveru kao neko ko želi da pristupa serverskim podacima. Server najčešće daje klijentu tri podatka. Prvi podatak je broj ID aplikacije. Dakle, autorska aplikacija se registruje na serveru, kao vlasniku podataka, i zahteva od korisnika da aplikacija pristupa određenim podacima, tada, server dodeljuje ID autorskoj aplikaciji, i u bazi podataka zapiše se ID aplikacije, i kao vlasniku aplikacije, klijentu se izdaju još dva podatka, to je username i password. Sada klijent ima tri podatka, jedan je ID aplikacije, drugi je username i treći je password. Ova tri podatka se ne koriste da bi se pristupilo podacima, već se poziva drugi servis, koji ne mora da bude na istom severu, gde su podaci. Klijent poziva servis, prosleđujući mu tri podatka, pri čemu sledi provera podataka, da li je ID aplikacije, za taj username tačan i da li je šifra tačna, ako to jeste, servis generiše jedan slučajan broj, koji se vraća klijentu, kao odgovor autorizacije, a u isto vreme u bazu podataka, kod svakog podatka upiše se kljientov autorizacioni broj, taj isti, ili se stavlja u neku tabelu autorizacije. Na slici 5.11. Šematski je prikazana autorizacija.



Slika 5.11. Pristup bazi podataka, sa autorizacijom

Linijom 1, korisnik zahteva ključ od sistema OAuth. Ovaj sistem generiše ključ i linijom 2 ga, upisuje u bazu podataka DB. Baza DB linijom 3, daje potvrdni odgovor, da je upisan ključ. Linijom 4, ključ se šalje klijentu, koji ga linijom 5 prosleđuje bazi DB, prilikom zahteva podataka. Linijom 6, podaci se dostavljaju klijentu.

Najbolje je da se koriste oba načina za zaštitu podataka, autorizacija i enkripcija, servisom SSL. Postoje dva načina autorizacije: vremenski ograničen i vremenski neograničen način autorizacije. Kod vremenski ograničene autorizacije, servis za autorizaciju, je izdao klijentu

dozvolu za pristup podacima na serveru, ograničen vremenski period. Posle isteka vremena, ukida se dozvola klijentu. Ili se dobija ključ za pristup podacima, za određen vremenski period, koji se meri od zadnjeg pristupa podacima na serveru. To se najčešće radi, i to se na webu zove sesija. To znači, dok je klijent aktivan, dobija neograničen vremenski ključ. Vreme se meri od zahteva za podacima, kada se pošalje novi zahtev, resetuje se vreme. Kada istekne vremenski period, od zadnjeg zahteva, sistem briše ključ iz tabele autorizacije. Ovakve stvari koristi facebook.

Stvari za autorizaciju je vezan za http request, za JSON, XML i Web servise, jer su Web servisi praktično XML. Prvi poziv ka serveru mora da bude OAuth, zahtev za autorizaciju.

6. ZAKLJUČAK I DALJA ISTRAŽIVANJA

U Disertaciji kreirana je klasična web radna aplikaciju koja koristi php i mysql za crud (create, read, update, delete) operacije, sa html i css korisničkim interfejsom. Napravljena je RIA (web2) aplikacija analogna klasičnoj web aplikaciji, ali sa FLEX korisničkim interfejsom, koja takođe izvršava CRUD operacije nad bazom podataka, sa sledećim varijacijama:

- XML REST servisi za komunikaciju klijent-server
- JSON REST servisi za komunikaciju klijent –server
- AMF RPC za komunikaciju klijent –server

U svim slučajevima testirana je brzina izvršenja crud operacija nad test bazom sa nekoliko stotina slogova. Aplikacija je implementirana u PHP/MySQL okruženju, otvorenog je koda i dostupna sa svakog mesta u svako vreme.

Analizom programskih jezika koji se koriste na aktivnim serverima, sve je jasnije da PHP programski jezik uzima primat u radu na serverskoj strani. Njegova snaga se ogleda ne samo u ceni i pristupačnosti, nego sve više u tome da se on razvio u jedan moćan i moderan objektno orijentisan jezik. Jednostavnost u pisanju koda kao i pokrivenost skoro svih područja u pisanju aplikacija i Web stranica uopšte, navodi nas na tezu da je primenom PHP-a moguće izraditi potpuno funkcionalnu web poslovnu aplikaciju koja po karakteristikama grafičkog korisničkog interfejsa ne zaostaje za desktop aplikacijama iste klase, a omogućuje sve prednosti web zasnovanih aplikacija.

Grafički korisnički interfejs radne aplikacije izrađen u Fleks tehnologiji je ne samo iste, nego i veće funkcionalnosti od korisničkog interfejsa desktop aplikacije. U Fleks tehnologiji, upotrebom ugrađenih komponenti je veoma jednostavno i brzo napraviti interfejs a zatim osnovnim poznavanjem ActionScript3 (AS3) jezika učiniti ga funkcionalnim. Pored prednosti manipulacije podacima na razne načine, ovakav korisnički interfejs omogućava i štampanje na računaru svakog korisnika, a to mu omogućavaju ugrađene funkcije za komunikaciju sa sistemom. Aplikacija se lako održava i menja po potrebi, a mogućnosti dizajna aplikacije su daleko veće od mogućnosti koje pruža HTML.

Realizacijom rešenja zadatog problema na osnovu iskustva smo dokazali da je uz upotrebu PHP programskog jezika moguće izraditi Web zasnovanu integrisanu aplikaciju elektronskog poslovanja koja je u potpunosti funkcionalna, slika 4.1.5.

Dokazali smo da je uz upotrebu PHP jezika moguće napraviti desktop Web zasnovanu aplikaciju za elektronsko poslovanje, a dodavanjem Adobe Flex-a za izradu korisničkog interfejsa i Web2.0 aplikaciju čije su performanse bolje od klasičnih desktop aplikacija.

Autor je primenom navedenih tehnologija realizovao i web2.0 aplikaciju za upravljanje proizvodnjom, knjigovodstvom i menadžmentom informacionih sistema, kao najbitnijim aplikacijama potrebnim za funkcionisanje jednog preduzeća.

Rezultati testiranja su predstavljeni na slici 5.8. Na osnovu izvršenih merenja prenosa podataka između klijenta i servera, u radnoj aplikaciji, ustanovljeno je da do 2000 podataka, preporuka je koristiti JSON tehnologiju za prenos podataka.

Za kompanijske aplikacije, gde se prenosi veliki broj podataka preporuka je koristiti AMF tehnologiju prenosa podataka, jer je znatno brža od JSON i XML tehnologije.

Ako se vrši segmentacija podataka, pri prenosu velike količine podataka, a blok nije veći od 2000 podataka, onda je preporuka da se koristi JSON tehnologija.

Ako radna aplikacija, koristi više izvora, koji pripremaju podatake u XML, a potrebno je da se neki servis napiše, onda je preporuka izabrati XML, radi konzistentnosti i bržeg pisanja koda, radi korištenja iste klase u celokupnoj aplikaciji, da bi se imao svuda isti način pripreme i obrade podataka, i omogućilo da se dva objekta saberu.

Kada je u pitanju kompleksnost koda na strani servera i klijenta, preporuka je koristitit AMF, zatim JSON tehnologiju. XML ima složeniju strukturu a JSON struktura više odgovara objektnoj strukturi, pa je obrada XML fajla složenija.

Kada je u pitanju veličina izlaznog JSON i XML fajla, rezultati testiranja su predstavljeni u Tabeli 4.1.1. Na osnovu rezultata testiranja potvrdili smo da bolje performanse ima JSON fajl, i preporuka je koristiti JSON tekstualni fajl, jer je prema izvršenim merenjima za 84% manji od XML fajla.

Utvrđeno je, na osnovu komparacijskog dijagrama, slika 5.8., kriva zavisnosti vremena od količine prenetih podataka, najsporije raste kod AMF-a a najstrmije kod XML.

Kada je u pitanju sigurnosni aspekt za XML i JSON tehnologiju, preporuka je da se koristi XML, JSON i Web servis HTTP prenos ako bezbednost podataka nije bitna. Ako se zahteva da prenos podataka bude bezbedniji, preporuka je koristiti HTTPS protokol za prenos.

Radi bezbednosti prenosa podataka, za tehnologije AMF, XML, JSON, Web servis, preporuka je koristiti AMF tehnologiju, jer AMF podaci nisu čitljivi.

Radna aplikacija je modelovana u UML-u. UML je jednako dobar za opis desktop, web, web2.0, i iznad web2.0 aplikacija.

Za analizu opterećenosti radne memorije korišćen je alat Profile as, koji daje široke mogućnosti, da se sa različitih aspekata izvrši pregled zauzetosti radne memorije, daje jasnu sliku autoru aplikacije kako da optimizuje kod, i na taj način poveća brzinu rada aplikacije, i onemogući curenje memorije.

Tokom izrade aplikacije preporuka je koristiti alat Profile as, kako bi se na vreme optimizovao kod, i nebi došlo do curenja memorije. U aplikaciji je utvrđeno da grafičke komponente zauzimaju dosta memorije i procesorskog vremena, preporuka je skin klase svesti na minimum.

Desktop i Web aplikacije se razlikuju po izgledu i funkcionalnosti. Desktop aplikacija koristi sve resurse računara, dok Web aplikacija koriste resurse browser-a i resurse servera za proračune i obradu podataka, prikaz podataka korisniku ograničen je samo na browser. Browser je desktop aplikacija na kojoj se prikazuje sadržaj sa nekog udaljenog servera. Na serveru može da bude bilo kakva aplikacija, a browser je ograničen veličinom interne memorije, u koju se može učitati limitirana količina podataka. Prvi browser-i su imali memorijske limite. Danas browser-i predstavljaju kompleksne aplikacije, sa ugrađenom komunikacijom sa računarskim sistemom, kroz razne plug-in-ove, i mogu da koriste resurse računara, kao što je grafičko ubrzanje, što omogućuje prikaz 3D sadržaja na browser-u, jer on koristi grafičku akseleraciju sa grafičke kartice direktno. OpenGL je tehnologija koja dozvoljava da se mogu koristiti 3D efekti za prikaz na Web-u, preko browser-a. OpenGL direktno se oslanja na procesor sa grafičke kartice.

Kada je u pitanju Web 1.0, Web 2.0 i doba posle Web 2.0, najbitnija osobina Wb 2.0 je iskoristljivost, omogućava interakciju između korisnika i Web-a, uvodi alate poput bloga, wikis-a, tagove, widget-e i RSS gde se nalaze podaci u bilo kakvoj formi[4]. Web 1.0 je prezentacionog karaktera, gde se vrši prezentacija firmi, korisnik je mogao samo da čita na webu, zvao se i Read. Web 2.0 je bio bogatiji izgledom u poređenju sa Web 1.0, sa znatno boljim korisničkim interfejsom. Tehnologija JavaScript-a je omogućila interaktivnost[5], asinhroni prenos podataka je ubrzao komunuikaciju između korisnika i servera, gde se pojedini delovi aplikacije izvršavaju paralelno, bez usaglašavanja vremenskih tajminga. Na ovaj način, je značajno poboljšana raspodela posla, kao i mogućnost da se blok koda može raščlaniti i dodeliti različitim nitima na izvršavaje. Kraj Web2.0 se smatra 2010-om godinom, a od te godine je period posle Web 2.0.

Period posle Web 2.0, na Web-u karakteriše personalizacija, semantički web, silosi[4,8]. Ovo doba još nije definisano, nema ime. Personalizacija je prilagođavanje interneta jednoj osobi. Danas za personalizaciju se koristi popularni servis Cloud computing, a računar služi kao pristupna tačka na internetu.

Kada korisnik traži neki podataka, ne traži ga na nekom serveru, već u servisu Cloud computing, a nije mu bitno na kom server je on smešten.

Druga karakteristika ovog doba, je semantička pretraga na web-u. Semantički web predstavlja mogućnost softvera da prepozna potrebui ne samo što se traži po ključnoj reči, traži se koren ključne reči.

Takođe, ovaj period karakteriše i lokalizovana pretraga. Period posle Web 2.0 karakterišu i mikroformati, silosi.

Kada se analizira MVC arhitektura i zadatak kontrolera, imajući u vidu autorsku aplikaciju, MVC struktura se uvek nalazi samo na jednoj strani, nije raspodeljena između klijenta i servera, sastoji se iz tri dela M, V i C, ali sve zajedno čini klijenta ili servera.

MVC pristup se može istovremeno koristiti i na serveru i na klijentu, kao što je slučaj sa autorskom aplikacijom, gde je na strani Java servera korišćen MVC play Framework.

M je model, radi sa podacima, može biti baza podataka ili web servis. Kontroler može biti na strani klijenta i na strani servera, izrađen u različitim ili istim tehnologijama, kao što je slučaj sa autorskom aplikacijom. On kontroliše, izdaje naredbe, vodi računa da podaci budu nađeni i prikazani u view.

U autorskoj aplikaciji, kontroler ima ograničenu ulogu, služi za trigerovanje događaja, slanje naredbi i obavlja neke logičke pojave, dok model priprema podatke, obrađuje podatke koju će view da prikaže, pristupa serveru a server mu vraća podatke. Autorska aplikacija predstavlja RIA aplikaciju, koja ima servisni i klijentski deo. Na klijentskom delu je korišćen MVC, i na serverskom delu je korišćen MVC ali kada je korišćena aplikacija u Javi.

Kada je pitanje, da li obradu podataka vršiti na serveru ili klijentu, preporuka je, da sve komplikovane operacije treba vršiti na serveru, zato što su serveri po pravilu mnogo jači računari od klijentskih računara. Softver koji se pravi, ne pravi se za klijentov računar, jer se nezna kakav klijent ima računar. Ono o čemu treba voditi računa, da softver koji se pravi nema veliku potrošnju memorije. Prednost RIA aplikacija, je što se na serveru nalaze podaci, i tamo se može vršiti složena obrada. Na klijentu obrada podataka treba da bude svedena na minimum, kao što je priprema podataka za prikaz korisniku, a ne neke velike računske operacije.

Po pitanju izbora desktop, Web ili Web2.0 odnosno Web koji predstavlja doba iznad Web 2.0, ne postoji oštra granica, što zavisi i od zahteva klijenta. Ako se pravi aplikacija koja je zahtevna i kompleksna, onda se preporučuje da bude desktop aplikacija, jer će potpuno koristiti resurse sistema. Međutim, ako se radi neki okvir, gde će se prikazati neki sadržaj sa servera, onda je dovoljno da bude Web aplikacija. I desktop aplikacije se dele na dve vrste. Postoje desktop aplikacije koje su potpuno nezavisne od interneta, i RIA desktop aplikacije, čiji se kontejner

nalazi na desktop, ali informacije i sadržaj se nalaze negde na internetu ili intranetu, što znači da nije na mašini, već negde na serveru. Kod Web aplikacije uvek je sadržaj negde na internetu ali i Web2.0 aplikacije su RIA aplikacije po svojoj definiciji. Rich ne znači da je bogata izgledom apikacija, već zato što omogućuje interakciju sa korisnikom.

Naučni značaj predstavljenog rada ogleda se u pristupu navedenim problemima koji do sada nisu proučavani na ovakav način, dok se praktični značaj ogleda u mogućnostima primene rezultata istraživanja u vidu Wampserver2.4-x86sistema skladištenja na unapređenje rešavanja problema koji se bave sličnom tematikom. Optimizacijom koda, optimalnim izborom broja skin klasa, može se uticati na smanjenje CPU vremena izvršavanja operacija u aplikaciji.

Na identičan način mogli bi se realizovati i data servisi, to jest direktna i neposredna komunikacija između zaposlenih u preduzeću, zatim video i audio konferencije u realnom vremenu kao i dodavanje novih grafičkih elemenata koji bi po mogućnosti pojednostavili upotrebu aplikacije i tako je približili većem broju korisnika samim tim olakšavajući i skraćujući obuku korisnika.

7. LITERATURA

[1] Using ADOBE FLEX 4.6 - Tutorial from official web site http://help.adobe.com/ (visited 19.11.2014.)

[2] Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, Perspectives on web services: applying SOAP, WSDL and UDDI to real-world, Springer, 2003.

[3] Cliff Hall, ActionScript Developer's Guide toPureMVC, ISBN-13: 978-1449314569, O'Reilly Media; 1 edition, 2011.

[4] Paul Anderson: Web 2.0 and Beyond: Principles and Technologies, ISBN-13: 978-1439828670, 2012.

[5] Robin Nixon: Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic, ISBN-13: 978-1491949467, 2014.

[6] Matt Zandstra: PHP Objects, Patterns, and Practice Paperback, Edition: 4th, ISBN-13: 978-1430260318, 2013.

[7] Jennifer Niederst Robbins: HTML5 Pocket Reference (Pocket Reference (O'Reilly)) Paperback, ISBN-13: 978-1449363352, Edition: Fifth Edition, 2013.

[8] Tom Funk, Web 2.0 and Beyond: Understanding the New Online Business Models, Trends, and Technologies, ISBN-13: 978-0313351877, 2008.

[9] C++ FAQ, Kept by Marshall Cline, retrieved from http://www.parashift.com/c++-faq-lite/index.html (visited 19.11.2014.)

[10] Encryption Basics - EFF Surveillance Self-Defense Project, Surveillance Self-Defense Project, n.d. Web. 06 Nov. 2013. <https://ssd.eff.org/tech/encryption (visited 19.11.2014.)

[11] Florescu, D, Fourny, G, JSONiq: The History of a Query Language, IEEE INTERNET COMPUTING, Volume: 17 Issue: 5 Pages: 86-90, 2013.

[12] Pettit, JB, Marioni, JC, bioWeb3D: an online webGL 3D data visualisation tool, BMC BIOINFORMATICS, Volume: 14, Article Number: 185, DOI: 10.1186/1471-2105-14-185, 2013.

[13] Jorstad, I, Bakken, E, Johansen, TA, Performance evaluation of JSON and XML for data exchange in mobile services, WINSYS 2008: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON WIRELESS INFORMATION NETWORKS AND SYSTEMS, Pages: 237-240, 2008.

[14] Rodrigues, C, Afonso, J, Tome, P, Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML, ENTERPRISE INFORMATION SYSTEMS, PT 2, Communications in Computer and Information Science, Volume: 220, Pages: 162-169, 2011.

[15] Adamanskiy, A, Denisov, A, EJDB - Embedded JSON database engine, 2013 FOURTH WORLD CONGRESS ON SOFTWARE ENGINEERING (WCSE), Pages: 161-164., DOI: 10.1109/WCSE.2013.29, 2013.

[16] Julian Merelo-Guervos, Juan; Castillo, Pedro A.; Laredo, J. L. J.; et al., Asynchronous Distributed Genetic Algorithms with Javascript and JSON, Conference: IEEE Congress on Evolutionary Computation Location: Hong Kong, 2008 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-8 Book Series: IEEE Congress on Evolutionary Computation Pages: 1372-1379 Published: 2008.

[17] Ye Jun; Li Zhishu; Ma Yanyan, JSON Based Decentralized SSO Security Architecture in E-Commerce, International Symposium on Electronic Commerce and Security Location: Guangzhou, PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON ELECTRONIC COMMERCE AND SECURITY Pages:471-475 Published: 2008

[18] Jean-Jacques Dubray, "Composite Software Construction" Understanding SOA in the Context of a Programming Model, C4Media Inc, 2007.

[19] Alonso G., Casati F., Kuno H., Machiraju V. , Web services: concepts, architectures and applications, Springer, 2004.

[20] IBM Corporation, "Patterns: Service-Oriented Architecture and Web Services", IBM RedBooks (April 2004), ftp://www.redbooks.ibm.com/redbooks/SG246303/

[21] Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, Perspectives on web services: applying SOAP, WSDL and UDDI to real-world, Springer, 2003.

[22] Bill Evjen, Kent Sharkey, Thiru Thangarathinam, Michael Kay, Alessandro Verne, Professional XML, Wrox/Wiley Pub., 2007

[23] Michael P. Papazoglou, M. Papazoglou, Web services: principles and technology, Pearson Prentice Hall, 2008.

[24] Elisa Bertino, Lorenzo Martino, Federica Paci, Anna Squicciarini, Security for Web Services and Service-Oriented Architectures, Springer, 2010.

[25] Juric M., "Business Process Execution Language for Web Services Second Edition", Packt Publishing, 2006.

[26] D. Chappell, Tyler Jewell "Java Web Services", O'Reilly,2002.

[27] Atherton., R., W., "Moving Java to the Factory", IEEE Spectrum, December 1998.

[28] Benis., W., Mische., M., "The 21st Century Organization -Reinventig Through Reeingineering, Jossey-Bass Publishers, San Francisco, 1997.

[29] Charles E. Brown, "The Essential Guide to Flex 2 with ActionScript 3.0", Friends of ED, Apress Company, USA

[30] Davidow W. H., Malone M. S. "The Virtual Corporation -Structuring and Revitalizing the Corporation for the 21ST Century", Harper Business, New York, 1992

[31] Grupa autora, Programiranje na jeziku c#, O'REILLY, 2006.

[32] Grupa autora, "Getting Started with Flex 2, Adobe Systems Incorporated", San Jose, CA, USA, 2006.

[33] Grupa autora, "Adobe® Flex® 3 Developer Guide", Adobe Systems Incorporated, San Jose, CA, USA, 2008.

[34] Hammer. M., Champy. J, "Reengineering the Corporation: a Manifesto for Busines Revolution", Harper Business, New York, 1993.

[35] Hammer. M., Stanton. S., "The Reengineering Revolution", Harper Business, New York, 1995.

[36] Johnston, W., B., Packer, A., H., "Workforce 2000", Hudson Institute, Indianopolis, 1987.

[37] Kazoun., C.,Lott., J., "Programming Flex2", O'Reilly, USA, 2008

[38] Laudon., K., C., Laudon., J., P., "Information Systems and the Internet -A Problem Solving Approuch", The Dryden Press, USA, 1998.

[39] Lovreković., Z., "Informatičke tehnologije-ključ za reinženjering", I savetovanje "Reinženjering proizvodno-poslovnih procesa", zbornik radova, Zobnatica, Mart 2000.

[40] Lovreković Z., Djurica M., "Information Systems for Production Management: Does the Educational System Meet the Changing Requirements of the Yugoslav Industry?", Proceedings, Portland International Conference on Management of Engineering and Technology –PICMET03, Portland, Oregon, SAD, 21-24 July, 2003.

[41] Lovreković, Z., "Virtuelna zajednica za upravljanje znanjem-jedno praktično iskustvo", Međunarodna naučna konferencija "Na putu ka dobu znanja", Split, 2009.

[42] Lovreković Z., Pavlović M., Marković, N. "Merenje nivoa informatizovanosti firme", Konferencija sa medjunarodnim učešćem, SII2002, Vrnjačka Banja, 2002.

[43] Lovreković Z., "Razvoj klijent-server aplikacije otvorenog koda kao sistema za prikupljanje, delenje i korištenje znanja za podršku mentorima u radu sa mentorskim grupama na dodiplomskim studijama", zbornik radova, 3.Savetovanje "Na putu ka dobu znanja", Fakultet za menadžment, Novi Sad, 2005.

[44] Lovreković, Z., Ristić, I., "Virtuelne zajednice kao sveobuhvatni sistem upravljanja znanjem", 6.Savetovanje "Na putu ka dobu znanja", Fakultet za menadžment, Sremski Karlovci, 2008.

[45] Lovreković, Z., "Internet programiranje", Udruženje građana"Inicijativa za upravljanje znanjem", Kać, 2009.

[46] Mihajlović., D., "Informacionim sistemi i projektovanje baza podataka", Univerzitet u Novom Sadu, Novi Sad, 1998.

[47] Ristić D., "Preoblikovanje organizacija", I savetovanje "Reinženjering proizvodno-poslovnih procesa", zbornik radova, Zobnatica, Mart 2000.

[48] Safo, P., "Znanje i intuicija u internet-ekonomiji", Ekonomska politika, br. 2508, 15. Maj 2000.

[49] Simple Object Access Protocol,http://www.w3.org/TR/soap/

[50] Web Services Description Language (WSDL) Version 1.1, W3C Note, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[51] Dušan Malbaški: Objekti i objektno programiranje kroz programske jezike C++ i pascal, 2006.

[52] Zoran Lovreković, Ćamil Sukić, Šabanović M. (2010), Desktop, web and web2.0 application comparing on the aspect of the realisation possibility the complex user interface, Technics Technologies Education Menagement, Vol. 5, No. 4, Science Citation SCIE.

[53] Sabanovic M., Saracevic M., Azizovic E. (2016), Comparative analysis of technologies for data transfer between the server and the client: AMF, JSON and XML, Periodicals of Engineering and Natural Sciences, ISSN: 2303-4521, Vol.4, No.2, DOI: 10.21533/pen.v4i2.57.g76.

[54] Sabanovic M., Saracevic M., Azizovic E. (2017), One software solution for data transfer between client and server with emphasis on saving memory and CPU usage, International Journal of Industrial Engineering and Management (IJIEM), ISSN: ISSN: 2217-2661, Indexed in Scopus (Elsevier).

[55] J. Meloni, *PHP 5,* Thomson Course Techology, Boston, MA, 2004.

[56] Ajit Kumar: Sencha MVC Architecture Paperback , 2012.

[57] Amuthan G: Spring MVC Beginner's Guide, 2014.

[58] Sabanovic M., Saracevic M., Dzemic K., Comparative analysis of Data transfer technology between Server and Client: AMF and objects on the website (in review).

INTERNET IZVORI:

[59] http://www.tricedesigns.com/2011/11/07/AMF-vs-JSON-in-air-mobile-applications/ (visited 25.11.2014)

[60] http://www.jamesward.com/2007/04/30/ajax-and-flex-data-loading-benchmarks/ (visited 25.11.2014)

[61] http://www.webopedia.com/TERM/T/TCP.html (visited 25.11.2014)

[62] http://sr.wikipedia.org/sr/TCP/IP(visited 14.12.2014)

[63] http://help.drenik.net/internet/arhitektura/index.htm(visited 14.12.2014)

[64] http://www.poslovniforum.hr/about02/net_http.asp(visited 14.12.2014)

[65] Universal Description, Discovery, and Integration,http://www.uddi.org(visited 14.12.2014)

[66] http://www.w3schools.com/webservices/default.asp(visited 14.12.2014)

[67] http://www.mysap.com(visited 14.12.2014)

[68] http://www.amfphp.org(visited 14.12.2014)

[69] http://www.w3.org(visited 14.12.2014)

[70] http://www.php.net(visited 14.12.2014)

[71] http://mysql.com(visited 14.12.2014)

[72] http://www.phpclasses.com(visited 14.12.2014)

[73] *Zend Technologies* http://www.zend.com(visited 14.12.2014)

[74] http://www.visible-form.com/blog/flex-amfphp-mapping-value-objects/(visited 14.12.2014)

[75] http://corlan.org/2008/10/10/flex-and-php-remoting-with-amfphp/(visited 14.12.2014)

[76] http://viconflex.blogspot.com/2007/04/mapping-vos-from-flex-to-php-using.html(visited 14.12.2014)

[77] http://www.mail-archive.com/flexcoders@yahoogroups.com/msg43982.html(visited 14.12.2014)

[78] http://www.adobe.com/devnet/flex/videotraining/xml/(visited 14.12.2014)

[79] http://www.json.org(visited 14.12.2014)

[80] http://www.onlamp.com/pub/a/onlamp/2007/07/19/introduction-to-flex-using-php.html(visited 14.12.2014)