УНИВЕРЗИТЕТ „УНИОН“
РАЧУНАРСКИ ФАКУЛТЕТ
Кнез Михаилова 6/VI
11000 БЕОГРАД

Број:
Датум:

UNIVERZITET UNION

RAČUNARSKI FAKULTET

BEOGRAD

RAČUNARSKE NAUKE

Diplomski rad

Kanditat: Stefan Dimić
Broj indeksa: 06/07

Tema rada: Implementacija društvene mreže upotrebom JavaEE platforme

Mentor rada: prof. dr. Vidaković Milan

Beograd, 2012

Apstrakt

Tema rada je opis implementacije jednog dela društvene mreže pod nazivom YouAndShoe. YouAndShoe je društvena mreža koncipirana oko ideje da se korisnicima pruži bogatije iskustvo pri izboru obuće pružajući im i dodatne informacije u vidu društvenih komponenti kao sto su komentari, ocenjivanje, recenzije i druge. Implementirani deo predstavlja profilsku stranicu određenog modela obuće koja te informacije prikazuje u sklopu objedinjenog korisničkog interfejsa.

Kompletna web aplikacija je razdvojena u dva projekta, serverski projekat koji sadrži modele podataka i koji obezbeđuje interfejse za manipulaciju nad tim podacima i GUI projekat koji sadrzi korisnički interfejs i sprege ka serverskom projektu. GUI projekat pristupa podacima na serverskom delu i manipuliše njima isključivo preko fasada.

Implementacija je realizovana u Java programskom jeziku pomoću Java Server Faces web framework-a, Richfaces i jQuery biblioteka kao i JPA 2.0 za perzistenciju podataka. Za aplikacioni server izabran je JBoss 5.1. Serverske fasade su implementirane pomoću EJB 3.0.

Rad će opisati razvoj klijentskog i serverskog dela profilske stranice, komponenti koje stranica koristi kao i prepreke koje su se javljale tokom razvoja.

Sadržaj

Uvod

Kako je projekat u celini obiman, akcenat će biti stavljen na profilsku stranu koja je ujedno i najbogatija sadržajem pa samim tim i najkompleksnija. Ostali delovi aplikacije biće opisani u letu, najčešće u delovima rada u kojima je predstavljena neka komponenta profilske strane koja je ponovno iskorišćena u nekom drugom delu aplikacije.

Poglavlje *Korišćene tehnologije* detaljnije opisuje i daje primere tehnologija i alata koji su korišćeni u implementaciji projekta.

U nastavku sledi poglavlje *Opis implementacije upotrebom Java Server Faces 1.2* u kojoj se detaljno opisuje specifikacija i implementacija korisničkog interfejsa i backend dela aplikacije. Ovo poglavlje nosi naziv *Opis implementacije upotrebom Java Server Faces 1.2* jer će u poglavlju *Migracija* biti predstavljena web aplikacija koja je u procesu razvoja i koja predstavlja evolutivni korak prve verzije kako u dizajnu tako i u korišćenju novih aktuelnih tehnologija.

Nakon poglavlja *Opis implementacije upotrebom Java Server Faces 1.2* sledi poglavlje *Logika aplikacije* u kojem se detaljno opisuje backend logika projekta. Poglavlje *Korisnički interfejs* opisuje komponente korisničkog interfejsa profilske strane

Poslednje poglavlje *Migracija* opisuje prelazak na aktuelne tehnologije kao što su JavaEE 6, JSF 2 i druge.

Pregled korišćenih tehnologija

Java Enterprise Edition

Java Enterprise Edition[1] predstavlja široko korišćenu platformu za serversko programiranje zasnovanu na Java programskom jeziku. Ona nadgrađuje standardnu Javu dodavanjem biblioteka koje omogućuju razvijanje web i enterprise aplikacija koje se izvršavaju na aplikacionom serveru. Kako je Java EE definisana svojom specifikacijom, provajder usluga (kao što je JBoss[2] aplikacioni server) mora da ispuni određene uslove kako bi svoje proizvode mogao da proglasi usaglašenim sa Java EE specifikacijom. Java EE uključuje nekoliko API specifikacija kao što su JDBC[3], RMI[4], servleti, Java Server Pages i druge. Java EE aplikacioni server rukuje transakcijama, bezbednošću, skalabilnošću, konkurentnošću i upravljanjem komponenti koje se na njemu izvršavaju, olakšavajući i ubrzavajući razvoj aplikacije.

JBoss

JBoss je aplikacioni server baziran na Java Enterprise Edition platformi. On ne samo da implementira server koji se izvršava na Java virtuelnoj mašini, nego i Enterprise deo Java specifikacije. Kako je baziran na Javi, JBoss server se može koristiti na svim operativnim sistemima koji podržavaju i Javu. Neke od glavnih karakteristika JBoss-a su:

- Podrška za aspekt-orijentisano programiranje
- Podrška za Enterprise JavaBeans 3 i 2.1
- Integracija Hibernate-a
- Podrška za Java Server Pages i Java Servleta
- Balansiranje opterećenja

Servleti

Servlet[5] predstavlja Java klasu u Java EE okruženju koja je u skladu sa Java Servlet API-jem, protokolom po kome Java klasa može da odgovara na zahteve. Servleti nisu vezani za specifičan klijent-server protocol, ali se najčešće koriste sa HTTP protokolom. Da bi se servlet koristio, potreban je Web kontejner. Web kontejner je komponenta Web servera koja upravlja životnim ciklusom servleta.

Filteri

Filter[6] predstavlja Java klasu koja može da izmeni zaglavlje i sadržaj (ili oba) nekog zahteva ili odgovora. Filteri obično ne kreiraju odgovore, za razliku od servleta. Umesto toga oni pružaju funkcionalnost koja se može ‘nakačiti’ na bilo koji web resurs.

Neki od scenarija u kojima su filteri pogodni za upotrebu su: autentifikacija, logovanje, kompresija, enkripcija, itd.

Java Server Faces (JSF)

Java Server Faces[7] predstavlja web framework zasnovan na Model-View-Controller[8] paradigmi i na HTTP zahtev/odgovor principu. JSF framework sadrži jedan master Controller, takozvani Faces servlet koji procesira dolazeće zahteve, učitava odgovarajući View na osnovu parametara zahteva, i isporučuje odgovarajući odgovor klijentu. Slede glavne karakteristike JSF framework-a.

Managed Beans[9] - predstavljaju java klase proglašene kao 'managed' (upravljan), u konfiguracionom fajlu samog framework-a. Da bi klasa bila managed potrebno joj je dodeliti neki od definisanih opsega (scope). JSF 1.2 implementacija definiše 3 opsega:

- *Request* - Životni vek managed objekta je ograničen na dolazni zahtev. JSF proizvodi novu instancu klase za svaki dolazeći zahtev, a na kraju zahteva ih uništava.

- *Session* - Životni vek objekta ograničen je na trajanje sesije jednog korisnika. Svaki korisnik poseduje jedinstvenu sesiju, tako da su i objekti sesije jedinstveni za svakog korisnika. Dokle god je sesija aktivna JSF će koristiti objekat asociran sa tom sesijom. Nakon isteka sesije JSF će uništiti sve objekte koji se nalaze u opsegu sesije.

- *Application* - Životni vek objekta vezan je za životni vek same aplikacije. Managed bean-ovi čiji je opseg aplikacija, biće kreirani prilikom podizanja same aplikacije na serveru, a uništeni kada se ugasi sama aplikacija na serveru. Ove objekte dele svi korisnici.

Request, Session i Application opsezi su implementirani kao mape koje sadrže parove ključ/vrednost.

```
<managed-bean>
   <description>Bean containing model of the searched shoe.</description>
   <managed-bean-name>currentShoeBean</managed-bean-name>
   <managed-bean-class>com.ui.ShoeBean</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

<managed-bean>
   <description>Bean containing selected tag information.</description>
   <managed-bean-name>searchTagBean</managed-bean-name>
   <managed-bean-class>com.ui.SearchTagBean</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
   <description>A bean containing all search filters</description>
   <managed-bean-name>filterListBean</managed-bean-name>
   <managed-bean-class>com.util.FilterListBean</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Listing 1. Primer proglašavanja opsega managed bean-ova

Managed Beans klase najčešće enkapsuliraju model podataka i poseduju metode koje se pozivaju od strane klijenta.

Java Server Pages[10] - predstavlja java tehnologiju za generisanje dinamičkog sadržaja zasnovanog na XHTML. JSF 1.2 koristi ovu tehnologiju kao podrazumevanu za generisanje prikaza tj. View dela aplikacije. JSF sadrži set XHTML i web orijentisanih UI komponenti koje se lako ugrađuju u JSP strane i olakšavaju razvoj korisničkog interfejsa.

```
<h:panelGroup styleClass="price_buy_holder" layout="block">
    <h:panelGroup styleClass="price_holder" layout="block">
        <h:panelGroup styleClass="price_wrapper" layout="block">
            <h:outputText value="#{currentShoeBean.price }"></h:outputText>
        </h:panelGroup>
    </h:panelGroup>

    <h:panelGroup styleClass="buy_holder" layout="block">
        <h:form>
            <a4j:jsFunction immediate="true" name="bindSplash"
                            oncomplete="goToBuyPage();">
                <f:setPropertyActionListener
                            target="#{currentUser.splashLink}"
                            value="#{currentShoeBean.buyLink }" />
            </a4j:jsFunction>
        </h:form>

        <h:outputLink target="_blank"
                      value="/jsp/splash.jsf?tgt=#{currentShoeBean.buyLink }">
            <h:panelGroup layout="block" styleClass="buy_btn buy_btn_unhovered" />
        </h:outputLink>
    </h:panelGroup>
</h:panelGroup>
```

Listing 2. Ilustracija JSF tagova unutar Java Server Pages strane.

Unified Expression Language (EL)[11] - predstavlja programski jezik posebne namene koji se koristi za ugrađivanje izraza u JSP stranice. EL sintaksom moze se pozvati proizvoljna metoda nekog managed objekta ili sam objekat. Novije implementacije EL mogu da zovu i metode sa parametrima, prosledjujući te parametre direktno iz JSP-a (ili neke druge tehnologije za generisanje prikaza). Objektima se pristupa preko sintakse #{*izraz*}.

```
<h:outputText value ="#{article.title }"rendered="#{not empty article.title }"/>
```

Listing 3. Primer pozivanja metoda objekta EL sintaksom

JSON

JSON (JavaScript Object Notation)[12] predstavlja lak format za razmenu podataka. Lak za ljude da ga pišu i čitaju, lak za mašine da ga parsiraju i koriste. Primer JSON formata:

{ "**id**": 1, "**name**": "Foo", "**price**": 123, "**tags**": ["B","E"], "**stock**": { "**warehouse**": 300, "**retail**": 20 } }

Vitičaste zagrade označavaju jedan objekat, pravougle zagrade predstavljaju niz. Vrednosti pre i posle dvotačke predstavljaju par ključ/vrednost.

AJAX

Ajax (Asynchronous JavaScript and XML)[13] predstavlja grupu tehnologija koje se zajedno koriste na klijentskoj strani za kreiranje asinhronih web aplikacija. Upotrebom ove tehnologije web aplikacije mogu da šalju podatke serveru i da ih prime od servera 'u pozadini', bez mešanja u trenutan prikaz stranice. Ovim mehanizmom se mogu osvežiti delovi stranice. Stranica samim tim postaje bogatija i prijatnija za korišćenje.

Iako sadrži XML u svom imenu, sam XML nije neophodan za korišćenje Ajax-a, čak se u najvećem broju slučajeva koristi JSON format.

```
$.getJSON('/brands', function(result) {
  console.log('Fetched brands: ' + result);
});
```

Listing 4. Primer asinhronog GET zahteva koji gadja java Servlet mapiran na "/brands" url. Metoda getJSON() jQuery biblioteke automatski parsira odgovor u JSON objekat.

Richfaces

Kako Java Server Faces 1.2 ne poseduje ugrađene Ajax sposobnosti potrebno je koristiti dodatne biblioteke koje to omogućuju. Jedna takva biblioteka je Richfaces[14]. Pomoću Richfaces biblioteke moguće je standardnim JSF komponentama dodati AJAX funkcionalnost, ili iskoristiti neku od predefinisanih Richfaces komponenti.

```
<a4j:form>
   <h:inputText value="#{userBean.name}" />
   <a4j:commandButton value="Say Hello" reRender="out" />
</a4j:form>

<h:panelGroup id="out">
   <h:outputText value="Hello #{userBean.name}"/>
</h:panelGroup>
```

Listing 5. Primer asinhronog zahteva pomoću <a4j:commandButton/> komponente. Kada klijentu stigne odgovor, samo će komponente čiji je *id* naveden u *reRender* atributu <a4j:commandButton/> biti ponovo osvežene.

jQuery

jQuery[15] predstavlja moćnu javascript biblioteku, kompatibilnu sa svim popularnum web pretraživačima, a dizajnirana sa ciljem da se olakša pisanje javascript koda na klijentskoj strani. Biblioteka apstrahuje ponašanja različitih pretraživača u kratku i konciznu sintaksu.

```
$(".elem").addClass("blue").text('example text');
```

Listing 6. Primer jednostavne manipulacije DOM elemenata pomoću jQuery biblioteke. Izraz će pronaći sve elemente koji imaju atribut *class* postavljen na 'elem', dodati im novu CSS klasu *blue* i svima im postaviti tekst 'example text'.

EJB Stateless Session Beans

EJB predstavlja server-side model koji enkapsulira biznis logiku aplikacije. EJB kontejner sadrži dve vrste bean-ova:

- Session Beans – koji mogu biti "Stateful", "Stateless" ili "Singleton"
- Message Driven Beans

Stateless Session Beans[16] su biznis objekti koji ne poseduju stanje. Korisnik nema garancije da će se za sledeći poziv metode koristiti instanca stateless bean-a koja je i prvi put korišćena. Korišćenje jedne instance bean-a je ograničeno na jednog korisnika istovremeno. Ukoliko se pokuša konkurentan pristup jednom bean-u, kontejner jednostavno preusmerava zahteve na drugu instancu. Session bean-u se pristupa preko Local ili Remote interfejsa, u zavisnosti da li se konkrentna implementacija session bean-a nalazi na istoj java virtuelnoj mašini ili na nekom drugom (udaljenom) sistemu.

Hibernate

Hibernate[17] predstavlja objektno-relacioni maper za Java programski jezik, koji je zadužen za mapiranje objektno-orijentisanih modela u tradicionalne relacione baze.

Hibernate-ova primarna funkcija je mapiranje iz Java klasa u tabele baza. Takodje pruža alate za perzistovanje i pronalaženje Java objekata. Hibernate biblioteka je jednostavna za korišćenje jer se mapiranje klasa svodi na upotrebu Java anotacija. Anotirane klase nose naziv *entiteti (Entities).*

```java
@Entity
@Table(name="SHOE")
public class Shoe implements Serializable {

    private static final long serialVersionUID = -6807408676660962072L;

    @Id
    @Column(name="SEARCH_SHOE_ID")
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column(name="NAME")
    private String name;

    @Column(name="BRAND")
    private String brand;

    @Column(name="PRICE")
    private Double price;

    public Shoe(){}

    //Getters and setters ommited
}
```

```java
@Stateless
public class ShoeFacade {

    @PersistenceContext
    private EntityManager em;


    public void insertShoe(Shoe shoe) {
        em.persist(shoe);
    }

    public Shoe getShoeById(Integer id) {
        return em.find(Shoe.class, id);
    }

    public void updateShoe(Shoe shoe) {
        em.merge(shoe);
    }
}
```

Listing 7. Primer mapirane klase (LEVO) i primer EJB Stateless bean-a koji sadrži biznis logiku za manipulaciju nad mapiranom klasom (DESNO).

PrettyFaces

PrettyFaces[18] biblioteka predstavlja ekstenziju u obliku filtera za Servlet, Java EE i Java Server Faces tehnologije. Ona omogućava kreiranje URL-ova koji su laki za sačuvavanje u pretraživaču, prijatniji i razumljiviji za ljude i najvažnije, kreiranje URL-ova koji su pogodni za SEO (Search Engine Optimization).

PrettyFaces pretvara URL tipa: *youandshoe.com/jsp/shoe.jsf?title=Nike&color=Red&id=123* u dosta čitljiviji URL tipa: *youandshoe.com/Nike/Red/123*

Da bi PrettyFaces filter obavljao svoj zadatak potrebno je definisati mapiranje u njegovom xml konfiguracionom fajlu:

```
<url-mapping id="shoe">
   <pattern value="/shoe/#{title}/#{color}/#{id}"></pattern>
   <view-id value="/jsp/shoe.jsf"></view-id>
</url-mapping>
```

Listing 8. Primer PrettyFaces konfiguracionog fajla

PrettyFaces filter će parsirati parametre dolaznog zahteva i smestiti ih u Request mapu asociranu sa tim zahtevom. Parametri će biti dostupni pod imenima datim u pravilu mapiranja (u primeru sa slike, prvi parametar će se zvati *title*, drugi *color*, itd.).

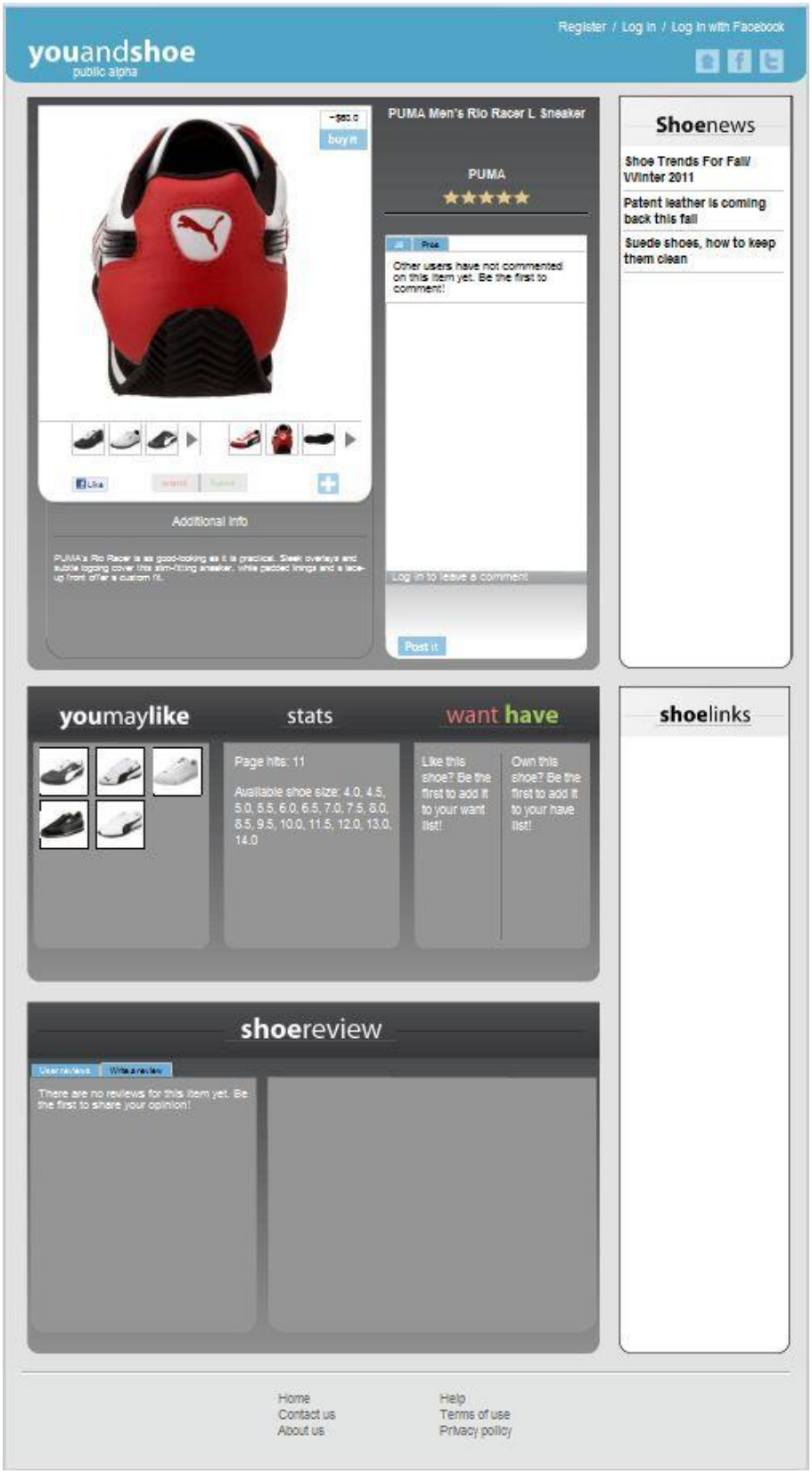
Opis implementacije upotrebom JSF 1.2

Uvod

YouAndShoe projekat je podeljen u dva web projekta koji su postavljeni na JBoss aplikacioni server kao jedna enterprise arhiva.

Prvi projekat predstavlja korisnički interfejs i sadrži biblioteke, resurse, Java Server Pages strane i managed bean-ove koji se tiču samog korisničkog interfejsa.

Drugi projekat predstavlja backend logiku aplikacije i sadrži modele klasa mapirane pomoću Hibernate biblioteke na odgovarajuće tabele kao i session bean-ove koji sadrže biznis metode za rad nad tim modelima. Session bean-ovi imaju ulogu fasada preko čijih interfejsa se pozivaju biznis metode od strane korisničkog dela projekta. Primer jednog takvog poziva bilo bi ostavljanje komentara na profilskoj strani cipele. Managed bean-u koji je zadužen za trenutni prikaz profilske strane prosledjuje se tekst komentara sa klijenta. Managed bean prvo dezinfikuje prosleđen tekst, uklanjajući maliciozne tagove, zatim pravi entitet vezan za komentar i poziva metodu fasade zaduženu za sačuvavanje komentara u bazu.

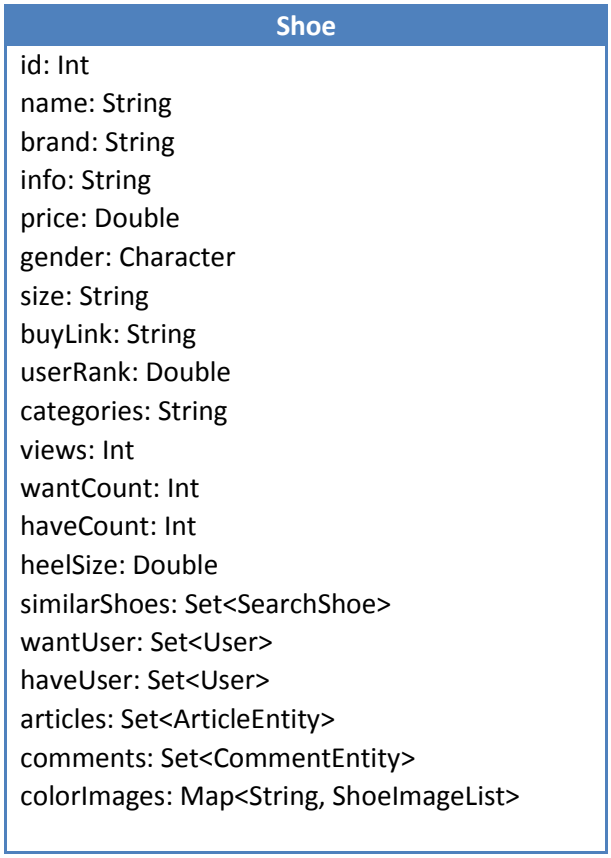


Slika 1. Pregled korisničkog interfejsa profilske strane

Logika aplikacije

Shoe entitet – predstavlja model koji sadrži sve potrebne informacije vezane za jednu cipelu. Klasa je anotirana kao entitet i mapirana na odgovarajuću tabelu u bazi. Kako Shoe klasa predstavlja model, ona ne sadrži nikakve metode koji manipulišu njenim atributima.

Pregled atributa Shoe klase dat je klasnim dijagramom:



Slika 2. Klasni dijagram Shoe entiteta

Sledi opis atributa Shoe klase:

- Mapiranje same Shoe klase:

```
@Entity
@Table(name="SHOE")
public class Shoe implements Serializable {}
```

- **id** – predstavlja primarni ključ Shoe objekta, mapiran kao:

```
@Column(name="ID")
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
```

gde @Column(name=”ID”) predstavlja ime kolone u tabeli na koje će se mapirati atribut.

- **name** – ovaj atribut predstavlja samo ime cipele (npr.: *PUMA Men's Rio Racer Sneaker*), mapirano kao:

```
@Column(name="NAME")
private String name;
```

- **brand** – predstavlja marku cipele (npr.: Puma), mapiranu kao:

```
@Column(name="BRAND")
private String brand;
```

- **info** – predstavlja dodatne informacije o cipeli, date od strane proizvođača. Atribut je mapiran kao:

```
@Column(name="INFO")
@Length(max=2000)
private String info;
```

- **price** – predstavlja cenu cipele, mapiranu kao:

```
@Column(name="PRICE")
private Double price;
```

- **gender** – atribut koji ukazuje da li je cipela za muškarce ili žene, mapiran kao:

```
@Column(name="GENDER")
private Character gender;
```

- **size** – atribut koji sadrži veličine u kojima je dostupna cipela, mapiran kao:

```
@Column(name="SIZE")
private String size;
```

- **buyLink** – atribut predstavlja URL ka sajtu prodavca. U trentunoj implementaciji ovaj link vodi na Amazon. Atribut je mapiran kao:

```
@Column(name="BUY_LINK")
private String buyLink;
```

- **userRank** – atribut predstavlja kumulativnu ocenu koju su registrovani korisnici dali nekoj cipeli**.** Vrednost ovog atributa je decimalna vrednost izmedju 1.0 i 5.0. Atribut je mapiran kao:

```
@Column(name="USER_RANK")
private Double userRank;
```

- **categories** – atribut koji opisuje kategoriju cipele (npr.: Men's / Athletic). Atribut je mapiran kao:

```
@Column(name = "CATEGORIES")
@Length(max=2000)
 private String categories;
```

- **views** – Broj koji pokazuje koliko je puta neka profilska strane cipele otvorena u web pretraživaču. Atribut je mapiran kao:

```
@Column(name="VIEWS")
private Integer views;
```

- **wantCount** – Broj koji pokazuje koliko je registrovanih korisnika stavilo trenutnu cipelu u svoju listu želja. Atribut je mapiran kao:

```
@Column(name="WANT_COUNT")
private Integer wantCount;
```

- **haveCount** – Broj koji pokazuje koliko je registrovanih korisnika stavilo trenutnu cipelu u svoju listu cipela koje poseduju. Atribut je mapiran kao:

```
@Column(name="HAVE_COUNT")
private Integer haveCount;
```

- **heelSize** – Atribut sadrži veličine peta cipele u kojima je trenutna cipela dostupna. Mapiran je kao:

```
@Column(name="HEEL_SIZE")
private Double heelSize;
```

- **similarShoes** – Atribut predstavlja set SearchShoe objekata koje predstavljaju cipele koje su po određenim kriterijumima slične izabranoj cipeli. SearchShoe klasa predstavlja 'lakšu' verziju Shoe klase. Atribut je mapiran kao:

```
@OneToMany(cascade={CascadeType.ALL}, fetch=FetchType.LAZY,
        targetEntity=SearchShoe.class)
@JoinTable(name="SHOE_SIMILAR",
        joinColumns={@JoinColumn(name="SHOE_ID")},
        inverseJoinColumns={@JoinColumn(name="SIMILAR_ID")})
private Set<SearchShoe> similar;
```

- **wantUser** – Atribut predstavlja set User objekata u čijoj se listi želja nalazi izabrana cipela. User klasa je, kao i Shoe klasa, entitet koji modeluje podatke jednog registrovanog korisnika. Atribut je mapiran kao:

```
@ManyToMany(mappedBy="wantList", cascade={CascadeType.ALL},
        fetch=FetchType.LAZY, targetEntity=User.class)
private Set<User> wantUser;
```

- **haveUser** – Atribut predstavlja set User objekata čija lista cipela koje poseduju sadrži trenutnu cipelu. Atribut je mapiran kao:

```
@ManyToMany(mappedBy="hasList", cascade={CascadeType.ALL},
        fetch=FetchType.LAZY, targetEntity=User.class)
private Set<User> haveUser;
```

- **articles** – Atribut predstavlja set ArticleEntity objekata. ArticleEntity je entitet koji modeluje pojedinačnu recenziju cipele od strane registrovanog korisnika. Atribut je mapiran kao:

```
@OneToMany(cascade={CascadeType.ALL}, fetch=FetchType.LAZY,
        targetEntity=ArticleEntity.class)
@JoinTable(name="SHOE_ARTICLE",
joinColumns={@JoinColumn(name="SHOE_ID")},
        inverseJoinColumns={@JoinColumn(name="ARTICLE_ID")})
private Set<ArticleEntity> articles;
```

- **comments** – Atribut predstavlja set CommentEntity objekata. CommentEntity je entitet koji modeluje pojedinačan komentar o nekoj cipeli od strane registrovanog korisnika. Atribut je mapiran kao:

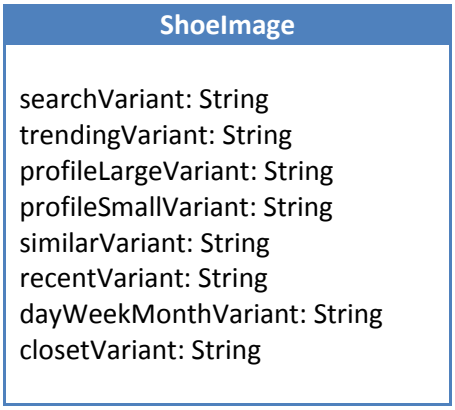
```
@OneToMany(cascade={CascadeType.ALL}, fetch=FetchType.LAZY,
        targetEntity=CommentEntity.class)
@JoinTable(name="SHOE_COMMENT",
         joinColumns={@JoinColumn(name="SHOE_ID")},
        inverseJoinColumns={@JoinColumn(name="COMMENT_ID")})
private Set<CommentEntity> comments;
```

- **colorImages** – Atribut predstavlja mapu čiji ključevi nose imena boja u kojima je trenutna cipela dostupna, a vrednosti mape su objekti ShoeImageList klase. Atribut je mapiran kao:

```
@OneToMany(cascade={CascadeType.ALL}, fetch=FetchType.EAGER)
        @JoinTable(name="SHOE_COLOR_IMAGE",
             joinColumns={@JoinColumn(name="SHOE_ID")},
            inverseJoinColumns={@JoinColumn(name="COLOR_IMAGE_ID")})
        private Map<String, ShoeImageList> colorImages;
```

ShoeImageList klasa enkapsulira listu ShoeImage objekata. Pojedinačan ShoeImage objekat predstavlja jednu orijentaciju slike cipele i on sadrži URL-ove do različitih veličina slika te orijentacije.

Sledi klasni dijagram ShoeImage entiteta:



Slika 3. Klasni dijagram ShoeImage entiteta

- **searchVariant** – URL do slike veličine 170x170 piksela. Ova varijanta se koristi za rezultate pretrage.
- **trendingVariant** – URL do slike veličine 32x32 piksela. Varijanta se koristi na početnoj strani za prikaz popularnih cipela.
- **profileLargeVariant** – URL do slike veličine 356x356 piksela. Varijanta se koristi na profilskoj strani za prikaz glavne slike izabrane cipele.
- **profileSmallVariant** – URL do slike veličine 38x38 piksela. Varijanta se koristi za prikaz orijentacija i boja cipele.
- **similarVariant** – koristi se za prikaz sličnih cipela na profilskoj strani.
- **recentVariant** - koristi se za prikaz cipela koje je korisnik skoro posetio. Ovaj prikaz se nalazi na profilskoj strani korisnika.
- **dayWeekMonthVariant** – koristi se za prikaz cipele dana/nedelje/meseca na početnoj strani.
- **closetVariant** – varijanta se koristi za prikaz cipela iz liste želja na profilskoj strani korisnika.

ShoeBean klasa

Predstavlja managed bean koji enkapsulira Shoe entitet i preko svojih metoda izlaže vrednosti polja Shoe entiteta JSP strani.

Kako ova klasa enkapsulira samu cipelu, za očekivati je pojavu visokog saobraćaja tj. korisnici će najveći deo vremena provesti u otvaranju stranica o cipeli u novim tabovima i/ili prozorima web pretraživača.

Samim tim sledi logičan zaključak da ovaj managed bean treba da poseduje što kraći životni ciklus kako se server ne bi preopteretio. Tom logikom, ShoeBean je stavljen u request opseg, što znači da će framework uništiti ShoeBean na kraju svakog zahteva.

Problem koji se javlja sa request opsegom je što će svaki AJAX poziv izazvati uništenje trenutnog ShoeBean-a, a to će za posledicu imati prikaz veoma ružne greške na strani klijenta.

AJAX osvežavanje profilske stranice se upotrebljava za gotovo svaku akciju koju korisnik može da pozove na profilskoj strani cipele, što znači da bi stranica bila potpuno nefunkcionalna u kombinaciji sa AJAX-om i ShoeBean-om u request opsegu.

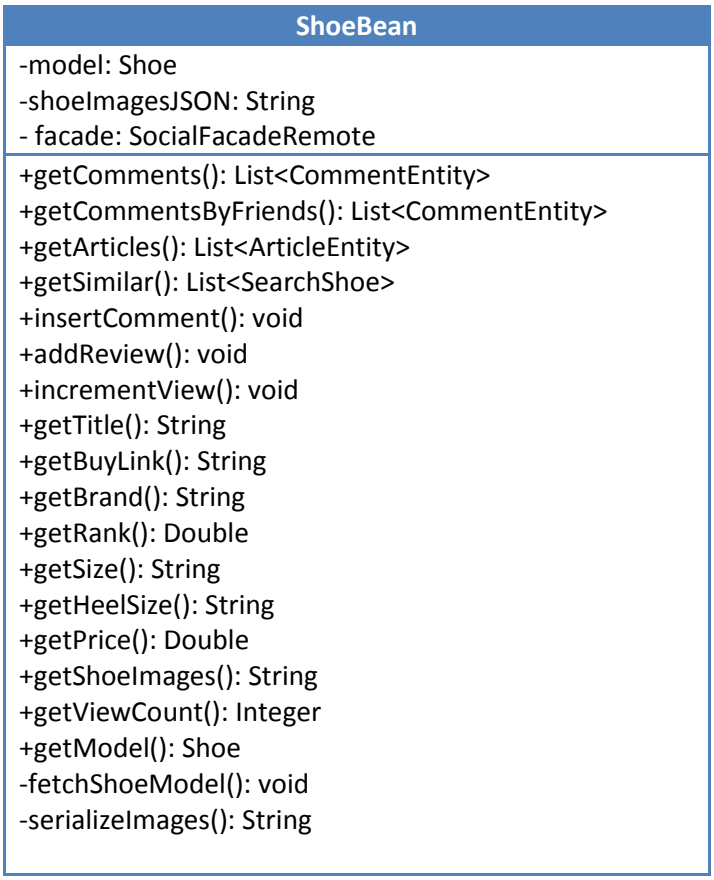
Rešenje bi bilo korišćenje takozvanog View opsega. View opseg je opseg koji se po svom životnom veku nalazi izmedju Request i Session opsega. Managed bean objekat sa ovim opsegom će ostajati u životu izmedju zahteva koji dolaze od istog klijenta, idealno rešenje kada se koristi AJAX.

Kako JSF 1.2 implementacija ne poseduje View opseg, rešenje je nađeno u Richfaces biblioteci u obliku < a4j:keepAlive /> taga koji upravo imitira željeni opseg. Ovaj tag će managed bean koji se nalazi u request opsegu veštački održavati u životu, dokle god dolaze srodni zahtevi.

```
<a4j:keepAlive beanName="currentShoeBean"></a4j:keepAlive>
```

Listing 9. Primer upotrebe <a4j:keepAlive/> taga. Managed bean, sa request opsegom, čije se ime nalazi u *beanName* parametru biće u simuliranom View opsegu

Klasni dijagram ShoeBean klase:



Slika 4. Klasni dijagram ShoeBean klase

Detaljniji opis atributa i metoda ShoeBean klase:

- *facade* – interfejs ka Stateless Session Bean-u
- *model* – model trenutno izabrane cipele
- *ShoeImagesJSON* – Lista URL-ova slika trenutne cipele serijalizovana u JSON format radi lakše manipulacije na klijentskoj strani
- *fetchShoeModel()* - metoda koja inicijalizuje *model* na osnovu parametara URL-a
- *insertComment()* – akcija kojom korisnik dodaje svoj komentar na profilskoj strani cipele
- addReview() – akcija kojom korisnik dodaje recenziju za trenutnu cipelu
- *incrementView()* – metoda koja broji koliko je puta nekoj profilskoj strani bilo pristupano
- *serializeImages()* – metoda koja serijalizuje URL-ove slika cipele pomoću Gson[19] biblioteke

Gson biblioteka konvertuje Java objekte u njihovu JSON reprezentaciju. Takođe, može da konstruiše Java objekat iz JSON stringa.

Primer upotrebe GSON biblioteke može se videti na sledećoj slici u *serializeImages()* metodi,a vrednost rezultujućeg stringa na listingu 20. u poglavlju *Glavni panel profilske strane*. Ostale metode predstavljaju omotače oko istoimenih metoda Shoe klase, a radi lakšeg pristupa EL sintaksom na JSP strani.

Implementacija ShoeBean klase data je na sledećoj slici:

```
public class ShoeBean{
    private Shoe model;
    private String shoeImagesJSON;
    private SocialFacadeRemote facade;

    private void serializeImages(){
        shoeImagesInJson = new Gson().toJson(shoeImages);
    }

    public void insertComment(){
        String text = FacesUtil.getRequestParameter("comments_form:comment_textarea");
        UserBean currentUser = (UserBean) BeanUtil.getBean("currentUser", UserBean.class);
        String escapedText = escapeCharacters(text);

        CommentEntity comment = new CommentEntity();
        comment.setUser(currentUser.getModel());
        comment.setContent(escapedText);
        comment.setDate(new Timestamp(GregorianCalendar.getInstance().getTimeInMillis()));

        model = facade.writeShoeComment(model, comment);
    }

    public void addReview(){
        String title = FacesUtil.getRequestParameter("review_form:review_title");
        String content = FacesUtil.getRequestParameter("review_form:review_content");
        String escapedContent = escapeCharacters(content);
        UserBean currentUser = (UserBean) BeanUtil.getBean("currentUser", UserBean.class);

        if(title != null)
            title = escapeCharacters(title);

        ArticleEntity article = new ArticleEntity();
        article.setUser(currentUser.getModel());
        article.setTitle(title);
        article.setContent(escapedContent);

        model = facade.writeShoeArticle(model, article);
    }

    private void fetchShoeModel(){
        String shoeTitle = HttpUtil.getRequestParameter("title");
        String color = HttpUtil.getRequestParameter("color");

        String IDparam = extractShoeID(shoeTitle);

        int parsedID = Integer.parseInt(IDparam);
        model = facade.getShoeById(parsedID);
    }

    private String extractShoeID(String input){
        String[] strings = split(input, '-');
        int index = strings.length - 1;
        String id = strings[index];

        return id;
    }
}
```

Listing 10. Implementacija ShoeBean klase

Kako je URL konfigurisan da bude u obliku: "/shoe/#{title}/#{color}", a sam ID cipele se nalazi na kraju *title* stringa, potrebno ga je ekstraktovati i parsirati. Primer URL-a: /shoe/Aetrex-Women's-784/Black

Kako bi se sprečili XSS (Cross-site scripting) napadi, svi unosi koji dolaze od korisnika se čiste pre upisa u bazu. Čišćenje se obavlja pomoću Jsoup[20] biblioteke: Jsoup.clean(input, Whitelist.basic());

Basic lista Jsoup biblioteke dozvoljava prisustvo samo osnovnih HTML tagova: *a, b, blockquote, br, cite, code, dd, dl, dt, em, i, li, ol, p, pre, q, small, strike, strong, sub, sup, u, ul* kao i atrbute ovih tagova.
Ovim je postignuta sigurnost a sa druge strane omogućeno je korisnicima da formatiraju svoj unos komentara i/ili recenzija do neke razumne granice. Sam model se osvežava prilikom upisa svakog komentara ili recenzije, kako bi bio ažuran sa spoljnim promenama.

Slično kao i ShoeBean objekat, UserBean objekat enkapsulira User model i poseduje metode koje kroz Social fasadu menjaju taj model. Neke od tih metoda su provera da li je korisnik ulogovan, metode koje dodaju i brišu trenutnu cipelu iz korisnikove liste želja ili liste cipela koje poseduje, i dr.

SocialFacade klasa

Predstavlja Stateless Session Bean koji sadrži biznis logiku vezanu za socijalni aspekt aplikacije. Metodama se pristupa preko Local interfejsa pošto se sama implementacija nalazi unutar iste Java virtuelne mašine. Definicija SocialFacade interfejsa je predstavljena sledećom slikom:

```
@Local
public interface SocialFacadeRemote {
    public Shoe getShoeById(int id);

    public Shoe writeShoeComment(Shoe shoe, CommentEntity comment);

    public Shoe writeShoeArticle(Shoe shoe, ArticleEntity article);

    public User addToWantList(User user, Shoe shoe);

    public User removeFromWantList(User user, Shoe shoe);

    public User addToHaveList(User user, Shoe shoe);

    public User removeFromHaveList(User user, Shoe shoe);

    public User addToRecentList(User user, Shoe shoe);

    public User removeFromRecentList(User user, Shoe shoe);

    public Shoe incrementShoeViewed(Shoe shoe);

    public Double getVotingResult(int shoeId, int userId);

    public CommentEntity getShoeCommentById(int commentId);

    public Double vote(int shoeId, int userId, double vote);

    public boolean isShoeAlreadyVoted(int userID, int shoeID);

    public int incrementShoeWant(int shoeID);

    public int decrementShoeWant(int shoeID);

    public int incrementShoeHave(int shoeID);

    public int decrementShoeHave(int shoeID);
}
```

Listing 11. SocialFacade interfejs

Sledi implementacija SocialFacade interfejsa (slične i trivijalne metode su izostavljene):

```
public class SocialFacade implements SocialFacadeRemote {

    @PersistenceContext
    private EntityManager em;

    public Double vote(int shoeId, int userId, double vote) {
        Shoe sh = getShoeById(shoeId);

        VotingHistoryPK pk = new VotingHistoryPK(shoeId, userId);
        VotingHistory vh = em.find(VotingHistory.class, pk);

        if(vh != null)
            return sh.getRank();

        sh.setRank(vote);

        createVotingHistory(shoeId, userId, vote);
        return newRank;
    }

    private void createVotingHistory(int shoeId, int userId, double vote) {
        VotingHistoryPK pk = new VotingHistoryPK(shoeId, userId);
        VotingHistory vh = new VotingHistory();
        vh.setPk(pk);
        vh.setVote(vote);
        em.persist(vh);
    }

    public Double getVotingResult(int shoeId, int userId) {
        VotingHistoryPK pk = new VotingHistoryPK(shoeId, userId);
        VotingHistory vh = em.find(VotingHistory.class, pk);
        if(vh == null)
            return 0.0;
        return vh.getVote();
    }

    public Shoe writeShoeComment(Shoe shoe, CommentEntity comment) {
        shoe = em.merge(shoe);
        comment = em.merge(comment);
        shoe.getComments().add(comment);
        return shoe;
    }

    public User addToHaveList(User user, Shoe shoe) {
        shoe = em.merge(shoe);
        user = em.merge(user);
        user.getHasList().add(shoe);
        return user;
    }

    public boolean isShoeAlreadyVoted(int userID, int shoeID) {
        ShoeVotingHistory vh = findPreviousShoeVotingHistory(userID, shoeID);
        return (vh != null);
    }

    private ShoeVotingHistory findPreviousShoeVotingHistory(int userID, int shoeID){
        ShoeVotingHistoryPK pk = new ShoeVotingHistoryPK(userID, shoeID);
        ShoeVotingHistory vh = em.find(ShoeVotingHistory.class, pk);
        return vh;
    }

    public int incrementShoeWant(int shoeID) {
        Shoe shoe = getShoeById(shoeID);
        shoe.incrementWantCount();
        shoe = em.merge(shoe);
        return shoe.getWantCount();
    }
}
```

Listing 12. Implementacija SocialFacade interfejsa

EntityManager se koristi za pravljenje, brisanje, pretragu entiteta, kao i pravljenje upita nad njima (CRUD operacije). Injektuje ga aplikacioni server.

Kako korisnik može samo jednom da oceni neku cipelu potrebno je obezbediti istoriju glasanja za svaki par korisnik/cipela. To je postignuto upotrebom kompozitnog ključa koji se sastoji od jedinstvenog identifikatora korisnika i jedinstvenog identifikatora cipele. Metoda *vote(...)* demonstrira taj mehanizam. VotingHistoryPK entitet predstavlja kompozitni ključ a VotingHistory entitet predstavlja samu vrednost glasa. VotingHistory sadrži VotingHistoryPK kao svoj primarni ključ. Ukoliko EntityManager svojom metodom *find* uspe da nađe VotingHistory pod datim kompozitnim ključem, korisnik je već glasao i metoda vraća postojeću vrednost glasa. Ukoliko VotingHistory ima null vrednost, korisnik nije dao glas trenutnoj cipeli pa treba napraviti istoriju glasanja. Postupak kreiranja istorije glasanja može se videti u *createVotingHistory(...)* metodi. U nastavku sledi implementacija VotingHistory i VotingHistoryPK entiteta.

```java
@Embeddable
public class VotingHistoryPK implements Serializable {

    @Column(name="USER_ID")
    private Integer userId;

    @Column(name="SHOE_ID")
    private Integer shoeId;

    public VotingHistoryPK() {      }

    public VotingHistoryPK(int shoeId, int userId) {
        this.userId = userId;
        this.shoeId = shoeId;
    }

    @Override
    public boolean equals(Object o) {
        if (o == this) {
            return true;
        }
        if ( ! (o instanceof VotingHistoryPK)) {
            return false;
        }
        VotingHistoryPK vhpk = (VotingHistoryPK) o;
        return this.userId.equals(vhpk.userId)
            && this.shoeId.equals(vhpk.shoeId);

    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int hash = 17;
        hash = hash * prime + this.userId.hashCode();
        hash = hash * prime + this.shoeId.hashCode();
        return hash;
    }

    //getters and setters ommited
}
```

```java
@Entity
@Table(name="VOTING_HISTORY")
public class VotingHistory {

    @EmbeddedId
    private VotingHistoryPK pk;

    @Column(name="VOTE")
    private Double vote;

    public VotingHistory() {}

    //getters and setters ommited
}
```

Listing 13. Implementacija VotingHIstoryPK kompozitnog ključa (LEVO), i njegovo korišćenje u sklopu VotingHistory entiteta (DESNO)

Korisnički interfejs

Java Server Faces komponente

Java Server Faces 1.2 pruža mogućnost pisanja sopstvenih komponenti kako bi se olakšao rad na korisničkom interfejsu i kako bi se isti kod iskoristio na nekom drugom mestu u aplikaciji. Pisanje komponenti u JSF 1.2 je veoma neintuitivno i zahteva nekoliko klasa i dosta XML-a kako bi se komponenta osposobila za korišćenje. Proces kreiranja JSF 1.2 komponente biće opisana pomoću SimmilarShoes komponente koja za datu cipelu pravi grafičku predstavu njenih srodnih cipela.

SimmilarShoes komponenta

```
public class SimmilarShoesComponent extends UIComponentBase {
    private ShoeBean shoeBean;
    private Integer shoesToDisplay;

    @Override
    public String getFamily() {
        return "com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesComponent";
    }

    public ShoeBean getShoeBean() {
        if(shoeBean != null){
            return shoeBean;
        }

        ValueExpression valueExpression = getValueExpression("shoeBean");

        if(valueExpression != null){
            ShoeBean value = (ShoeBean) valueExpression.getValue(getFacesContext().getELContext());
            return value;
        } else
            return shoeBean;
    }

    public Integer getShoesToDisplay() {
        if (shoesToDisplay != null) {
            return shoesToDisplay;
        }

        ValueExpression valueExpression = getValueExpression("shoesToDisplay");

        if (valueExpression != null) {
            Integer value = (Integer)valueExpression.getValue(getFacesContext().getELContext());
            return value;
        } else {
            return shoesToDisplay;
        }
    }

    @Override
    public Object saveState(FacesContext facesContext) {
        Object values[] = new Object[2];
        values[0] = super.saveState(facesContext);
        values[1] = shoesToDisplay;
        return values;

    }

    @Override
    public void restoreState(FacesContext facesContext, Object state) {
        Object values[] = (Object[])state;
        super.restoreState(facesContext, values[0]);
        shoesToDisplay = (Integer)values[1];
    }

    //setters ommited
}
```

Listing 14. Implementacija Component klase SimmilarShoes komponente

Klasa sadrži atribute koji će se dodeliti komponenti na JSP strani bilo kroz EL sintaksu ili kao fiksna vrednost. SimilarShoes komponenta sadrži *shoeBean* atribut, koji predstavlja cipelu i *shoesToDisplay* koji određuje broj sličnih cipela koji će se prikazati korisniku. Kako se profilska strana često osvežava AJAX-om, atribute komponente je potrebno sačuvati izmedju AJAX poziva, u suprotnom njihove vrednosti će se izgubiti i komponenta će proizvesti grešku. U *saveState()* metodi sačuvavamo *shoesToDisplay* atribut kako bi prilikom sledećeg osveženja korisničkog interfejsa taj podatak bio dostupan. ShoeBean atribut nije potrebno čuvati jer je on Managed Bean sa View opsegom pa je dostupan dokle god korisnik svojim akcijama osvežava istu stranicu.

Sledeći deo komponente predstavlja SimmilarShoesTag klasa.

```
public class SimmilarShoesTag extends UIComponentELTag {
   private ValueExpression shoeBean;
   private ValueExpression shoesToDisplay;

   @Override
   public String getComponentType() {
      return "com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesComponent";
   }

   @Override
   public String getRendererType() {
      return "com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesRenderer";
   }

   @Override
   protected void setProperties(UIComponent component) {
      super.setProperties(component);

      SimmilarShoesComponent simmilarShoesComponent = (SimmilarShoesComponent) component;

      if(shoeBean != null)
         if(!shoeBean.isLiteralText()){
            simmilarShoesComponent.setValueExpression("shoeBean", shoeBean);
         } else
            simmilarShoesComponent.setShoeBean((ShoeBean)BeanUtil.getBean("shoeBean");

      if(shoesToDisplay != null)
         if (!shoesToDisplay.isLiteralText()) {
            simmilarShoesComponent.setValueExpression("shoesToDisplay", shoesToDisplay);
         } else
            simmilarShoesComponent.setShoesToDisplay(Integer.valueOf(shoesToDisplay.getExpressionString()));
   }
//getters and setters ommited
}
```

Listing 15. Implementacija Tag klase SimmilarShoes komponente

Ova klasa predstavlja sam XHTML tag koji se ugrađuje u JSP stranu. Klasa sadrži atribute sa istim imenom kao i Component klasa ali im je tip ValueExpression što naglašava da ovi parametri taga mogu biti podešeni i kroz EL izraz. *SetProperties()* metoda proverava da li je parametar običan tekst ili EL izraz i u zavisnosti od toga podešava attribute Component klase. Izgled taga u JSP stranici, sa podešenim parametrima može se videti na sledećem listingu:

```
<yas:SimmilarShoes shoeBean="#{currentShoeBean }" shoesToDisplay="12" />
```

Listing 16. XHTML tag SimmilarShoes komponente sa obaveznim parametrima

Parametar *shoesToDisplay* je u ovom slučaju fiksiran na 12, ali bi i on mogao (kao i *shoeBean*), dinamički da se podesi EL izrazom.

Poslednja klasa SimmilarShoes komponente je Renderer klasa. Ova klasa je zadužena za generisanje vizuelnog dela komponente.

```
public class SimmilarShoesRenderer extends AbstractRenderer {

    @Override
    public void encodeBegin(FacesContext context, UIComponent component)
    throws IOException {

        SimmilarShoesComponent simmilarShoesComponent = (SimmilarShoesComponent) component;

        ShoeBean shoeBean = simmilarShoesComponent.getShoeBean();
        Integer shoesToDisplay = simmilarShoesComponent.getShoesToDisplay();
        StringBuilder builder = new StringBuilder();
        List<SearchShoe> similarShoes = shoeBean.getSimilar();

        builder.append("<div class='simmilar_wrapper'>");

        if(similarShoes.size() == 0){
            builder.append("<p style='padding-left: 10px; color:white;'>There are no similar shoes at the moment.</p>");
        }

        else{
            if(similarShoes.size() < shoesToDisplay)
                shoesToDisplay = similarShoes.size();

            for (int i = 0; i < shoesToDisplay; i++) {
                builder.append("<div class='simmilar_shoe_holder'>");

                SearchShoe shoe = simmilarShoes.get(i);
                String url = shoe.getImageURL();
                String shoeUrl = shoe.getDefaultURL();

                    builder.append("<a href="+ shoeUrl + ">");
                        builder.append("<img class='simmilar_shoe_image' src='" + url + "' alt='Simmilar shoe'/>");
                    builder.append("</a>");
                builder.append("</div>");
            }
        }
        builder.append("</div>");

        ResponseWriter writer = context.getResponseWriter();
        String resultHtml = builder.toString();
        writer.write(resultHtml);
    }
}
```

Listing 17. Implementacija Renderer klase SimmilarShoes komponente

Slika 5. Izgled generisanog XHTML-a od strane SimmilarShoesRenderer klase

Sledeći korak predstavlja definisanje Component i Renderer klase u faces-config XML fajlu kako bi JSF znao koju klasu da pozove za koju komponentu:

```
<component>
   <display-name>SimmilarShoesComponent</display-name>
   <component-type>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesComponent</component-type>
   <component-class>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesComponent</component-class>
</component>

<render-kit>
   <renderer>
      <component-family>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesComponent</component-family>
      <renderer-type>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesRenderer</renderer-type>
      <renderer-class>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesRenderer</renderer-class>
   </renderer>
</render-kit>
```

Listing 18. Definisanje Component i Renderer klase SimmilarShoes komponente unutar faces-config konfiguracionog fajla

Na kraju preostaje definisanje TLD (Tag Library Descriptor) fajla. To je XML fajl (sa .tld ekstenzijom) u kome se opisuju atributi taga, da li su opcioni, kog su tipa, kao i navođenje atributa koje svaka komponenta automatski nasleđuje.

Primer nasleđenih atributa su *id* i *rendered*. Da bi se nasleđeni atributi mogli koristiti kao parametri taga, moraju se navesti u TLD fajlu samog taga.

```
<tag>
   <name>SimmilarShoes</name>
   <tag-class>com.youandshoe.ui.components.simmilar_shoes.SimmilarShoesTag</tag-class>
   <body-content>empty</body-content>

   <attribute>
      <name>id</name>
      <required>false</required>
   </attribute>

   <attribute>
      <name>rendered</name>
      <required>false</required>
      <deferred-value>
         <type>boolean</type>
      </deferred-value>
   </attribute>

   <attribute>
      <name>shoeBean</name>
      <required>true</required>
      <deferred-value>
         <type>com.youandshoe.ui.beans.model_adapter.ShoeBean</type>
      </deferred-value>
   </attribute>

   <attribute>
      <name>shoesToDisplay</name>
      <required>true</required>
      <deferred-value>
         <type>java.lang.Integer</type>
      </deferred-value>
   </attribute>
</tag>
```

Listing 19. TLD fajl SimmilarShoes komponente

Glavni panel profilske strane

Glavni panel profilske strane cipele sastoji se od panela koji sadrži galeriju slika. U fokusu je velika slika cipele koja zauzima najveći deo panela. Ispod glavne slike levo nalaze se boje u kojima je cipela dostupna.

Klikom na neku od boja, izvršiće se promena slika koje predstavljaju orijentacije (desno), kao i glavna slika. Prelaskom preko orijentacija promeniće se glavna slika.

Ispod galerija slika nalaze se socijalna dugmad: Facebook like dugme, want dugme kojim registrovani korisnik dodaje cipelu u svoju listu želja, have dugme kojim korisnik dodaje cipelu u listu cipela koje poseduje. Ova dugmad utiču na WantHave komponentu.

Slika 6. Izgled glavnog panela profilske strane

Additional info panel sadrži dodatne informacije o cipeli date od strane proizvođača.

Desno od additional info panela nalaze se informacije o marki i modelu cipele. Ispod njih se nalazi komponenta za glasanje i na kraju komponenta za čitanje komentara, kao i polje za unos novog komentara. Komponente će biti detaljnije opisane u nastavku izlaganja.

Kako bi manipulacija slikama bila trenutna, svi URL-ovi slika se isporučuju klijentu u JSON formatu kada klijent zatraži stranicu. Na sledećoj slici prikazana je globalna Javascript varijabla koja čuva JSON objekat slika. U prvom redu je prikazan uzgled varijable u izvornom kodu JSP strane. EL sintaksom ubacuje se vrednost atributa *JSONImages* ShoeBean klase.

Ta vrednost predstavlja serijalizovanu listu objekata koji reprezentuju slike cipele. Serijalizacija se obavlja pomoću GSON biblioteke a rezultat je string u JSON formatu. Deo JSON stringa prikazan je na sledećoj slici:

```
//JSP source
var jsonColors = '<h:outputText value="#{currentShoeBean.JSONImages}" escape="false"/>';

//Parsed result
var jsonColors = [
                {"images":
                  [
                    {"smallUrl":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL75_.jpg",
                      "mediumUrl":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL160_.jpg",
                      "largeUrl":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL.jpg",
                      "searchVariant":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL170_.jpg",
                      "trendingVariant":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL32_.jpg",
                      "profileLargeVariant":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL356_.jpg",
                      "profileSmallVariant":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL38_.jpg",
                      "dayWeekMonthVariant":"http://ecx.images-amazon.com/images/I/51-pA%2BTn0PL._SL170_.jpg"
                    }
                  ],
                  "color": "Navy"
                }
              ]
```

Listing 20. Rezultat serijalizovanja Java objekta u JSON format upotrebom GSON biblioteke

Javascript logika zadužena za promenu boje data je na sledećoj slici:

```
function appendShoeColors(){
    var colorHolder = $('.color_h_items');
    var colorLength = jsonColors.length;

    if(colorLength == 1){
        var info = $('<div/>').text('Only in this color.').css('font-size', '0.7em');
        colorHolder.append(info);

        var large = jsonColors[0].images[0].profileLargeVariant;
        appendLargeImage(large);

        return;
    }

    $.each(jsonColors, function(index, elem) {
        var img = elem.images[0].profileSmallVariant;
        var colorName = elem.color;
        var div = $('<div/>', {'class': 'small_image_holder', click: function(){
            appendShoeOrientations(colorName);
            appendLargeImage(elem.images[0].profileLargeVariant);
        }});

        var imgElem = $('<img/>',{src: img, title: colorName});

        colorHolder.append(div.append(imgElem));

        if(colorName === currentColor){
            var largeImg = elem.images[0].profileLargeVariant;
            appendLargeImage(largeImg);
        }
    });
}
```

Listing 21. Logika zadužena za promenu slika pri izboru druge boje cipele

Funkcija iterira kroz niz boja trenutne cipele pomoću jQuery *each()* funkcije. *U* svakoj iteraciji, each() funkcija proizvodi dve vrednosti, indeks iteracije (*index* varijabla) i sam element trenutne iteracije (*elem* varijabla).

Varijabla *elem* je objekat koji sadrži ime boje u svom *color* polju i listu objekata koji sadrže URL-ove svih orijentacija cipele,u različitim veličinama, u *images* polju.

Varijabla *div* sadrži novokreirani HTML element, sa 'small_image_holder' CSS klasom i sa funkcijama *appendShoeOrientations()* i *appendLargeImage()* koje se aktiviraju kada se klikne na sam element. Ovaj element predstavlja omotač oko same slike koja se kreira *imgElem* varijablom.

Ostale funkcije date su sledećom slikom:

```
function appendShoeOrientations(color){
    var orientationsHolder = $('.orient_items').empty();
    var colorObject;
    var colorFound = false;

    $.each(jsonColors, function(index, elem){
        if(elem.color === color){
            colorObject = elem;
            colorFound = true;
            return false;
        }
    });

    if(!colorFound)
        colorObject = jsonColors[0];

    $.each(colorObject.images, function(index, elem){
        var imgURL = elem.profileSmallVariant;

        var div = $('<div/>', {'class': 'small_image_holder', mouseenter: function(){
            appendLargeImage(elem.profileLargeVariant);
        }});

        var img = $('<img/>',{src: imgURL, 'class': 'small_image'});

        orientationsHolder.append(div.append(img));
    });
}

function appendLargeImage(imageURL){
    var image = $('<img/>',{src: imageURL, 'class': 'main_img'});
    $('#shoe_image').empty().append(image);
}
```

Listing 22. Logika zadužena za dodavanje slika orijentacija i glavne slike cipele

Funkcija *appendShoeOrientations()* na osnovu izabrane boje pravi odgovarajuće orijentacije slika,a *appendLargeImage()* na osnovu datog URL-a pravi <img/> element i ubacuje ga u držač glavne slike.

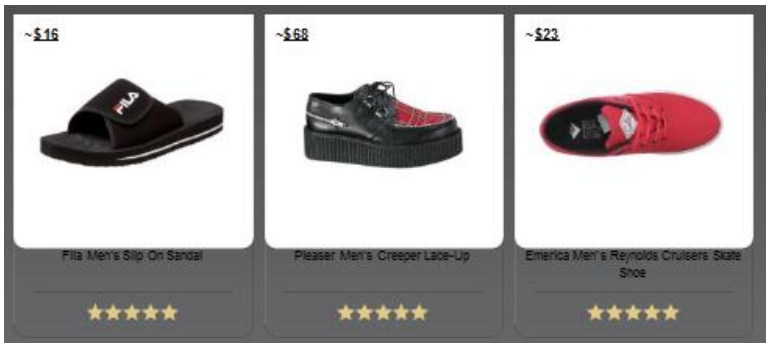
ShoeRating komponenta

Ova komponenta se može naći u nekoliko različitih stanja. Ukoliko korisnik nije ulogovan, komponenta će prikazati kumulativnu ocenu cipele. Ukoliko je ulogovan a nije glasao, komponenta će prikazati kumulativnu ocenu, a ispod toga će stajati mogućnost unosa ocene. Ukoliko je korisnik ulogovan i glasao je, komponenta će prikazati kumulativnu ocenu, a ispod toga ocenu koju je korisnik dao. Poslednje stanje određuje se *embedded* parametrom. Ukoliko je vrednost ovog parametra *true*, to znači da se komponenta ugrađuje u rezultate pretrage. Opisana stanja su prikazana sledećim slikama:

Slika 7. Izgled komponente ukoliko korisnik nije ulogovan.

Slika 8. Izgled komponente ukoliko je korisnik ulogovan i nije glasao.

Slika 9. Izgled komponente ukoliko je korisnik ulogovan i dao je glas.



Slika 10. Izgled komponente u *embedded* stanju.

Tag ShoeRating komponente:

```
<yas:ShoeRating rating="#{currentShoeBean.rank}" embedded="false" />
```

Listing 23. XHTML tag ShoeRating komponente

Mogućnost glasanja postignut je kombinacijom jQuery komponente,koja kreira efekat pri prelasku preko zvezdica za glasanje i klikom na poslednje izabranu zvezdicu, i Richfaces taga koji poziva metodu zaduženu za glasanje na serverskoj strani.

```
function bindRating(){
   jQuery("input[type=radio].star").rating({
      callback: function(){
         var checkedStars = jQuery("div.star-rating-on").length;

         if(isNaN(checkedStars) || checkedStars == 0)
            return;

         submitRating(checkedStars);
      }
   });
}

<h:form>
   <a4j:jsFunction immediate="true" name="submitRating"
      reRender="rating_holder_id" action="#{currentUser.rateCurrentShoe }">

      <a4j:actionparam name="starsRated"
         assignTo="#{currentUser.starsRated }" />
   </a4j:jsFunction>
</h:form>
```

Listing 24. Kombinovana upotreba jQuery komponente i RichFaces <a4j:jsFunction/> taga

Funkcija *bindRating()* poziva jQuery komponentu koja pravi korisnički interfejs za glasanje. Kada se glasanje obavi ova funkcija će pozvati *submitRating()* funkciju i proslediti joj rezultat glasanja.

Funkcija *submitRating()* je ustvari komponenta <a4j:jsFunction> Richfaces biblioteke koja se u Javascript-u može pozvati kao obična funkcija i kojoj se mogu proslediti parametri. <a4j:jsFunction> će pozvati odgovarajuću metodu na serveru, proslediti parametar glasanja i osvežiti samu komponentu stavljajući je u novo stanje.

Renderer klasa ShoeRating komponente:

```
public class ShoeRatingRenderer extends AbstractRenderer {

    @Override
    public void encodeBegin(FacesContext context, UIComponent component)
    throws IOException {

        ShoeRatingComponent shoeRatingComponent = (ShoeRatingComponent)component;
        Double rating = shoeRatingComponent.getRating();

        UserBean currentUser = (UserBean) BeanUtil.getBean("currentUser", UserBean.class);
        ShoeBean currentShoe = (ShoeBean)BeanUtil.getBean("currentShoeBean", ShoeBean.class);
        double previousVote = socialFacade.getVotingResult(currentShoe.getShoeID(), currentUser.getId());

        boolean loggedIn = currentUser.isLoggedIn();
        boolean embedded = shoeRatingComponent.getEmbeded();
        boolean alreadyVoted = (previousVote != 0.0);

        StringBuilder builder = new StringBuilder();
        String img = "silhouette_child_";

        if(loggedIn)
            img += "small_";

        //read-only rating
        builder.append("<div class='rating' id='average_rating_id'>");

        for (int i = 1; i <= 5; i++) {
            builder.append("<img src=\"/images/" + img);
            if (rating >= i) {
                builder.append("filled");
            } else {
                builder.append("empty");
            }
            builder.append(".png\" />");
        }

        builder.append("</div>");
        //end of read-only rating

        //spacer
        if(!embedded)
            builder.append("<hr style='float: left; color: #141215;"
                + "width: 252px; height: 1px; margin-bottom: 5px; background-color: #141215;' />");

        //rating input
        builder.append("<div class='rating_user' " +
            (alreadyVoted == true ? "style='padding-left: 84px;'" : "") + ">");

        if(loggedIn && !embedded){
            for (int i = 1; i <= 5; i++)
                builder.append("<input name='rating_star' type='radio' " +
                    (alreadyVoted == true ? "disabled='disabled'" : "") + " class='star' " +
                    (previousVote >= i ? "checked='checked'" : "") + "/>");
        }
        builder.append("</div>");
        //end of rating input

        ResponseWriter writer = context.getResponseWriter();
        String resultHtml = builder.toString();
        writer.write(resultHtml);
    }
}
```

Listing 25. Renderer klasa ShoeRating komponente

CommentsView komponenta

U zavisnosti od *commentType* parametra, komponenta prikazuje sve komentare trenutne cipele (*all* parametar), komentare prijatelja trenutno ulogovanog korisnika (*friends* parametar) ili komentare ostavljene od strane profesionalnih recenzenata (*pros* parametar).

Komponenta je u tri primerka stavljena u Richfaces tabPanel komponentu čime je postignut efekat straničenja. Izvorni kod dela JSP strane koji se odnosi na ovu komponentu može se videti na sledećoj slici:

```
<rich:tabPanel switchType="client" id="comment_tab_panel_id">
   <rich:tab label="All" id="all_tab_id" name="all_tab">
      <yas:CommentsViewTag currentUserBean="#{currentUser}"
         shoeBean="#{currentShoeBean}" commentType="all"
         rendered="#{not empty currentShoeBean.comments }" />
   </rich:tab>
   <rich:tab label="Friends" rendered="#{currentUser.loggedIn }"
      id="friends_tab_id" name="friends_tab">
      <yas:CommentsViewTag currentUserBean="#{currentUser}"
         shoeBean="#{currentShoeBean}" commentType="friends"
         rendered="#{currentUser.loggedIn and not empty currentShoeBean.commentsByFriends}" />
   </rich:tab>
   <rich:tab label="Pros" id="pros_tab_id" name="pros_tab">
      <yas:CommentsViewTag currentUserBean="#{currentUser}"
         shoeBean="#{currentShoeBean}" commentType="pros"
         rendered="#{not empty currentShoeBean.commentsByPros }" />
   </rich:tab>
</rich:tabPanel>
```

Listing 26. Upotreba CommentsView komponente

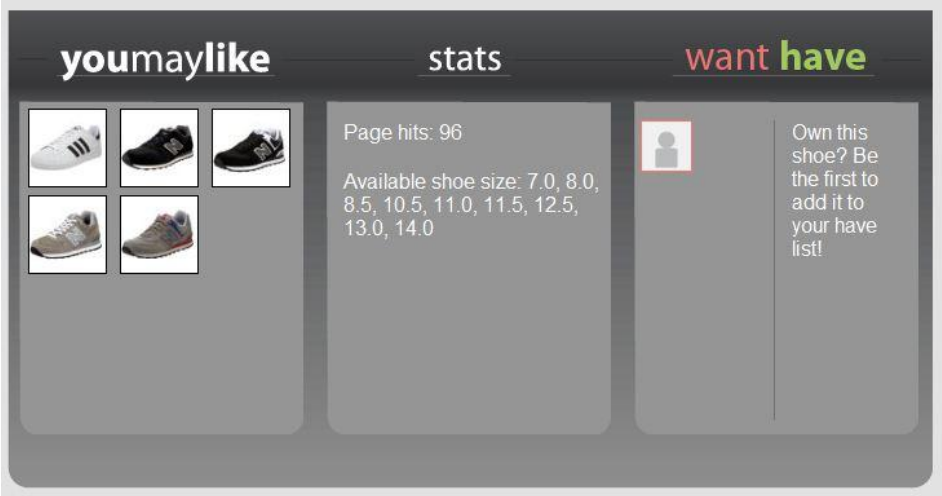
Izgled komponente se može videti na sledećoj slici:

Sam unos komentara predstavlja odvojenu celinu. Unosom komentara i pritiskom na dugme Post it, poziva se metoda ShoeBean objekta koja dodaje nov komentar i osvežava CommentsView komponentu. Ukoliko je broj komentara veći od 5, komponenta će sa desne strane prikazati traku za skrolovanje. Kako je Richfaces tabPanel komponenta podešena da promenu tabova radi na klijentskoj strani, sama CommentsView komponenta će osvežiti svoj sadržaj samo u slučajevima kada korisnik ponovo osveži celu stranicu ili doda nov komentar.

Slika 11. Izgled CommentsView komponente

Središnji panel profilske strane

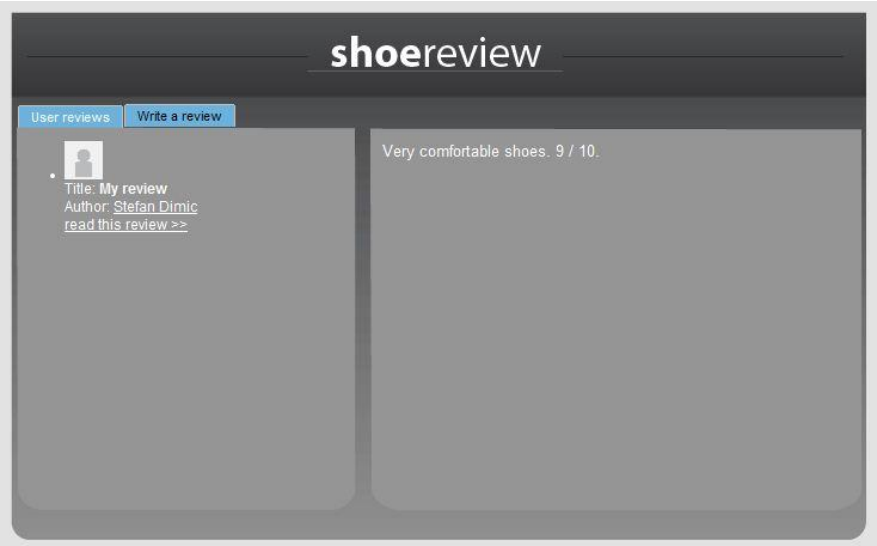
Središnji panel se sastoji od SimmilarShoes komponente, zatim panela koji sadrži informacije kao što su broj poseta trenutnoj stranici, veličina u kojoj je dostupna cipela i slično. Treći panel sadrži jednostavnu komponentu koja je zadužena za prikaz korisnika koji su trenutnu cipelu dodali u svoju listu želja i/ili listu cipela koje poseduju. Sledi prikaz središnjeg panela:



Slika 12. Središnji panel profilske strane

Panel za recenzije

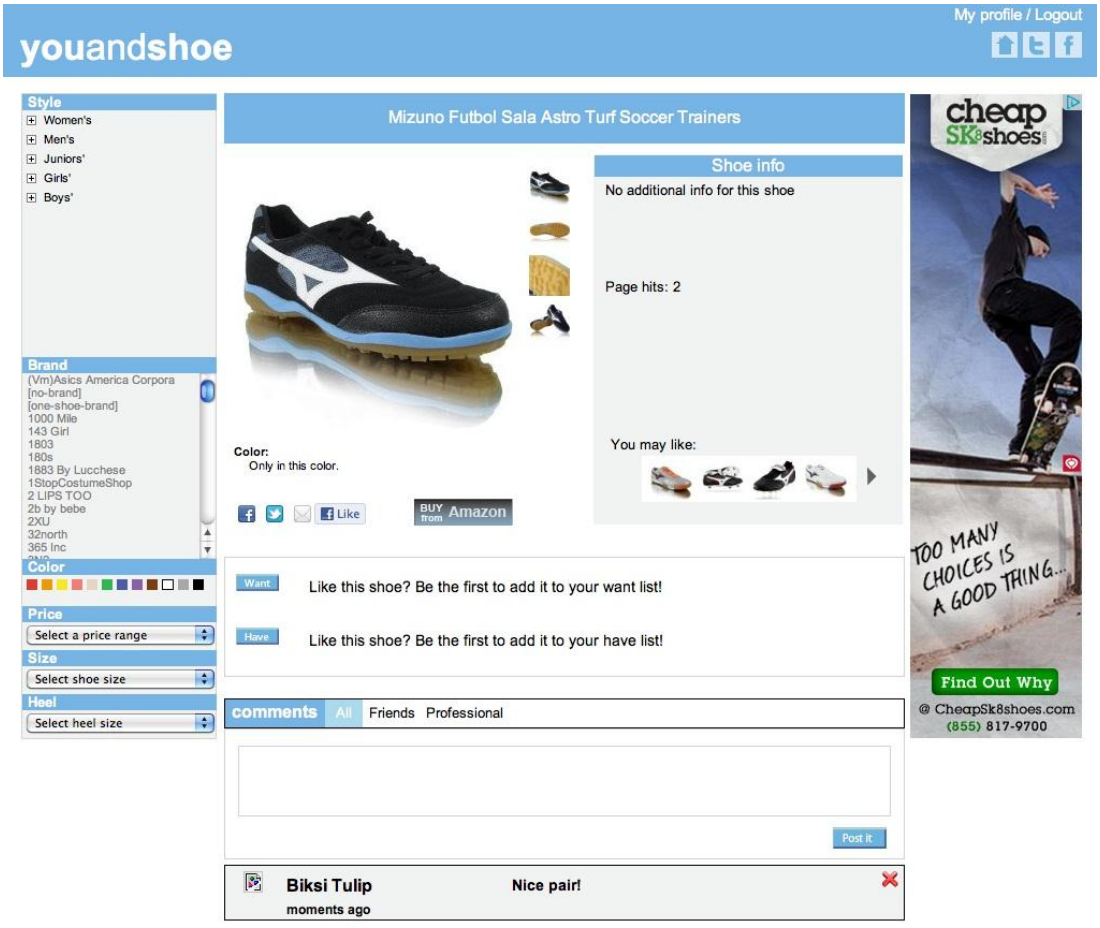
Poslednji panel profilske strane predstavlja panel za recenzije, podeljen u dva tabovana panela. Prvi panel sadrži recenzije korisnika koje trenutni posetilac može da čita u okviru istog taba. Drugi panel omogućuje ulogovanom korisniku da napiše sopstvenu recenziju.



Slika 13. Izgled panela za čitanje recenzija

Slika 14. Izgled panela za pisanje recenzija

Migracija



Slika 15. Nov dizajn profilske strane

Kako se Java Server Faces 1.2 implementacija pokazala kao veoma neintuitivna i teška za razvoj web aplikacija, napravljena je migracija ka Java Server Faces 2.0 kao i Java EE 6 platformi.

Glavne zamerke koje se odnose na JSF 1.2 su:

- Komplikovan process pravljenja i održavanja komponenti
- Nedostatak ugrađene podrške za AJAX
- Korišćenje JSP tehnologije
- Ograničenost EL sintakse

Proces pravljenja sopstvene JSF komponente je opisan na primeru SimilarShoes komponente. Proces obuhvata kreiranje 3 Java klase i pisanje XML atributa 2 različita fajla (faces-config.xml i tld fajl same komponente). Povećanjem broja komponenti, broj klasa i xml fajlova raste linearno pa se održavanje istih pretvara u noćnu moru.

Kako JSF 1.2 ne poseduje nikakvu ugrađenu podršku za AJAX, potrebno je dodati neku spoljašnju biblioteku koja bi rešila ovaj problem. Richfaces biblioteka pruža odličnu AJAX podršku kako u obliku vizuelnih tako i nevizuelnih komponenti, ali se plaća velika cena u veličini stranice koja se isporučuje korisniku. Kako bi funkcionisao Richfaces u stranice ugrađuje sopstven kod kao i css fajlove u slučaju korišćenja vizelnih komponenti.

Veličina biblioteka učitanih od strane Richfaces-a može da naraste i preko 500KB što znatno usporava učitavanje stranice.

Iako je JSP tehnologija veoma moćna, ona ne leži dobro unutar JSF ekosistema. JSP ima veoma ograničenu podršku za pravljenje šablona (templating), <jsp:include> je jedini tag kojim se može postići šablonizovanje stranica (njime se jedna JSP stranica ugrađuje u drugu). Ovime su developeri osuđeni na pisanje sopstvenih komponenti u čistom Java kodu kako bi postigli neku rudamentaran oblik šablona, što se pokazalo kao veoma naporan posao.

Expression Language koji dolazi uz JSF 1.2 je veoma jednostavan za korišćenje i svoj posao dobro obavlja, ali je veoma ograničen u pogledu svoje sintakse jer podržava samo zvanje metoda Managed Bean-ova koje nemaju parametre. Taj nedostatak može se videti na potrebi za formatiranjem nekih podataka Managed Bean-a za prezentaciju. Kako ti parametri ne mogu da se proslede EL sintaksom u neku metodu, ostaje samo da se parametri formatiraju unutar Managed Bean-a, što dovodi do prljanja samog Bean-a jer on ne bi trebao da bude zadužen za formatiranje prikaza. Drugi pristup bi bilo pisanje komponente koja će obaviti formatiranje, što je opet zamoran proces.

Java Server Faces 2

JSF 2 pokriva sve nedostatke koje JSF 1.2 verzija poseduje.

Kako JBoss 6 aplikacioni server interno poseduje potrebne JSF 2 biblioteke, one će automatski biti dodate na classpath svake JSF aplikacije tokom njenog deploy-ovanja na server. JBoss će prepoznati web aplikaciju kao JSF aplikaciju ukoliko je neki od navedenih uslova ispunjen:

- FacesServlet proglašen u WEB-INF/web.xml fajlu
- faces-config.xml fajl je prisutan u WEB-INF folderu aplikacije
- faces-config.xml fajl se nalazi u META-INF folderu neke biblioteke koja se nalazi u WEB-INF/lib folderu aplikacije
- *.faces-config.xml fajl se nalazi u META-INF folderu neke biblioteke koja se nalazi u WEB-INF/lib folderu aplikacije
- `javax.faces.CONFIG_FILES` kontekstualni parametar je definisan u WEB-INF/web.xml fajlu
- `org.jboss.jbossfaces.JSF_CONFIG_NAME` parametar je definisan u WEB-INF/web.xml fajlu
- "alwaysAddJSF" parametar je podešen na **true** u jsf-integration-deployer-jboss-beans.xml fajlu

YouAndShoe JSF projekat je konfigurisan preko faces-config.xml fajla koji se nalazi u WEB-INF folderu projekta, a sadržaj samog fajla je dat na sledećem listingu:

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

<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
</faces-config>
```

Listing 27. Minimalan sadržaj faces-config.xml fajla potrebnog za identifikovanje web projekta kao JSF projekta

Korišćenje XML-a je svedeno na minimum i zamenjeno anotacijama. Proglašavanje Java klasManaged Bean-om se u JSF 1.2 svodilo na pisanje nekoliko linija XML koda:

```
<managed-bean>
        <managed-bean-name>currentShoeBean</managed-bean-name>
        <managed-bean-class>com.beans.ShoeBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Ovo je zamenjeno stavljanjem anotacija iznad same Java klase:

```
@ManagedBean
@RequestScoped
Public ShoeBean{…}
```

JSF 2 donosi i view opseg, anotiran pomoću @ViewScoped. Ovime se eliminiše potreba za simuliranjem View opsega pomoću eksternih biblioteka kao što je Richfaces <a4j:keepAlive/> tag.

JSF 2 poseduje sopstvenu AJAX implementaciju:

```
<h:form>
    <h:inputText value="#{bean.text}" >
        <f:ajax event="keyup" render="text"/>
    </h:inputText>

    <h:outputText id="text" value="#{bean.text}" />
</h:form>
```

Listing 28. Primer upotrebe JSF 2 ugrađene AJAX podrške

Novi <f:ajax/> tag omogućava slanje AJAX zahteva od strane bilo koje komponente. Atribut *event* <f:ajax/> taga određuje na koji će ce događaj aktivirati ajax zahtev. U gornjem primeru ajax se aktivira na pritisak tastera - *keyup.* Atribut *render* sadrži id-eve komponenti čiji će se prikaz osvežiti kada stigne AJAX odgovor. Vrednosti *render* atributa date su sledećom tabelom:

Moguće vrednosti	Opis
@all	Osveži sve komponente na strani
@none	Ništa se ne osvežava
@this	Osveži samo komponentu koja je inicirala AJAX zahtev
@form	Osveži sve komponente unutar forme iz koje je potekao AJAX zahtev
id	id jedne ili više komponenti koje treba osvežiti
EL	EL izraz koji se mora razrešiti u kolekciju stringova koja sadrži id-eve komponenti

Tabela 1. Pregled mogućih vrednosti *render* parametra <f:ajax> taga

<f:ajax/> tag sadrži i *execute* atribut kojim se mogu specificirati komponente forme koje će biti poslate serveru na procesuiranje. Atribut može imati iste vrednosti kao i *render* atribut. Ovim tagom se eliminiše potreba za Richfaces ajax bibliotekom.

Nova EL sintaksa omogućava prosleđivanje proizvoljnog broja parametara (proizvoljnog tipa) u metode Managed Bean-ova:

```
<h:form>
  <div class='want_have_title'>

    <h:commandButton value="Have"
      disabled="#{not userSession.loggedIn or userSession.isInHaveList(shoeProfile.id)}"
      action="#{shoeProfile.addToHaveList(shoeProfile.id)}">
        <f:ajax execute="@form" render="@form" />
    </h:commandButton>

  </div>
</h:form>
```

Listing 29. Ilustracija unapređene EL sintakse

Metode zadužene za formatiranje podataka mogu se pomeriti u specijalizovane managed bean klase i lako se pozivati EL sintaksom:

```
<ui:repeat value="#{resultPage.shoes}" var="shoe">
   <h:outputLink value="#{shoe.link}">
      <h:graphicImage value="#{shoe.image}" title="#{shoe.title}"
         class='shoe_image' />
   </h:outputLink>

   <div class='title_holder'>
      <h:outputText
         value="#{shoeUtil.formatPresentationName(shoe.title, 57, '.')}" />
   </div>

   <div class='rank_holder'>
      <h:outputText
         value="#{rankUtil.formatPresentationRank(shoe.rank)}" />
   </div>
</ui:repeat>
```

Listing 30. Formatiranje podataka za prezentaciju upotrebom unapređene EL sintakse

Facelets

Inicijalno, Faceleti[21] su bili odvojeni projekat i predstavljali su alternativu za pravljenje korisničkog interfejsa za Java Server Faces 1.1 i 1.2, koji su oboje koristili JSP kao podrazumevanu tehnologiju za izgradnju korisničkog interfejsa.

Počev od JSF 2.0, Faceleti postaju podrazumevana tehnologija za izgradnju korisničkog interfejsa. JSP tehnologija je proglašena zastarelom i zaostavštinom.

Faceleti podržavaju upotrebu šablona. Facelet fajl može da referencira master šablon i da ubaci sadržaj u 'držače' koji su definisani u master šablonu. Korišćenje šablona je demonstrirano na sledećoj slici:

```
<!--master_template.xhtml-->
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets">

   <h:head></h:head>

   <h:body>
      <div id='header'>Header</div>

      <div id='main'>
         <ui:insert name="body_content" />
      </div>


      <div id='footer'></div>
   </h:body>
</html>

<!--client_template.xhtml-->
<ui:composition template="/templates/master_template.xhtml"
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets">

   <ui:define name="body_content">
      Ovo ce biti utisnuto u body_content placeholder
   </ui:define>
</ui:composition>
```

Listing 31. Primer master šablona i njegovo korišćenje u klijentskom Faceletu

Pored šablona, Faceleti pružaju mogućnost ponovnog iskorišćavanja nekog dela korisničkog interfejsa tako što omogućavaju korisniku da uključi neki sadržaj, koji se nalazi u drugom fajlu. Uključivanje sadržaja se može obaviti na 3 načina:

- Referenciranjem fajla
- Pravljenjem sopstvenih tagova
- Kompozitnim komponentama

Najjednostavniji način da se iz jednog Facelet fajla uključi sadržaj drugog Facelet fajla je referenciranjem po imenu fajla upotrebom <ui:include /> taga. Sadržaj Facelet fajla koji se ugrađuje u drugi fajl mora biti obavijen <ui:composition></ui:composition> tagom. Prilikom ugrađivanja, samo sadržaj unutar <ui:composition/> taga će biti uključen u stranicu.

Weld

Weld predstavlja referentnu implementaciju JSR-299 specifikacije - novog Java standarda za injekciju resursa i kontekstualnog upravljanja životnog ciklusa objekata (takozvani CDI). Objekat vezan za neki životni ciklus zove se bean. CDI ima ugrađenu podršku za nekoliko tipova bean-ova, između ostalog i sledeće java EE komponente:

- Managed bean-ove
- EJB session bean-ove

Kako managed bean-ovi tako i EJB session bean-ovi mogu da injektuju druge bean-ove. Ali i objektima koji nisu bean-ovi mogu se injektovati bean-ovi pomoću CDI. U Java EE platformi sledećim komponentama se mogu injektovati bean-ovi:

- Message-driven bean-ovi
- Interceptori
- Servleti, filteri i servlet event listeneri
- JAX-WS service endpoint-i i njihovi handleri

JBoss 6 aplikacioni server interno poseduje Weld implementaciju tako da je dovoljno dodati beans.xml fajl u META-INF folder ili u WEB-INF folder web aplikacije, a JBoss će obaviti sav posao oko uključivanja potrebnih Weld biblioteka. Minimalan sadržaj beans.xml fajla potrebnog za aktivaciju Weld-a dat je na sledećem listingu:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://docs.jboss.org/cdi/beans_1_0.xsd">

</beans>
```

Listing 32. Minimalan sadržaj beans.xml fajla potrebnog aktiviranje Weld-a

Iako je gore opisan beans.xml fajl sasvim dovoljan za korišćenje Weld-a, u njemu se mogu definisati i napredne mogućnosti kao što su presretači, dekoratori i alternative. Primer sadržaja takvog beans.xml fajla dat je na sledećem listingu:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://docs.jboss.org/cdi/beans_1_0.xsd">

  <interceptors>
    <class>com.acme.intercept.SecurityInterceptor</class>
  </interceptors>

  <decorators>
    <class>com.acme.decorate.SpecialGiftDecorator</class>
  </decorators>

  <alternatives>
    <class>com.acme.business.MockPaymentProcessor</class>
  </alternatives>
</beans>
```

Listing 33. Beans.xml fajl sa definisanim presretačem, dekoratorom i alternativom

Detaljniji primeri vezani za presretače, dekoratore i alternative se mogu pogledati u Weld-ovoj dokumentaciji u poglavljima Interceptors[23], Decorators[24] i Alternatives[25].

Pojavom Weld-a, JSF 2.0 @ManagedBean anotacija zastareva i uglavnom se koristi u scenarijima u kojima CDI nije dostupan.

Primer managed bean-a pod kontrolom Weld-a, dat je sledećom slikom:

```
@Named("userSession")
@SessionScoped
public class UserSession implements Serializable {

    private Integer id;

    private User model;

    private boolean loggedIn;

    @Inject
    private SocialFacadeRemote f;

    public void fireUpModelUpdate() {
        model = f.getUserById(id);
    }

    @PostConstruct
    private void init() {
        loggedIn = false;
        id = -1;
    }

    public void logInUser(int userID) {
            this.id = userID;
            this.loggedIn = true;
            fireUpModelUpdate();
    }

    public void logOut(){
        loggedIn = false;

        HttpSession session = HTTPUtil.getCurrentSession();
        session.invalidate();
    }
}
```

Listing 34. Primer bean-a pod kontrolom Weld-a

@Named anotacijom daje se ime bean-u pomoću kojeg se on može referencirati u JSP strani ili Faceletu pomoću EL sintakse. @Named anotacija nije marker koji označava klasu kao managed bean, Weld će sam prepoznati klase koje imaju odličja bean-ova, a nalaze se unutar aplikacije pod kontrolom Weld-a.

Sposobnosti Weld-a demonstrirane su sledećim primerom:

```
@Named("shoeProfile")
@ViewAccessScoped
public class ShoeProfilePage implements Serializable {

    private String id;

    private String title;

    private String color;

    @Inject
    private UserSession userSession;

    @Inject
    private SocialFacadeRemote f;

    @PostConstruct
    public void init() {
        title =FacesUtil.getRequestParameter("title")
        color =FacesUtil.getRequestParameter("color");
        id = FacesUtil.getRequestParameter("id");

    }

    public void addToRecentList() {
        f.addToRecentList(userSession.getId(), Integer.parseInt(id));
        userSession.fireUpModelUpdate();
    }

    //other fields and methods ommited
}
```

Listing 35. Primer injekcije resursa pomoću Weld-a

ShoeProfilePage klasa je ekvivalent ranije opisane ShoeBean klase. UserSession klasa je managed bean sa session opsegom koji predstavlja trenutnog korisnika. SocialFacadeRemote predstavlja interfejs statless session bean-a. Weld inicijalizaciju ovih parametra svodi na jednu anotaciju *(@Inject*), nema potrebe za vađenjem managed bean-ova iz mapa opsega ili obavljanja lookup-a kroz JNDI za session bean-ove. Nema potrebe ni za eksplicitnim instanciranjem objekata preko *new* operatora jer Weld može da injektuje svaku klasu koja ima konstruktor bez parametara.

Ali ne može se sve što se želi injektovati svesti na bean klasu instanciranu od strane kontejnera preko *new*. Nekad je potreban dodatan stepen kontrole. Producer metoda je metoda koji ima ulogu kreatora instanci bean-ova. Sama deklaracija metode opisuje bean i kontejner poziva metod kako bi došao do nove instance, ukoliko instanca ne postoji u specificiranom kontekstu. Producer metoda se deklariše stavljanjem *@Produces* anotacije iznad neke metode bean klase.

Primer Producer metode:

```
public class RandomNumberGenerator {

  private Random random = new Random(System.currentTimeMillis());

  @Produces @Named @Random int getRandomNumber() {
    return random.nextInt(100);
  }

}
```

Listing 36. Weld Producer metoda

Metoda *getRandomNumber()* je označena kao Producer metoda, *@Named* anotacija omogućava pristup preko EL-a, *@Random* anotacija je proizvoljno definisana anotacija koja će naslediti *@Dependent* opseg, što znači da će životni ciklus injektovane promenjive biti vezan za bean u koji je injektovana. Sada se slučajno generisan broj može injektovati bilo gde kao*: @Inject @Random* int randomNumber;

Čak i u EL izrazu:

```
<p>generisan broj: #{randomNumber}.</p>
```

Pored opisanih mogućnosti, Weld sadrži još naprednih opcija, kao što su alternative, dekoratori, stereotipi i mnoge druge.

Zaključak

Iako sa očiglednim ograničenjima i nedostacima, Java Server Faces 1.2 framework pokazao se kao sofisticiran alat za razvoj enterprise aplikacija. Dolaskom Java Server Faces 2, usvajanjem Faceleta kao oficijelne tehnologije za izgradnju korisničkog interfejsa i unapređenjem EL sintakse, razvoj aplikacija je podignut na novi nivo kako u kvalitetu alata tako i u kvalitetu samog koda.

JSR-299 specifikacija i njena referentna implementacija (Weld) predstavljaju veliki napredak u enterprise tehnologijama pa je preporuka koristiti ih bez rezerve.

Java Server Faces se pokazao kao tehnologija koja može da se uhvati u koštac sa složenim projektima kao što je YouAndShoe socijalna mreža. Sofisticiranost tehnologija koje okružuju JavaEE i Java Server Faces, kao i veliko bogatstvo informacija na Internetu stavlja Javu i njene tehnologije u prve redove kada je u pitanju izbor tehnologija za implementaciju web i/ili enterprise tehnologija.

Reference

1. http://docs.oracle.com/javaee/6/tutorial/doc/
2. http://www.jboss.org/jbossas/docs/
3. http://www.oracle.com/technetwork/java/overview-141217.html
4. http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html
5. http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html
6. http://docs.oracle.com/javaee/6/tutorial/doc/bnagb.html
7. http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html
8. http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
9. http://docs.oracle.com/javaee/6/tutorial/doc/bnatx.html
10. http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html
11. http://java.sun.com/products/jsp/reference/techart/unifiedEL.html
12. http://www.json.org/
13. http://en.wikipedia.org/wiki/Ajax_(programming)
14. http://docs.jboss.org/richfaces/latest_4_1_X/Developer_Guide/en-US/html/
15. http://jquery.com/
16. http://en.wikipedia.org/wiki/Session_Beans
17. http://docs.jboss.org/hibernate/core/4.0/devguide/en-US/html_single/
18. http://ocpsoft.com/docs/prettyfaces/3.3.2/en-US/html_single/
19. https://sites.google.com/site/gson/gson-user-guide
20. http://jsoup.org/
21. http://facelets.java.net/nonav/docs/dev/docbook.html
22. http://docs.jboss.org/weld/reference/latest/en-US/html/
23. http://docs.jboss.org/weld/reference/latest/en-US/html/interceptors.html
24. http://docs.jboss.org/weld/reference/latest/en-US/html/decorators.html
25. http://docs.jboss.org/weld/reference/latest/en-US/html/injection.html#alternatives