# Implementacija MVC arhitekture u JSP tehnologiji

Jelena Ćosović, Miljan Radojičić
studenti drugog ciklusa studija
Elektrotehnički fakultet
Istočno Sarajevo, Bosna i Hercegovina
e-mail: jelena-cosa@hotmail.com, miljan.radojicic@yahoo.com

*Sadržaj*—**Na početku rada biće predstavljen WWW i klijent-server arhitektura koja je najzastupljenija arhitektura u Web aplikacijama. Na osnovu ove arhitekture izvršena je podjela Web tehnologija koje se izvršavaju na klijentskoj i na serverskoj strani. U radu će biti riječi o JSP tehnologiji i MVC arhitekturi kao i primjeni te arhitekture u JSP-u, tj. kako je izvršena podjela na model (upotrebom klasičnih Java klasa), kontroler (upotrebom servleta) i view (upotrebom standardnih JSP stranica). U radu će biti dati primjeri Web aplikacije koju su autori kreirali, a koja komunicira sa alumni bazom Elektrotehničkog fakulteta u I. Sarajevu i omogućava manipulisanje podacima koji se nalaze u toj bazi podataka.**

***Ključne riječi-JavaServer Pages, Java, Model-View-Controller, JDBC***

## I. UVOD

U današnje vrijeme Interneta i ubrzane modernizacije, sve više kompanija, organizacija i pojedinaca žele da povećaju svoj udio na tržištu i reklamiraju i popularizuju svoje usluge i proizvode. Kao najefikasnije rješenje nameće se izrada Web stranica upotrebom različitih tehnologija koje su danas dostupne. Svakako, na samom početku, potrebno je postaviti ciljeve, analizirati sve dostupne podatke, naći odgovarajuću strategiju, kako željeni rezultat na kraju ne bi izostao. Nakon ovih koraka, prelazi se na implementaciju, tj. izradu Internet prezentacije. Da bi se napravilo kvalitetno rješenje, mora se poznavati tehnologija i alati koji se koriste za izradu.

U ovom radu predstavljeni su tehnički aspekti izrade Web aplikacije, tj. Internet softvera za informacione sisteme upotrebom JSP-a (*eng*. Java Server Pages) kao jedne od najpopularnijih tehnologija za izradu Internet softvera.

## II. WWW I KLIJENT-SERVER ARHITEKTURA

### A. WWW

WWW (eng. World Wide Web) ili skraćeno Web predstavlja svjetsku mrežu međusobno povezanih dokumenata napisanih posebnim jezikom HTML (*eng*. HyperText Markup Language) koji omogućava povezivanje sa drugim elementima i uključivanje svih elemenata multimedije. Tim Berners-Lee uz pomoć Robert Cailliau-a napisao prijedlog za projekat nazvan "WorldWideWeb" (skraćeno W3) koji je predstavljao "hipertekstualni sistem" u kojem su jasno bili odvojeni dijelovi za pohranjivanje i dijelovi za prikazivanje informacija [1]. Pomoću klijentskih programa tj. pretraživača (*eng*. browsers), koji se mogu pokrenuti na različitim platformama, korisnici pristupaju podacima koji se nalaze na hipertekstualnim serverima.[1]

### B. Klijent-server arhitektura

Kao što se može vidjeti iz prethodnog, klijent-server arhitektura je usko vezana za Web još od samog početka razvoja. Klijent-server predstavlja vezu među procesima koji s pokreću na istim ili različitim mašinama koje su mrežno povezane.[2]

Klijent najčešće predstavlja Web pretraživač na lokalnom računaru. Pretraživač kontaktira Web server i šalje mu zahtjev za određeni resurs koji se nalazi na serveru. Pretraživač prima informaciju koja stiže kao odgovor od servera na zahtjev koji mu je prethodno klijent poslao, i pretraživač prikazuje informaciju na korisničkom računaru.[2] Serveri predstavljaju računare na kojima su pokrenuti procesi koji su odgovorni za obradu HTTP klijentskih zahtjeva i za slanje HTTP odgovora klijentima koji najčešće predstavljaju HTML stranice. Serveri su zaduženi i za obradu ne samo statičkog već i dinamičkog sadržaja (kreiranih pomoću JSP-a, PHP-a, ASP-a, ASP.NET-a i slično). [2]

## III. WEB TEHNOLOGIJE

Komunikacija na Internetu se može podijeliti na dva odvojena dijela: klijente i servere. Upravo ovakva podjela omogućila je da aplikacije budu sastavljene iz više komponenti. Ove komponente mogu da budu na različitim mašinama, a koje rade zajedno komunicirajući putem mreže (Interneta). U skladu sa klijent-server arhitekturom i razvijanje komponenti koje čine Web aplikaciju je podjeljeno na programiranje na klijentskoj strani i programiranje na serverskoj strani. [3]

Tehnologije koje su razvijene za programiranje na klijentskoj strani, tj. client-side tehnologije imaju jedan glavni cilj, a to je omogućavanje da se sadržaj Web stranica mijenja u skladu sa promjenama koje korisnik unosi. Dijelovi skripti se ugrađuju u HTML dokumente ili se oni pozivaju iz dokumenta i na taj način mogu da npr. procesiraju podatke koji se unose u forme, kako bi se provjerilo da li podaci koji su unijeti odgovaraju zadanim šablonima ili da se ponašanje određenih objekata unutar HTML dokumenta određuje skriptama koje reaguju na događaje koje generišu ti objekti. Najpoznatije tehnologije koje se koriste na klijentskoj strani su: JavaScript, Ajax (*eng*. Asynchronous JavaScript and XML) i jQuery kao posebna JavaScript biblioteka.[3]

Tehnologije za programiranje na serverskoj strani, tzv. server-side tehnologije uključuju skripte unutar HTML koda koje rezultuju obradom klijentskog zahtjeva od strane skripti koje se pokreću na serverskoj stani prije nego što server pošalje odgovor klijentu. Obrada na serverskoj strani obuhvata dinamičko dodavanje ili promjenu sadržaja Web stranica. Ove skripte se koriste za pristupanje podacima u bazi, za obradu podataka HTML formi, za uređivanje Web stranica kao i za zaštitu, pošto se serverski kod ne može vidjeti iz klijentskog pretraživača. Najzastupljenije server-side tehnologije danas su: PHP (*eng*. Hypertext Preprocessor), ASP (*eng.* Active Server Pages) i JSP (*eng*. Java Server Pages).[3]

## IV. JSP

JSP predstavlja jednu od najpopularnijih tehnologija koje se pokreću na serverskoj strani. . U JSP je predstavljena JSTL (*eng*. JSP Standard Tag Library) biblioteka koja pruža dodatne tagove koji su bili veoma korisni i doveli su do značajnog porasta interesovanja za ovu vrstu tehnologije. JSTL biblioteka je kolekcija tagova koji enkapsuliraju fukcionalnost mnogih standardnih JSP tehnologija i na taj način uklanjaju eventualna ponavljanja i kreiraju mnogo kompaktniju aplikaciju. Zajedno sa JSTL predstavljen je i JSP EL (*eng*. Expression Language), koji je služio za kreiranje prilagođenih komponenti. [4].

JSP je stekla popularnost uvođenjem određenih novina i alata koje su umnogome učinile jednostavnijim i efikasnijim proces izrade Web aplikacija. Prednosti koje JSP tehnologija pruža su [5]:

*a) Osobina da se jednom napiše, a svugdje pokreće* – JSP tehnologija je nazavisna od platforme, Web servera i osnovnih serverskih komponenti. JSP stranice se mogu pokrenuti na bilo kojoj platformi i bilo kom Web serveru i može im se pristupati sa bilo kog Web pretraživača.

*b) Visokokvalitetna podrška alatima* – Nezavisnost od platformi dozvoljava JSP korisnicima da izaberu nabolje alate. Takođe, jedan od ciljeva dizajna JSP-a je taj da se naprave visokokvalitetni prenosivi alati.

*c) Ponovna upotreba komponenti i biblioteka za tag-ove* – JSP tehnologija naglašava ponovnu upotrebu komponenti kao što su JavaBean komponente i biblioteke za tag-ove. Ove komponente se mogu koristiti sa interaktivnim alatima za razvoj komponenti i kompoziciju stranice, pružajući tako nezavisnost od platformi i fleksibilnost jezika.

*d) Odvajanje dinamičkog i statičkog sadržaja* – JSP tehnologija pruža mogućnost odvajanja statičkog sadržaja od dinamičkog sadržaja. Ovo u mnogome pojednostavljuje proces kreiranja sadržaja. Odvajanje podržavaju i tzv. bean komponente, koje su posebno dizajnirane za interakciju sa objektima na serverskoj strani i mehanizam za tag-ove.

*e) Podrška akcijama, izrazima i skriptovanju* – JSP tehnologija podržava i skriptne elemente i akcije. Akcije enkapsuliraju korisne funkcionalnosti u veoma pogodnu formu kojom alati mogu da manipulišu. Izrazi se koriste za pristupanje podacima. JSP 2.0 verzija je dodala i EL (eng. expression language) Java-baziranim skriptama. Izrazi u EL-u direktno izražavaju autorove koncepte kao svojstva u tzv. bean-ovima i pružaju više kontrolisan pristup podacima u aplikaciji. JSP 2.0 specifikacija je dodala i mehanizam kojim autori mogu pisati akcije koristeći direktno JSP tehnologiju. Ovo znatno pojednostavljuje proces pisanja akcija.

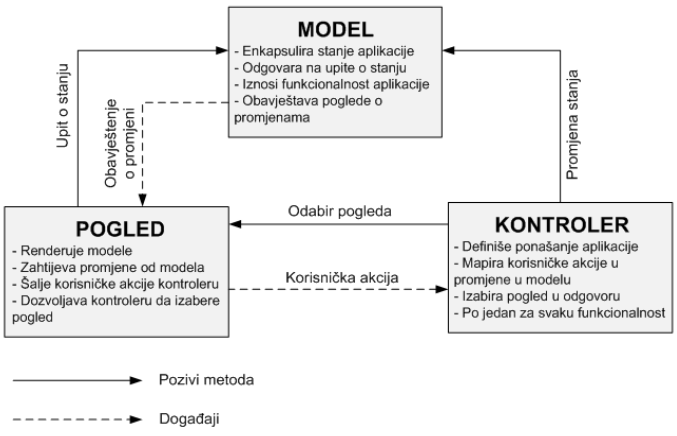
## V. MVC ARHITEKTURA

MVC (*eng*. Model-View-Controller) predstavlja softversku arhitekturu ili dizajn šablon koji se koristi u softverskom inženjerstvu a čiji je osnovni princip baziran na ideji da logika aplikacije treba biti razdvojena od njene prezentacije, tj. logika je podjeljena na tri odvojene jedinice: model, prezentacioni nivo (*eng*. view) i kontroler (*eng*. controller) koji komuniciraju jedni sa drugima.[6]

Model se bavi ponašanjem i podacima aplikacije, odgovara na zahtjeve za informacijama o svom stanje (koji najčešće šalje pogled) i odgovara na instrukcije za promjenom stanja (koje najčešće šalje kontroler). Model je zadužen za izvršavanje upita nad tabelama u bazi podataka i obavljanje računanja u zavisnosti od ulaznih promjenljivih.[7]

Pogled (*eng*. view) je zadužen za prikaz informacija koje je dobio od modela. Pogled ili prezentacioni sloj nije zadužen za obradu podataka koji se unose, već je jedino zadužen za renderovanje podataka koje dobije od modela i slanje podataka na odgovarajuće mjesto.[7]

Kontroler (*eng*. controller) je zadužen za interpretaciju ulaza koje korisnik unosi putem miša i tastature, na osnovu čega informiše model i/ili pogled da izvrše ažuriranje, ukoliko je to potrebno. Kontroler je zadužen za obradu zahtjeva koje šalje klijent, na osnovu kojih učitava ili ažurira prezentacioni nivo tj. poglede. Na slici 1. prikazana je komunikacija između dijelova MVC arhitekture.[7]




Slika 1. MVC arhitektura

## VI. IMPLEMENTACIJA MVC ARHITEKTURE U JSP TEHNOLOGIJI

Web aplikacija, koju su autori kreirali u praktičnom dijelu rada, predstavlja softver kreiran upotrebom JSP tehnologije koji omogućava komunikaciju sa alumni bazom Elektrotehničkog fakulteta u Istočnom Sarajevu. Korisniku aplikacije je omogućeno da manipuliše podacima u bazi, tj. da unosi nove zapise, mijenja postojeće zapise kao i da ima uvid u postojeće zapise u bazi podataka. Aplikacija je krerana

upotrebom MVC arhitekture, pa su u nastavku predstavljeni svi dijelovi aplikacije, prema MVC arhitekturi.

### A. Model

Model u okviru MVC arhitekture je predstavljen samim podacima koji se nalaze u bazi podataka i obavljanjem CRUD operacija nad tim podacima.

#### 1) JDBC API

Za pristupanje podacima u bazi podataka koristi se je JDBC (eng. Java DataBase Connectivity) API koji pruža Java aplikacijama pristup većini sistema baza podataka pomoću SQL-a (eng. Structured Query Language). Različiti DBMS (eng. Database Management Systems) imaju veoma malo zajedničkog: samo sličnu svrhu i većinom kompatabilan upitni jezik. Pored toga, svaka baza podataka ima svoj API koji se mora proučiti kako bi se mogli napisati programi koji komuniciraju sa tom bazom. [8]

JDBC predstavlja pokušaj kompanije Sun da se kreira platformski nezavisan interfejs između baza podataka i Java-e. Osnovne tri funkcije JDBC-a su [9]: ostvarivanje konekcije sa bazom podataka, slanje SQL komandi bazi podataka i obrada rezultata. Sa JDBC-om se pruža mogućnost korištenja standardnog seta metoda za pristup bazi. JDBC API definiše set interfejsa koji enkapsuliraju veći dio funkcionalnosti baze, uključujući izvršavanje upita, obradu rezultata i slično. Programeri pišu tzv. JDBC driver, koji predstavlja set klasa koje implementiraju ove interfejse za određeni DBMS. Aplikacija koristi JDBC za interakciju sa jednom ili više baza podataka, a da pritom ne posjeduje informacije o implementaciji driver-a [8].

#### 2) Povezivanje sa bazom podataka

Da bi se ostvarila konekcija sa određenom bazom podataka, prvo je potrebno registrovati JDBC driver pomoću JDBC `Driver Manager`-a. `Driver Manager` pruža osnovne servise za upravljanjem JDBC driver-ima [9]:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

Nakon registracije odgovarajućeg driver-a, pomoću `Connection` objekta, koji enkapsulira konekciju na odgovarajuću bazu podataka, formira se osnova za upravljanje podacima:

```
Connection con = DriverManager. getConnection ("url", "user", "password");
```

Metodi `getConnection()` se prosljeđuju tri parametra: JDBC URL, korisničko ime i lozinka. Kada se pozove ova metoda, `Driver Manager` poziva sve registovane driver-e i provjerava koji od njih razumije URL. Ukoliko driver razumije URL, on vraća `Connection` objekat.[9]

#### 3) Kreiranje SQL naredbi

Jednom kada se ostvari konekcija na bazu, ona se koristi za slanje SQL upita bazi podataka. Pošto ne postoje ograničenja koja se odnose na vrstu SQL naredbi koja se šalju pomoću JDBC-a, korisnik ima mogućnost da koristi upite specifične za bazu ili ne-SQL upite. JDBC API pruža tri klase za kreiranje SQL naredbi nad bazom podataka [9]:

*a) Statement* - objekat koji se koristi za slanje jednostavnih SQL naredbi

*b) PreparedStatement* – objekat koji se koristi za izvršavanje naredbi više puta

*c) CallableStatement* – objekat koji se koristi za izvršavanje SQL uskladištenih procedura.

#### 4) Manipulacija rezultatima SQL naredbi

Nakon što se izvrši SQL upit, rezultati formiraju pseudotabelu koja sadrži sve redove koji zadovoljaju kriterijum upita. Tekstualna reprezentacija koja nastaje ovim putem i koja sadrži podatke nije pogodna za Java programe. Umjesto toga, JDBC koristi `java.sql.ResultSet` interfejs za enkapsuliranje rezultata upita kao Java primitivne tipove i objekte. `ResultSet` se može predstaviti kao tabela rezultata upita u kojoj se mogu koristiti metode za navigaciju kroz redove i preuzimanje vrijednosti određenih kolona. [9]

Podaci se preuzimaju iz `ResultSet`-a upotrebom getter metoda koje referenciraju kolone sa podacima. `ResultSet` getter metode omogućavaju uzimanje podataka odgovarajućeg tipa iz kolona u trenutnom redu. Podaci u kolonama se u trenutnom redu mogu uzeti u bilo kojem redoslijedu. Preuzimanje podataka iz kolona se može obaviti na dva načina referenciranjem imena tabele ili referenciranjem broja kolone [9]:

```
ResultSet rs = stmt.executeQuery(query);
System.out.println(“Ime” + rs.getString(“Ime”) +
“Starost” + rs.getInt(3));
```

#### 5) Uskladištene procedure

Uskladištena procedura predstavlja kolekciju T-SQL (*eng*. Transact-SQL) naredbi koje mogu da prime i vrate parametre koje dobijaju od korisnika. T-SQL predstavlja proširenje SQL-a koje omogućava upotrebu uskladištenih procedura. Uskladištene procedure se mogu predstaviti kao metode ili funkcije napisane u T-SQL-u. [9]

Web aplikacija za alumni bazu visokoškolske ustanove, prilikom komunikacije sa bazom podataka i manipulisanja podacima u bazi, se oslanja na pozivanje uskladištenih procedura.

#### 6) DAO

Direktno manipulisanje podacima koji se nalaze u bazi podataka u aplikaciji se obavlja Java klasama upotrebom Data Access Object (DAO) šablona. DAO šablon je standardni šablon koji se koristi za enkapsuliranje pristupa podacima. Umjesto pozivanja JDBC-a ili nekog drugog API-ja direktno iz svih Java klasa koje pristupaju podacima, vrši se enkapsuliranje pristupa podacima u jedan ili više objekata za pristup podacima (*eng*. data access objects - DAO). DAO tipično sadrži metode za kreiranje, ažuriranje, uzimanje i brisanje objekata iz baze podataka, kao i metode za pristup bazi podataka koje vraćaju kolekcije objekata. [10]

*a) Primjena DAO šablona u Web aplikaciji za alumni bazu*

U aplikaciji za alumni bazu visokoškolske ustanove, pomoću Java klasa, obavljena je cjelokupna komunikacija sa

bazom podataka. Naime, za svaku CRUD operaciju (osim DELETE operacije) kreirane su posebne Java klase. Tako je za dobijanje podataka iz tabele, tj. izvršenje SELECT naredbe, kreirana klasa `SelectService` klasa, za promjenu podataka kreirana je `UpdateService` klasa i za dodavanje novog zapisa kreirana je `CreateService` klasa.

Na primjeru klase za unos novog zapisa u bazu podataka, može se prikazati primjena DAO šablona i pozivanje uskladištenih procedura. U `create()` metodi se poziva se poziva sljedeća procedura:

```
SP = conn.prepareCall("{ call add" + naziv + "('" +
parametri+ "') }",
SQLServerResultSet.TYPE_SCROLL_INSENSITIVE,
SQLServerResultSet.CONCUR_READ_ONLY);
```

Varijabla `parametri` predstavlja vrijednosti podataka koje se prema odgovarajućem redoslijedu upisuju u tabelu.

Nakon što se izvrše uskladištene procedure, potrebno je sačuvati informaciju o tome da li su se procedure uspješno izvršile. U `create()` metodi se u heš mapu upisuje podatak o uspješnosti i prebacuje u `json` format:

```
try {
        ...
        SP.executeUpdate();
        kon.put("Result", "OK");
        ...
        kon.put("Record", red);
} catch (SQLException e) {
        kon.put("Result", "ERROR");
        kon.put("Message", "Nije moguće dodavanje novog
zapisa");
        e.printStackTrace();
}
json = gson.toJson(kon);
```

*B. Pogled*

Dio pogled u okviru MVC arhitekture predstavljen je JSP stranicama. JSP stranica predstavlja Web stranicu koja sadrži elemente HTML-a, tj. HTML tag-ove. Pored HTML-a, JSP stranica može sadržati i JSP tag-ove koji dozvoljavaju serveru da ubaci dinamički sadržaj u stranicu. JSP elementi se mogu koristiti u različite svrhe, kao što je povrat informacija iz baze podataka. Kada korisnik zatraži određenu JSP stranicu, server izvršava JSP elemente i spaja rezultate sa statičkim dijelovima stranice i tako dinamički kreiranu stranicu šalje nazad Web pretraživaču.

Kada je riječ o komunikaciji između pogleda i kontrolera u okviru MVC arhitekture, podaci se razmjenjuju u oba smjera. U prvom slučaju, kada se vrši slanje podataka od pogleda ka kontrolerima, kreira se `request` objekat koji sadrži informacije o podacima koje je unio korisnik i dodatne informacije koje se implicitno kreiraju. Ovaj objekat se dalje obrađuje u servletima, kako bi se dobili željeni podaci.

U drugom slučaju, kada se podaci šalju od kontrolera ka pogledima, JSP stranica koristi `jsp:useBean` akciju kako bi dobila podatke iz zahtjeva koji je pristigao. Stranica za prikaz informacija o korisniku koji je trenutno prijavljen na sistem daje najbolji primjer upotrebe ove akcije.

Da bi se prikazali podaci o trenutno prijavljenom korisniku, JSP stranica mora da uzme podatke iz zahtjeva koji je poslao servlet, a to obavlja upotrebom `jsp:useBean` akcije:

```
<jsp:useBean id="trenutniKorisnik" class="model.UserBean"
        scope="session">
</jsp:useBean>
```

U prethodnom kodu instancira se objekat klase `UserBean` koji se nalazi u sesiji a čiji je naziv `trenutniKorisnik`. Ovom objektu se može pristupati bilo gdje unutar JSP stranice, kao npr:

```
<tr>
<td><img src="images/user.png"></td>
<td width="164" height="43" class="tabela">
         Ime
</td>
<td width="164" height="43" class="tabela">
        <%=trenutniKorisnik.getIme() %>
</td>
</tr>
```

Iz ovog primjera se vidi način uzimanja podatka o imenu korisnika koji je prijavljen u sistem i prikazivanja tog podatka u okviru tabele sa ostalim podacima o korisniku.

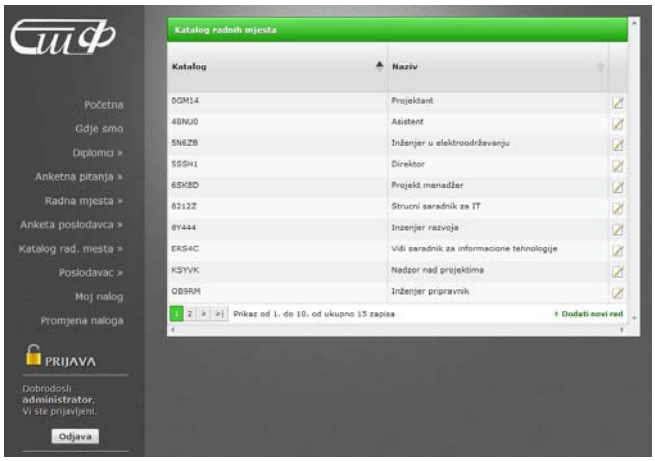
Osnovna komponenta koja se koristi u aplikaciji u JSP stranicama, JTable komponenta, kreirana je pomoću jQuery tehnologije i vrši Ajax pozive prema serveru. Pošto je kreirana pomoću jQuery tehnologije, sve akcije i informacije u vezi sa poljima tabele se navode u okviru `<script type="text/javascript"> </script>` tagova i `$(document).ready()` funkcije, kako bi se obezbijedilo da se akcije obavljaju prije učitavanja sadržaja stranice:

```
<script type="text/javascript">
$(document).ready(function () {
 $('#Tabela').jtable({
            title: 'Radno Mjesto',
            paging: true,
            pageSize: 10,
            defaultSorting: 'DR_OZNAKA ASC',
            actions: {
              ...
            },
            fields: {
              ...
           },
         ...
       });
});
</script>
```

Odavde se vidi da se na samom početku statički navodi naziv tabele i maksimalan broj redova koji se prikazuju na jednoj strani. Pored toga, navodi se i inicijalni poredak podataka koji se uzimaju iz baze, što je u ovom slučaju po nazivima u koloni DR_OZNAKA, i to u opadajućem redoslijedu. Kada korisnik vrši neku od CRUD operacija nad tabelom, JTable komponenta šalje Ajax poziv odgovarajućem servletu, a to je u ovom slučaju ProcessServlet:

```
actions : {
        listAction : 'ProcessServlet?akcija=select',
        updateAction : 'ProcessServlet?akcija=update',
        createAction : 'ProcessServlet?akcija=create'
},
```

Na osnovu vrijednosti parametra “akcija”, `ProcessServlet` poziva odgovarajuće metode za obavljanje CRUD operacija. Na slici 2, prikazan je izgled JSP stranice sa tabelom:




Slika 2. Izgled JSP stranice sa podacima iz baze podataka

## C. *Kontroler*

Kontroler unutar MVC arhitekture predstavlja HTTP servlet, koji nasljeđuje `javax.servlet.http.HttpServlet` klasu. Servleti predstavljaju Java programe koji se pokreću na Web ili aplikacionim serverima i koji se ponašaju kao srednji sloj između zahtjeva koji dolaze od Web pretraživača ili drugih HTTP klijenata i baza podataka na HTTP serverima. [11]

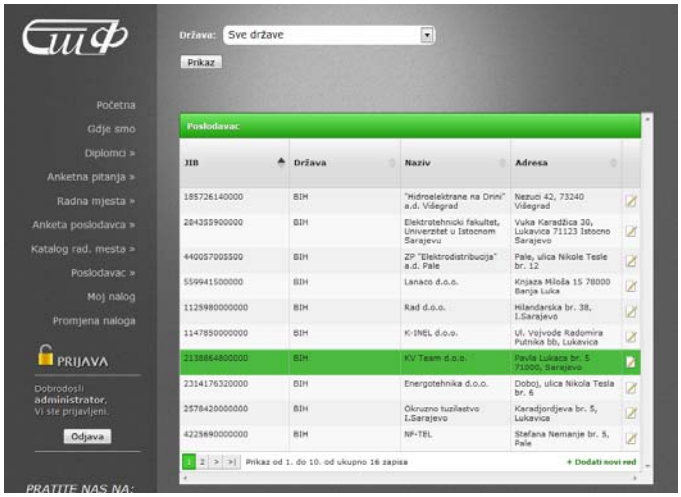
Kada se klijent poveže sa serverom i kreira HTTP zahtjev, taj zahtjev može biti različitog tipa tj. može biti u različitim metodama. Servlet posjeduje sljedeće metode: `doDelete()`, `doGet()`, `doPost()`, `doHead()`, `doOptions()`, `doPut()`, `doTrace()`, `getLastModified()` i `service()`. Najčešće korištene metode su `doGet()` i `doPost()` metode. Ove metode dozvoljavaju servlet-u da izvrši više različitih zadataka u zavisnosti od načina na koji je pozvan, obrađujući klijentske zahtjeve koji pristižu POST ili GET metodom. [11]

Unutar realizovane Web aplikacije nalaze se četiri servleta, koji zajedno predstavljaju kontrolerski dio aplikacije. Glavni dio kontrolera je predstavljen `ProcessServlet`-om u kojem je realizovana osnovna logika za obavljanje CRUD operacija u bazi podataka. Kada korisnik obavi određenu CRUD operaciju, informacija o operaciji se POST metodom šalje servletu. Prvo što je potrebno da `ProcessServlet` uradi je da uzme podatak o obavljenoj operaciji i da na osnovu te vrijednosti pozove odgovarajuće metode koje komuniciraju sa bazom.

U slučaju kada korisnik želi da otvori stranicu za prikaz podataka neke od tabela iz alumni baze, kreira se klijentski zahtjev za obavljanje SELECT operacije. `ProcessServlet` iz `Request` objekta uzima podatak o CRUD operaciji i dalje poziva odgovarajuće metode:

```
String akcija = request.getParameter("akcija");
if (akcija.equals("select")) {
int startIndex =
Integer.parseInt(request.getParameter("jtStartIndex"));
int pageSize =
Integer.parseInt(request.getParameter("jtPageSize"));
String sorting = request.getParameter("jtSorting");
        ...
SelectService select2 = new SelectService((String)
request.getSession().getAttribute("imeTabele"),
startIndex,pageSize, sorting, filterName, filterValue);
String json = select2.uzmiPodatkeTabele();
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8");
response.getWriter().write(json.toString());
}
```

Odavde se može vidjeti da nakon određivanja vrste CRUD operacije, koja je u ovom slučaju SELECT operacija, uzimaju podaci o rednom broju prvog reda i ukupnom broju redova koji se prikazuju na jednoj strani. Na osnovu tih podataka poziva se konstruktor `SelectService()` i metoda `uzmiPodatkeTabele()` koja uzima podatke iz tabele i vraća ih u formatu `json`-a. Te podatke `ProcessServlet` pomoću `Response` objekta vraća odgovarajućoj JSP stranici.

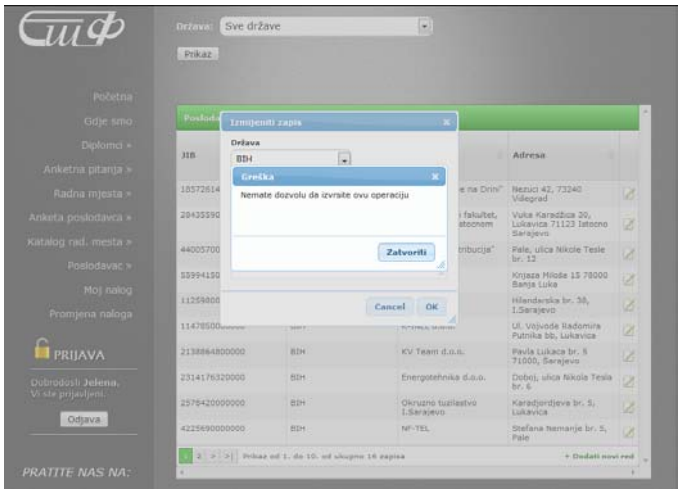



Slika 3. Izgled JSP stranice u slučaju uspješne promjene zapisa

U aplikaciji su UPDATE i INSERT operacije dozvoljene samo administratorima, dok ostali korisnici nemaju dozvolu za njihovo izvršenje. Ukoliko korisnik unese podatke za dodavanje ili promjenu zapisa u tabeli, `ProcessServlet` provjerava da li je korisnik administrator. Ukoliko jeste, uzimaju se podaci koje je korisnik unio i pozivaju se metode koje će se povezati sa bazom i izvršiti odgovarajuće uskladištene procedure:

```
UserBean user = (UserBean)request.getSession().
        getAttribute("trenutniKorisnik");
String json = "";
if (user.getNivo().intValue() == 9) {
Enumeration<?> parametri = request.getParameterNames();
        ...
UpdateService updateS = new UpdateService();
json = updateS.update((String) request.getSession()
        .getAttribute("imeSP"), parametri);
}
else {
json = "{\"Result\":\"ERROR\",\"Message\":\"Nemate
dozvolu da izvrsite ovu operaciju\"}";
}
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8");
response.getWriter().write(json.toString());
```

Podaci koje vraća metoda `update()` su u formatu `json`-a, i u zavisnosti od ishoda izvršenja UPDATE operacije, vraćaju se podaci o uspješnom ili neuspješnom izvršenju. Ukoliko korisnik koji želi da izvrši promjenu zapisa nije administrator, u `json` podatak koji se vraća JSP stranici se upisuju podaci o nepostojanju dozvole za izvršenje te operacije. Na slikama 3. i 4. je prikazan izgled iste stranice za slučaj uspješne i neuspješne promjene zapisa u bazi podataka, respektivno.




Slika 4. Izgled JSP stranice u slučaju neuspješne promjene zapisa

Prilikom INSERT operacije, `ProcessServlet` obavlja istu provjeru kao i u prethodnom primjeru, a jedina razlika je što će se umjesto `update()` metode pozvati `create()` metoda:

```
CreateService add = new CreateService();
json = add.create((String)
request.getSession().getAttribute("imeSP"),
                imenaKolona, parametri);
```

I ovdje će se u slučaju da korisnik nije administrator, u `json` podatak koji se vraća JSP stranici upisati informacija o nepostojanju dozvole za izvršenje INSERT operacije.

## VII. ZAKLJUČAK

Danas se može reći da se JSP tehnologija manje koristi u odnosu na druge Web tehnologije, a jedan od razloga je i način realizacije aplikacije koja nameće upotrebu MVC arhitekture. Osnovni cilj aplikacije je bio da se prikaže, kako se razdvajanjem na dijelove sa definisanim zadacima pojednostavljuje izrada Web aplikacije.

U radu je pažnja posvećena realizaciji dijelova MVC arhitekture u JSP tehnologiji u Web aplikaciji za alumni bazu. Kada je riječ o modelu, prikazano je kako se, primjenom DAO šablona, realizuju tzv. *JavaBeans* klase, tj. klasične Java klase zadužene za povezivanje sa bazom podataka i obavljanje CRUD operacija nad podacima u bazi. Pošto se Web aplikacija povezuje sa udaljenom bazom podataka, u ovom dijelu prikazan je i način povezivanja aplikacije i baze upotrebom JDBC driver-a. U dijelu za kontrolere, predstavljeni su servleti i njihove dvije najznačajnije metode `doPost()` i `doGet()` koje pružaju servlet-ima mogućnost da usaglase rad svih dijelova MVC arhitekture. JSP stranice su opisane u okviru *view* dijela, gdje je akcenat stavljen na metode koje JSP stranice koriste za uzimanje i predstavljanje podataka koje dobiju od kontrolera.

## ZAHVALNICA

Veliku zahvalnost dugujemo prof. dr Branku Perišiću, koji je ujedno i mentor ovog rada, i asistentu Vladimiru Vujoviću na neiscrpnoj pomoći tokom izrade teorijskog i praktičnog dijela u okviru projekta-2 i završnog rada.

## LITERATURA

[1] Tim Barners-Lee. "Information Management: A Proposal" Internet: http://www.w3.org/ Proposal.html, Novembar 1990. god. [27.Nov.2012.].

[2] S.C. Yadav i S.K.Singh. "Introduction," u *An Introduction To Client Sever Computing*, vol. 1, New Delhi, India: New Age International Publishers, 2009, pp. 1-25.

[3] R. Orfali, D. Harkey i J. Edwards. "Web Client/Server: The Hypertext Era," u *Client/Server Survival Guide*, vol. 3, New York, United States: John Wiley & Sons, 1999.

[4] G. Zambon i M. Sekler. "Introducing JavaServer Pages and Tomcat" u *Beginning JSP, JSF, and Tomcat Web Development: From Novice to Professional*, vol. 1, New York, United States: Apress, 2007.

[5] P. Delisle. "Overview" u *JSP Specification*, vol. 2, Santa Clara, United States: Sun Microsystems, Inc., 2009.

[6] Trygve Reenskaug. "Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC)" Internet: http://st-www.cs.illinois.edu/users/smarch/ st-docs/mvc.html, 1979. god. [01.Jan.2013.].

[7] Microsoft. "Model-View-Controller" Internet: http://msdn.microsoft.com/en-us/library/ff649643.aspx, 2012. god. [01.Jan.2013.].

[8] Jim Farley i William Crawford. "JDBC" u *Java Enterprise in a Nutshell*, vol. 3, Sebastopol, United States: O'Reilly Media, 2005.

[9] John O'Donahue. "Introduction to JDBC" u *Java Database Programming Bible*, vol. 1, , New York, United States: John Wiley and Sons, 2005.

[10] S.Brown, S.Dalton i drugi. "Data Access Options for Web Applications" u *Pro JSP 2*, vol. 4, New York, United States: Apress, 2005.

[11] Marty Hall i Larry Brown. "Servlet basics" u *Core Servlets and JavaServer Pages*, vol. 2, Santa Clara, United States: Prentice Hall, 2003.

## ABSTRACT

At the beginning of the paper will be presented the World Wide Web and client-server architecture, which is the most common architecture in Web applications. Based on this architecture, it is made distribution of Web technologies that are executed on the client and on the server side. This paper will address the JSP technology as one of the technologies that are executed on the server side. After that will be discussed the MVC architecture and implementation of this architecture in the JSP, i.e. how is made the division on model (using the classic Java classes), controller (using servlets) and view (using standard JSP pages). The paper will give examples of practical realization of Web application for alumni database of the university.

## IMPLEMENTATION OF MVC ARCHITECTURE IN JSP TECHNOLOGY

Jelena Ćosović, Miljan Radojičić