Univerzitet u Beogradu

Matematički fakultet

MASTER RAD

Primena JSF tehnologije u izradi web aplikacije za opštinski info i uslužni servis

Mentor:
Prof. dr. Dušan Tošić

Student:
Ivana Srbljanin

Beograd, jul 2012

Sadržaj:

Apstrakt

Java Server Faces tehnologija predstavlja Java aplikativno okruženje, koje za osnovni cilj ima pojednostvaljeni i brži razvoj web aplikacija sa bogatim korisničkim interfejsom. Arhitektura JSF–a integriše MVC dizajnerski obrazac,kojim se domen rada razdvaja u 3 sloja: Model, View i Controller. Svaki sloj implementiran je jednom od tehnologija: JavaBeans, Java Server Pages, Java Servlet. Ovakva arhitektura omogućava da se svi koji učestvuju u razvoju softvera, a koriste ovu tehnologiju, fokusiraju na svoj deo posla. S obzirom da se za kreiranje pomenutih kompleksnih korisničkih interfejsa koriste komponente, koje se ugrađuju u web stranice, može se reći i da je JSF komponentno-zasnovano okruženje (framework).
U radu je iskorišćena Java Server Faces tehnologija za izradu Web-aplikacije pomoću koje se mogu dobiti razne informacije (o restoranima, pozorištima, izložbama,…) vezane za beogradske opštine. Pored dobijanja informacija, aplikacija pruža i neke usluge (naručivanje hrane, rezervacija karata, …) korisniku. Ova aplikacija omogućava da se demonstriraju neke mogućnosti Java Server Faces tehnologije.

Abstract

Java Server Faces technology is Java application framework of which the main purpose is to simplify and speed development of web applications that have rich user interface. The architecture of JSF integrates MVC design pattern, so that the work domain is separated into three layers: Model, View and Controller. Each layer is implemented by one of the following technologies: JavaBeans, Java Server Pages and Java Servlet. This type of architecture enables that every team member, taking part in software development, focuses on his own work. For creation of mentioned rich user interfaces, components are used and embedded into web pages, hence, it could be said that JSF is component-oriented web framework.
In this work, JSF technology is used for development of the web application which gives different kind of information related to municipalities of Belgrade (information about restaurants, theatres etc). Besides this, application offers user some services, like ordering food, ticket booking etc. This application demonstrates some of the main JSF possibilities.

1. Uvod

Glavna tema ovog master rada je, kao što i sam naslov kaže, primena Java Server Faces tehnologije u izradi web aplikacija. JSF tehnologija je postala veoma primenjena u svetu web programiranja, što je i bio jedan od razloga za izbor teme mog master rada.
Ukratko, Java Server Faces predstavlja Java aplikativno okruženje koje za osnovni cilj ima pojednostavljeni i brži razvoj web aplikacija sa bogatim korisničkim interfejsom. U daljem tekstu će skraćenica UI biti korišćena upravo kada je potrebno spomenuti korisnički interfejs (UI-User Interface). Što se tiče drugih stručnih termina, uglavnom će biti prevođeni na srpski jezik, osim u slučajevima kada se prevođenjem gubi smisao ili je neka engleska reč već uveliko ustaljen pojam (kao što je recimo interfejs).
U okviru glavnog dela rada će takođe biti reči i o konceptima bitnim kako za razumevanje samog JSF-a, tako i za izradu web aplikacija uopšte.
Osnovna ideja je da se nakon čitanja ovog rada stekne jasna slika o svim ključnim konceptima koje JSF tehnologija donosi.

2. MVC koncept

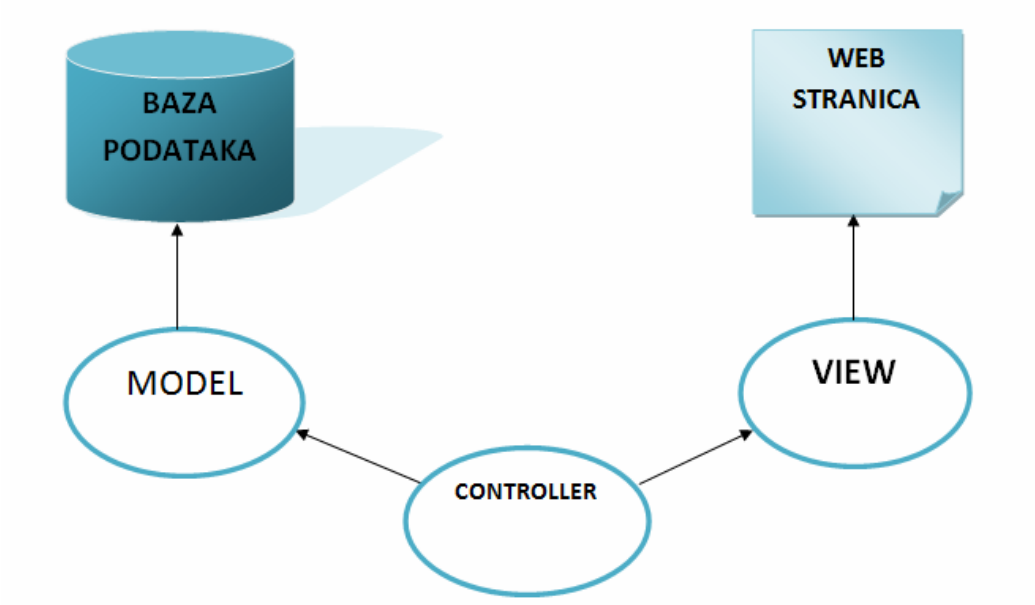
Kada govorimo o Java web programiranju, pojam koji je veoma važno razumeti je MVC obrazac. MVC predstavlja arhitekturalni stil koji se koristi kod interaktivnih sistema i njegova osnovna odlika je da razdvaja domen rada u tri jasno razdvojena sloja. To su Model, View i Controller, od kojih i potiče skraćenica MVC. Možemo reći da MVC obrazac razdvaja modeliranje domena, prezentaciju i akcije bazirane na inputu (ulaznim vrednostima) korisnika. Arhitektura aplikacije je definitivo veoma bitna stavka u dizajnu aplikacije. Ona određuje kako elementi aplikacije međusobno deluju i kakvu funkcionalnost svaki element pruža. Pravilno osmišljena arhitektura aplikacije omogućava da kasnije održavanje aplikacije, kao i proširenje po broju korisnika i nadgradnja u smislu funkcionalnosti ne bude velika prepreka.

Model predstavlja poslovnu logiku aplikacije. Obuhvata klase za čuvanje podataka i manipulaciju istim tj. upravlja podacima i ponašanjem aplikacije (predstavlja podatke koji se u aplikaciji koriste, putem objakata). Model odgovara na zahteve za informacijama o stanju domena (od View-a), kao i na instrukcije za promenu stanja (najčešće od Controller-a). Omogućava olakšano testiranje aplikacije, poboljšava kvalitet itd. JavaBeans je tehnologija koja predstavlja implementaciju ovog sloja.

View služi za predstavljanje informacija krajnjem korisniku. Sastoji se od elemenata korisničkog interfejsa, recimo: text box-ova, input polja, labela itd. Tehnologija koja implementira ovaj sloj je Java Server Pages tehnologija.

Controller predstavlja sponu između Model-a i View-a i sadrži svu aplikativnu logiku. Kontroliše interakciju između druga dva sloja, korišćenjem akcija kao što su klikovi mišem ili tipkanje tastature. Controller omogućava promene modela u zavisnosti od inputa korisnika. Vrši transformacije potrebne za prosleđivanje podataka od Model-a do View-a.
Za implementaciju ovog sloja neophodna je Servlet klasa.

Konceptualno, MVC arhitektura je triangularna.



Uvođenjem Contoller-a postiže se to da promene korisničkog interfejsa ne utiču na podatke, kao i to da promene u organizaciji podataka ne utiču na korisnički interfejs.

Važno je napomenuti da i View i Controller zavise od Model-a, a s druge strane, Model ne zavisi ni od View-a ni od Controller-a. Ova činjenica je jedna od glavnih koristi koje MVC razdvajanje donosi. Ta razdvojenost omogućava razvoj i testiranje modela nezavisno od vizuelne reprezentacije. U web aplikacijama, odvojenost View-a (pretraživača) i Controller-a (komponente na serveru koja rukovodi HTTP zahtevom) je veoma dobro definisana.
Arhitektura JSF-a integriše MVC dizajnerski obrazac, što aplikacije razvijene korišćenjem JSF-*a* čini dobro dizajniranim i lakim za održavanje.

Aplikacije koje su izrađene tako da podržavaju MVC obrazac možemo zvati *multi-tier* odnosno višeslojnim aplikacijama.

3. Java EE platforma

U svetu informacionih tehnologija, složene aplikacije moraju biti projektovane i proizvedene sa manje novčanih sredstava, manje resursa, a većom brzinom. Java EE platforma je upravo ta koja sadrži skup koordiniranih tehnologija koje značajno smanjuju cenu kao i kompleksnost razvoja, primene i upravljanja *multi-tier* aplikacijama na strani servera.

Java EE obezbeđuje izvršno okruženje i skup API-ja za razvoj i pokretanje “enterprajz softvera”, uključujući network i web servise, kao i druge portabilne, robustne, skalabilne, sigurne i pouzdane serverske aplikacije. Robustnost predstavlja sposobnost sistema da upravlja greškama tokom izvršavanja ili sposobnost algoritma da nastavi sa radom uprkos neispravnom ulazu i slično. Skalabilnost se odnosi na mogućnost povećanja resursa. Skalabilnim aplikacijama mogu se dodati resursi da bi se upravljalo dodatnim zahtevima bez modifikovanja same apikacije.
S obzirom da JSF predstavlja sastavni deo Java EE (Java Enterprise Edition) platforme, veoma se široko koristi prilikom razvoja enterprajz softvera.

> ”Enterprajz softver” je softver koji se koristi u organizacijama kao što su neke velike poslovne ili recimo vladine organizacije. Servisi koje je često potrebno omogućiti ovim softverom su tipični poslovno-orjentisani servisi, kao na primer *online* kupovina, plaćanje preko interneta, interaktivni katalozi itd. Za razliku od *single-user* aplikacija koje se izvršavaju na korisnikovom računaru, enterprajz softverska aplikacija se izvršava na serveru i istovremeno omogućava usluge velikom broju korisnika. Neke od najvećih organizacija za razvoj ove vrste softvera su: IBM, Oracle Corporation, Microsoft, JBoss

(Red Hat), Adobe Systems, SAP, HP Software division i monoge druge. (Više informacija o ovome može se naći u [11].)

Java EE je definisana preko svoje specifikacije. Obuhvata brojne API specifikacije i definiše kako se njima barata. Neke od tih su: JDBC (Java Database Connectivity), RMI (Remote Method Invocation), JMS (Java Message Service), web services, XML (Extensible Markup Language). Pored ovoga, Java EE se ističe nekim specifikacijama jedinstvenim za nju samu, kao što su EJB (Enterprise Java Beans), Connectors, Servlets, Portlets, JSF itd.
Trenutna, odnosno najnovija verzija ove platofme je Java EE 6. Među mnogobrojnim paketima koji su njom obuhvaćeni, tu su : javax.faces, javax.servlet, javax.ejb, javax.validation, javax.persistance, javax.transaction, javax.mail, od kojih će neki biti veoma bitni za ovaj rad, ali ce oni biti razmotreni kasnije**.** Ono što je sada važno jeste konkretnija priča o JSF-u.

4. Nastanak i razvoj JSF tehnologije

Imajući u vidu već postojeće i široko korišćene tehnologije, sasvim je prirodno zapitati se šta je iniciralo JSF projekat i zašto je JSF tehnologija postala tako popularna. Za to svakako postoje razlozi i njih su sigurno svesni svi oni koji su ikada imali prilike da rade na razvoju kompleksne web aplikacije korišćenjem tehnologija kao što su JSP ili recimo Servlets. Evo nekoliko problema koji su se ovom prilikom javljali:

- Prethodne tehnologije su od programera zahtevale mnogo zamornog i ponavljajućeg kodiranja.
- Korišćenjem prethodnih tehnologija programeri su bili primorani na direktan rad sa HTTP zahtevima (*request*) i odgovorima (*respons*).
- Nedostupnost razvojnih okruženja (IDE - Integrated Development Environment) je još jedna velika mana koja utiče na produktivnost programera , pa samim tim i na porast cene projekta.

Zahvaljujući JSF tehnologiji ovi problem su otklonjeni.
Java Server Faces tehnologija je razvijena od strane JCP-a (JCP- Java Community Process), tj. od strane velikog broja eksperata u domenu web aplikacija i to iz različitih kompanija kao što su: Jakarta Struts, Oracle, Sun, IBM, ATG itd. Njihovim zajedničkim naporima stvorena je tehnologija koja predstavlja najbolju moguću kombinaciju postojećih okruženja (*framework*-a),

s obzirom da je razvijena uzimajući najbolje, a izostavljajući mane tehnologija od kojih je nastala.
JSF je sastavni deo Java EE platforme i predstavljena je specifikacijom, dok različiti proizvođači softvera razvijaju sopstvene implementacije, pri čemu je i onih koje su besplatne, komercijalne pa tako i svima dostupne. U današnje vreme mnogi proizvođači softvera razvijaju IDE za razvoj JSF zasnovanih aplikacija, što je zaista dobra vest za one koji se odluče da savladaju ovo okruženje.

> *Framework* (okruženje) se odnosi na biblioteku ili skup biblioteka, definisan API, pravila i metode koje se koriste da se ubrza i standardizuje izrada softverskih proizvoda, tj. aplikacija. Web framework se koristi za izradu web aplikacija. Ondosi se na oboje: biblioteke i način na koji funkcioniše. Kad se kaže JSF framework, misli se na način kako JSF funkcioniše, pravila pisanja JSF aplikacija, dokumentaciju, kao i biblioteke koje dolaze kao deo JSF-a.

Pre pojave JSF-a, standardno okruženje za razvoj web aplikacija bio je Struts. U to vreme, Struts je obezbeđivao sva neophodna svojstva, međutim, kako su konkurenti razvijali dodatne mogućnosti, koje su Struts-u nedostajale, tako je postalo neophodno da se formira novi Java standardni framework sa moćnim komponentnim modelom. Tako je nastala JSF tehnologija. Osnovni cilj prilikom razvoja JSF-a bio je kreiranje skupa API-ja koji bi omogućio rukovođenje stanjima UI komponenti, rukovođenje događajima i validaciju.

Evo i kratkog pregleda verzija JSF-a :

- JSF 1.0 (11.03.2004.) je inicijalna verzija JSF specifikacije. Nije bila deo nijedne od verzija Java EE platforme.
- JSF 1.1 (27.05.2004.) je verzija u kojoj su uklonjeni “bagovi” uočeni u prethodnoj verziji, bez ikakvih promena u specifikaciji.
- JSF 1.2 (11.05.2006.) je verzija koja je donela mnoga poboljšanja i uklapa se u Java EE 5 platformu.
- JSF 2.0 (28.06.2009.) je verzija koja je omogućila jednostavno korišćenje i poboljšanu funkcionalnost. Uklapa se u Java EE 6 platformu.
- JSF 2.1 (22.10.2010.) je poslednja verzija JSF-a. U odnosu na prethodnu verziju, razlikuje se veoma malo u pogledu specifikacije.

5. JSF impementacija MVC obrazca

Kao što je već spomenuto ranije, JSF framework integriše MVC dizajnerski obrazac. Svaki od slojeva (Model, View, Controller) podržan je jednom tehnologijom (JavaBeans, JSP, Servlet).

5.1. JavaBeans

JavaBeans je tehnologija koja omogućava korišćenje višekratno upotrebljivih softverskih komponenti (*reusable software components*). Ove komponente se koriste da enkapsuliraju više objekata u jedan (*bean*). Bean komponente su predstavljene Java klasama koje zadovoljavaju određenu konvenciju. JavaBean objekti su Java objekti koji su serijalizabilni, imaju prazan konstruktor (konstruktor bez argumenata) i omogućavaju pristup atributima korišćenjem *getter* i *setter* metoda. Ova tehnologija je veoma popularna i pogodna jer sprovodi u delo ideju „Piši jednom, koristi bilo gde“.

Tipična JSF aplikacija sadrži jedan ili više *backing bean*-ova (*managed bean*-ova), a to su JavaBeans komponente pridružene UI komponentama stranice. *Backing bean* definiše svojstva UI komponenata od kojih je svako povezano ili sa vrednošću komponente ili sa njenom instancom. Takođe, *backing bean*-om su definisane metode kojima se sprovode funkcionalnosti komponenata, uključujući validaciju, rukovođenje događajima i navigaciju.
Za povezivanje vrednosti ili instanci UI komponenata sa *backing bean*-ovima ili za referenciranje *backing bean* metoda iz tagova UI komponenti koristi se JSF EL (Java Server Faces Expression Language).

5.1.1. Backing (Managed) bean opsezi važenja

Prilikom konfigurisanja *backing bean* atributa, po potrebi je moguće odrediti opseg važenja nekog objekta (promenljive). JSF aplikacija podržava *application*, *session* i *request* opseg. Kao što i sama imena govore, važi sledeće:

- *Backing bean* objekat, koji treba da bude globalan tj. dostupan tokom celog života aplikacije, treba da bude konfigurisan kao *application scope*.
- *Backing bean* objekat, koji treba da bude dostupan tokom sesije korisnika, treba da bude konfigurisan kao *session scope*.
- *Backing bean* objekat je dostupan samo tokom jednog zahteva, ako je konfigurisan kao *request scope*. *Request scope* je podrazumevani opseg važenja.

5.1.2. Java Server Faces Expression Language

Osnovna svrha JSF EL-a je da omogući pozivanje i ažuriranje bean atributa ili procesiranje nekih jednostavnijih iskaza, bez pisanja Java koda.
U JSF-u, EL se najviše koristi za povezivanje UI komponenata sa backing bean atributima ili objektima modela. Izrazi JSF-a se procesiraju u vreme izvršavanja (obično u vreme prikazivanja pogleda), a ne u vreme kompilacije.
Osim toga što omogućava referenciranje jednostavnih atributa, obuhvata sintaksu za pristupanje elementima nizova, kolekcija itd. Pored ovoga, obuhvata i skup implicitnih objekata (varijabli). Implicitne varijable su posebni identifikatori EL-a koji omogućavaju jednostavan pristup objektima koji su često potrebni programerima (parametrima zahteva, HTTP zaglavljima, cookies-ima itd).
JSF EL izrazi mogu da vrše, kako dodeljivanje i ažuriranje vrednosti atributa, tako i preuzimanje tih vrednosti. Zato za njih kažemo da su dvosmerni (*two-way*).
Sintaksa EL-a je takva da ispred svakog izraza mora da stoji znak '#'.

Postoji zaista mnogo detalja vezanih za sintaksu JSF EL-a, ali o njima ovde neće biti reči. Ono što je važno napomenuti jeste da postoje dve važne vrste izraza. To su:

- Izrazi povezivanja vrednosti (*Value-binding expressions*)
  Ovi izrazi se mogu koristiti za povezivanje vrednosti UI komponente sa atributom backing bean-a, za povezivanje instance UI komponente sa atributom *backing bean*-a ili za inicijalizaciju atributa UI komponente.
- Izrazi pridruživanja metoda (*Method-binding expressions*)
  Ovi izrazi se koriste za povezivanje obrađivača događaja (*event handler*-a) ili metoda validacije sa komponentama.

Svi atributi koji se referenciraju korišćenjem JSF Expression Language-a moraju biti JavaBean objekti.

5.2. JSP

Java Server Pages predstavlja tehnologiju koja se koristi za implementaciju prezentacionog sloja web aplikacije izgrađene JSF tehnologijom. Praktično predstavlja kombinaciju HTML-a i Java koda.

Iako JSF tehnologija nije vezana za određenu tehnologiju prikaza, JSP je tehnologija koja je u današnje vreme veoma primenjena jer omogućava jednostavno projektovanje web stranica sa odvojenim dinamičkim sadržajem od statički, šablonski prikazanih podataka.

5.3. Java Servlet

Servlet je Java klasa koja se koristi da proširi mogućnosti servera na kojima se izvršavaju aplikacije koje funkcionišu po principu zahtev-odgovor. Iako servleti mogu da odgovore na bilo koji tip zahteva, najčešće se koriste kao podrška aplikacijama koje se izvršavaju na web serveru. Za takve aplikacije, Java Servlet tehnologija definiše HTTP-specifične servlet klase. Javax.servlet i javax.servlet.http su paketi koji obezbeđuju klase i interfejse za pisanje servleta.
Važna stvar kod servleta jeste da moraju da rukuju konkurentnim zahtevima.

Dakle, servleti su Java klase koje se pokreću na serveru u okviru Java virtualne mašine, prihvataju zahteve klijenata i generišu odgovor. Servleti su pravljeni tako da budu serverski nezavisni. S obzirom da se pokreću u okviru JVM-a, servleti su platformski nezavisni, a s obzirom da se pokreću na serveru, nije ih potrebno prilagođavati ni pretraživaču. Nisu dizajnirani za određeni protokol, ali se ipak najčešće koriste sa HTTP-om.

5.3.1. Hypertext Transfer Protocol

HTTP je mrežni protokol koji predstavlja najčešći metod prenosa informacija na webu. Osnovna namena ovog protokola jeste isporuka web stranica (HTML dokumenata).
To je protokol za komunikaciju između servera i klijenta (koji je najčešče web pretraživač), koji funkcioniše po principu zahtev-odgovor. Server konstantno osluškuje zahteve na određenom komunikacionom portu, čekajući da se klijent poveže i pošalje svoj zahtev. Zahtev klijenta se obrađuje na serveru i u zavisnosti od ispravnosti zahteva, klijentu se šalje odgovarajući odgovor.
HTTP je *stateless* protokol. To znači da ne zahteva od servera da zadržava stanja i informacije o korisniku tokom više zahteva. Kao odgovor na ovaj problem, Servlet API obezbeđuje korišćenje sesije.
(Više informacija o Http protokolu može se naći u [11] i [12].)

6. Struktura JSF aplikacije

JSF aplikacije su standardne web aplikacije za čiji je razvoj neophodno imati instalirano nekoliko stvari. To su:

1. Implementacija JSF-a (Na primer: Oracle Mojarra, Apache MyFaces, RichFaces)
2. JSP Standard Tag Library (JSTL)
3. Java razvojno okruženje (Na primer: Eclipse)
4. Web kontejner (Na primer: Tomcat)

Implementacija JSF-a predstavljena je u formi .jar fajlova. U zavisnosti od implementacije, razlikuju se biblioteke koje su sa tom implementacijom isporučene. Ipak, postoje neke biblioteke klasa koje moraju biti sadržane u okviru svake implementacije. To su:

1. Jsf-api.jar (JSF API klase desinisane JSF specifikacijom)
2. Jsf-impl.jar (JSF klase specifične za datu implementaciju)

JSTL je kolekcija korisnih JSP tagova koji enkapsuliraju osnovne funkcionalnosti zajedničke mnogim JSP aplikacijama (tagovi za iteraciju i uslove, za manipulisanje XML dokumentima, za SQL itd). Važno je napomenuti da JSF biblioteke zavise od JSTL biblioteka. Zato je neophodno i njih uključiti u aplikaciju, iako ih nije neophodno u okviru aplikacije koristiti. JSTL biblioteke je kao i implementaciju JSF-a potrebno preuzeti sa interneta i smestiti ih u lib direktorijum aplikacije.

Ukoliko se koristi izvršno okruženje i/ili web kontejner, koji obuhvataju implementaciju JSF-a, o svemu ovome nije potrebno voditi računa. Ako to nije slučaj, pre nego što se počne sa radom na aplikaciji, moraju se obaviti spomenuta podešavanja.

Što se tiče web kontejnera, to je komponenta web servera koja omogućava izvršavanje JSP strana i servleta, pa samim tim predstavlja komponentu u okviru koje se vrši obrada zahteva i slanje odgovora kljentu (web pretraživaču).

Tipična JSF web aplikacija sadrži:

1. JSP stranice (u koje se ugrađuju JSF komponente).

2. JavaBeans komponente, koje predstavljaju osobine i funkcije komponenti korisničkog interfejsa na web strain.
3. *Custom* komponente, validatore, konvertere, osluškivače događaja.
4. Web.xml datoteku (U okviru ove datoteke navodi se da je FacesServlet odgovoran za rukovođenje JSF aplikacijom. FacesServlet klasa (javax.faces.webapp.FacesServlet) prima sve zahteve za JSF aplikaciju i inicijalizuje JSF komponente pre nego što se JSP stranica prikaže).
5. Faces.config datoteku (Ovo je aplikaciona konfiguraciona datoteka u kojoj su definisana pravila navigacije, konfigurisani *bean*-ovi i drugi *custom* objekti, kao što su recimo *custom* komponente, validatori ili konverteri).

Kao i sve Java web aplikacije, i JSF aplikacija mora da sadrži web.xml datoteku. U okviru ove datoteke vrši se podešavanje JSF Servleta. Konkretno, JSF aplikacije zahtevaju Faces Servlet koji ima ulogu glavnog kontrolera cele aplikacije. Da bi se podesio Faces Servlet, potrebno je uraditi dve stvari:

1. Definisati servlet
2. Mapirati servlet URL šablonom

Definisanje servleta podrazumeva njegovo imenovanje i navođenje klase čija će se instanca kreirati prilikom kreiranja servleta.
Mapiranje servleta URL šablonom može biti:

- Prefiksno `<url-pattern>/faces/*</url-pattern>`
- Sufiksno `<url-pattern>*.faces</url-pattern>`

U prvom slučaju, svaki zahtev za stranicom čija putanja sadrži /faces/ biće obrađen od strane JSF-a, dok će u drugom slučaju to da se dogodi za sve zahteve za stranicama sa ekstenzijom .faces (JSF uvek traži ime datoteke sa podrazumevanim sufiksom, tako da će bez problema obraditi zahtev za stranicom čija je ekstenzija .faces).

Osim podešavanja servleta, postoje još neki parametri koji se podešavaju u okviru web.xml datoteke. Različite implementacije JSF-a omogućavaju i različite parametre, ali samo neki od njih su propisani specifikacijom kao neophodni. To su:

- Javax.faces.CONFIG_FILES
- Javax.faces.DEFAULT_SUFFIX
- Javax.faces.LIFECYCLE_ID
- Javax.faces.STATE_SAVING_METHOD

Parametrom CONFIG_FILES definiše se lista konfiguracionih datoteka koje je potrebno uključiti u aplikaciju. Ovo je poželjno koristiti kada na razvoju aplikacije radi više programera ili timova od kojih svaki ima sopstveni konfiguracioni fajl.

DEFAULT_SUFFIX parametrom se definiše podrazumevana ekstenzija stranica aplikacije. Podrazumevana ekstenzija je inače .jsp, tako da ukoliko se kao tehnologija prikaza koristi JSP tehnologija, ovaj parametar nije potrebno koristiti.
LIFECYCLE_ID parametrom se specificira identifikator životnog ciklusa aplikacije, koji određuje na koji način JSF sprovodi ceo taj proces. U većini slučajeva nije ga potrebno koristiti.
JSF ima mogućnost da na stranici čuva stanja komponenti, tako da ako se stranica prikaže ponovo, vrednosti koje je korisnik uneo ostaju zapamćene. Stanje komponenti se može čuvati na klijentu ili na serveru i upravo je STATE_SAVING_METHOD parametar, onaj kojim se to kontroliše. Njegove moguće vrednosti su, očekivano, *client* ili *server*. Čuvanje stanja komponente na kljientu rezultuje smanjenim opterećenjem servera. Podrazumevana vrednost ovog parametra je *server*. Ukoliko je ipak postavljen na vrednost *client*, stanje komponente se čuva u velikom skrivenom *input* polju pretraživača. Čuvanje stanja na strani klijenta je pogodno u situacijama pada servera jer stanje komponenti ostaje sačuvano.

Pored web.xml fajla, JSF ima sopstveni sistem za konfiguraciju koji obezbeđuje mnoštvo dodatnih mogućnosti. Konfiguracija specifična za JSF obavlja se u okviru aplikativnog konfiguracionog fajla, tj. faces.config.xml fajla. Tu se vrši podešavanje pravila navigacije, podešavanje *backing bean*-ova, registrovanje komponenata, renderera, validatora, konvertera itd.
JSF podržava višestruke konfiguracione fajlove. Podrazumevano, JSF pre svega traži fajl pod nazivom faces-config.xml u okviru WEB-INF direktorijuma. Dodatni konfiguracioni fajlovi se u aplikaciju uključuju korišćenjem već opisanog javax.faces.CONFIG_FILES parametra. Konfiguracioni fajlovi JSF-a su XML fajlovi koji počinju `<faces-config>` elementom. Svi ostali elementi su opcioni i mogu se svrstati u 3 kategorije.

1. XML elementi za svakodnevnu aplikativnu konfiguraciju (`<application>`, `<managed-bean>`, `<referenced-bean>`, `<navigation-rule>`)
2. XML elementi za registrovanje UI komponenti (`<component>`, `<render-kit>`, `<validator>`, `<converter>`)
3. XML elementi za napredniju konfiguraciju. (`<phase-listener>`, `<factory>`)

Prilikom razvoja aplikacije najčešće je dovoljno brinuti o prvoj kategoriji elemenata.
JSF aplikacija ne može da se pokrene sve dok u nekom konfiguracionom fajlu postoji bar jedna greška. U zavisnosti od implementacije JSF-a, nekad je prilikom promena u konfiguraciji neophodno ponovo pokrenuti aplikaciju kako bi promene imale efekta.

7. Životni ciklus JSF strane

Da bi se kreirale što kvalitetnije i što efikasnije web aplikacije, važno je biti upoznat sa tokom dešavanja prilikom pokretanja web stranice. Razumevanje načina na koji JSF procesira svaki korisnikov zahtev omogućava bolju izradu web aplikacija jer nas upoznaje sa tim koja će se operacija odigrati i kada. Zato ćemo sada detaljnije razmotriti životni ciklus JSF strane.

Svaka JSF stranica predstavljena je stablom UI komponenti, koje nazivamo pogled (*view*). Tokom svog životnog ciklusa JSF stranica prolazi kroz šest faza. To su:

1. Faza formiranja stabla (*Restore view phase*)
2. Faza dodele vrednosti (*Apply request values*)
3. Faza validacije (*Process validation phase*)
4. Faza ažuriranja vrednosti (*Update model values phase*)
5. Faza pokretanja aplikacije (*Invoke application phase*)
6. Faza prikaza odgovora (*Render response phase*)

Životni ciklus može da obrađuje dve vrste zahteva:

1. Inicijalne zahteve (*Initial requests*)
2. Ponovne zahteve (*Postbacks*)

Inicijalni zahtev se formira kada korisnik zatraži stranicu po prvi put. Najčešće, prvi zahtev za JSF stranom stiže kao posledica klika na *hyperlink* HTML stranice koja je povezana sa JSF stranom. Kao rezultat klika, formira se novi pogled, koji se čuva u *FacesContext* objektu,a koji sadrži sve kontekstualne informacije povezane sa obradom dolazećeg zahteva i kreiranjem odgovora. Aplikacija zatim zahteva reference objekata potrebne pogledu i poziva *FacesContext.renderResponse()* metod koji trenutno pokreće prikazivanje pogleda prelaskom na poslednju fazu ciklusa. Dakle, kod inicijalnih zahteva izvršavaju se samo prva i poslednja faza životnog ciklusa, jer ne postoje ulazni podaci ili akcije koje treba procesirati.

S druge strane, kada je u pitanju ponovni zahtev, JSF implementacija obrađuje zahtev i prolazi kroz sve faze životnog ciklusa kako bi obavila sve neophodne validacije, konverzije, ažuriranja modela i kako bi generisala odgovor.

Nakon većine faza, JSF implementacija prenosi događaje osluškivačima događaja (*event listener*-ima). Osluškivači događaja izvršavaju aplikativnu logiku i manipulišu komponentama. Oni mogu da utiču na JSF životni ciklus na jedan od sledećih načina:

1. Mogu da dozvole da se životni ciklus izvršava normalno.

2. Mogu da pozovu *FacesContext.renderResponse()* metodu da bi se preskočio ostatak životnog ciklusa sve do faze prikazivanja odgovora.
3. Mogu da pozovu *FacesContext.responseComplete()* metodu da bi se preskočio ostatak životnog ciklusa, ceo, ujedno sa poslednjom fazom.
   (Poslednja faza ciklusa se preskače u slučaju da aplikacija treba da se preusmeri na neki drugi resurs web aplikacije, kao što je Web servis, ili da generiše odgovor koji ne sadrži JSF komponente.)

Razmotrimo sada svaku od faza pojedinačno.

7.1 Faza formiranja stabla (pogleda)

Kao što je već spomenuto, pogledom su predstavljene sve komponente od kojih je sačinjena jedna stranica. Drugim rečima, svaki pogled je sačinjen od drveta komponenti. Pogled počinje *UIViewRoot* komponentom, koja je kontejner za sve ostale komponente na stranici. Svaki pogled mora imati svoj jedinstveni identifikator. Ovaj identifikator odgovara delu putanje koji dolazi nakon imena servleta (Recimo, ako je URI (*Uniform Resource Identifier*) */faces/login.js*p, onda je traženi identifikator pogleda */login.jsp*). Pogled može biti smešten na klijentu (obično u skrivenom polju pretraživača) ili na serveru (najčešće, u sesiji korisnika).

Faza kreiranja pogleda počinje u trenutku kada korisnik napravi zahtev za JSF stranicom (klikom na link, dugme i sl.). Zahtev dolazi kroz *Faces Servlet* kontroler. Kontroler ispituje zahtev i izvlači identifikator pogleda koji je određen imenom te strane. Taj identifikator pogleda se koristi za pronalaženje komponenti za tekući pogled. Moguće situacije su:

1. Inicijalni zahtev
2. Ponovni zahtev (*Postback*)

U slučaju inicijalnog zahteva (kada se stranica pokreće prvi put), JSF implementacija kreira prazan pogled i životni ciklus prelazi direktno na fazu prikaza odgovora, s obzirom da zahtev ne sadrži nikakve ulazne podatke. Prazan pogled se popunjava kada se stranica obradi *postback*-om.
U slučaju ponovnog zahteva (kada korisnik šalje zahtev za stranicom sa koje i šalje taj zahtev), pogled koji odgovara toj stranici već postoji, pa ga u tom slučaju samo treba rekonstruisati. JSF implementacija rekonstruiše pogled koristeći informacije sačuvane na klinentu ili na serveru, u sesiji korisnika.

Dakle, u prvoj fazi ciklusa se, zavisno od vrste zahteva, ili kreira prazan pogled (inicijalni zahtev) ili se već postojeći pogled rekonstruiše (ponovni zahtev). Kreirani pogled se čuva u *FacesContext* objektu. *Faces Context* objekat čuva pogled u okviru svog *viewRoot* atributa. *ViewRoot* sadrži sve JSF komponente za tekući identifikator pogleda. U *FacesContext* objektu sadržane su sve informacije potrebne za procesiranje zahteva.

7.2. Faza dodele vrednosti

Tokom ove faze, JSF implemantacija prolazi kroz objekte komponenti u okviru stabla komponenti i osnovni cilj je da svaka komponenta preuzme svoju tekuću vrednost koja je prosleđena od strane korisnika u tekućem zahtevu (Svaka komponenta preuzima svoju novu vrednost i čuva je lokalno). Vrednosti komponenata se obično preuzimaju iz parametara zahteva. Ovaj proces se naziva dekodiranje. Svaka komponenta traži parametar čije je ime isto kao i njen klijentski identifikator (klijentski identifikator je identifikator koji se šalje pretraživaču) i kada je on pronađen, vrednost komponente se postavlja na vrednost tog parametra. JSF implementacija postavlja nove vrednosti komponenata pozivanjem *processDecodes()* metode za *UIViewRoot* komponentu. Ovim pozivom se ta metoda rekurzivno primenjuje na sve *child input* komponente (ulazne dete-komponente), a na kraju svaka komponenta poziva *decode()* metodu. Ako je konverzija vrednosti neuspešna, generiše se poruka o grešci vezanoj za komponentu i dodaje se u *FacesContext*. Ova poruka će biti prikazana zajedno sa greškama validacije koje su nastale u trećoj fazi ciklusa.

Decode() metoda ili osluškivači događaja, mogu da skrate životni ciklus i da preusmere procesiranje na poslednju fazu, pozivom *renderResponse()* metode ili da u potpunosti završe procesiranje. Ako aplikacija ima potrebu da u ovoj fazi generiše odgovor koji ne sadrži JSF komponente, tada poziva metodu *FacesContext.responseComplete()*.

Svaka *input* komponenta (UI komponenta koja prihvata unos korisnika, ulazna komponenta) ima *immediate* (neposredno) atribut. Ako je vrednost ovog atributa postavljena na *true*, onda se validacija, umesto u trećoj fazi, obavlja u ovoj fazi. Osim *input* kontrola (ulaznih kontrola), atribut *immediate* imaju i kontrole koje proizvode određene akcije, kao što su recimo *button* ili *hyperlink*. Ako je za njih atribut *immediate* postavljeno na *true*, događaji koje oni iniciraju se stavljaju u red događaja, a JSF implementacija ih procesira nakon dodeljivanja vrednosti komponenti, ali pre prelaska na sledeću fazu ciklusa. Ako je atribut *immediate* postavljen na *false*, vrši se konverzija, ali se ne vrši validacija.

7.3. Faza validacije

Tokom ove faze životnog ciklusa vrši se validacija lokalne vrednosti svake komponente u stablu komponenti. Svaka UI komponenta može imati pridružen validator (posebna JSF komponenta) i/ili može direktno da implementira *validate()* metod.
JSF implementacija vrši validaciju kontrola pozivom *processValidators()* metode za *UIViewRoot* komponentu u stablu komponenti. Ovim pozivom se ta metoda rekurzivno primenjuje na sve

ulazne dete-komponente, a zatim se metoda *validate()* poziva za svaki pridruženi validator, a na kraju i za samu komponentu. Ako je validacija uspešna, životni ciklus se nastavlja normalno prelaskom na narednu fazu. Ako je vrednost koju je korisnik prosledio neispravna, generiše se poruka o grešci i smešta u *FacesContext*. JSF implementacija tada prelazi na fazu renderovanja odgovora, prikazujući ponovo traženu stranicu, zajedno sa graškama validacije, a ujedno i konverzije (ako ih je bilo u prethodnoj fazi), i pri tome omogućavajući korisniku ponovni unos.
Kao i tokom druge faze, i tokom ove, događaji mogu biti stavljeni u red događaja za procesiranje. Nakon što se izvrše sve validacije, JSF implementacija procesira događaje pre prelaska na narednu fazu.

Validate() metoda, sami validatori pridruženi komponenti ili osluškivači događaja mogu da skrate životni ciklus i da preusmere procesiranje na poslednju fazu ciklusa, pozivom *renderResponse()* metode, ili da u potpunosti završe procesiranje. Ako aplikacija ima potrebu da u ovoj fazi generiše odgovor koji ne sadrži JSF komponente, tada poziva metodu *FacesContext.responseComplete().*

Dakle, na kraju ove faze, sve komponente su validirane, a slično kao i prethodna faza i ova se može nastaviti normalno, može se preusmeriti na fazu prikaza odgovora ili može potpuno prekinuti životni ciklus.

7.4. Faza ažuriranja vrednosti modela

Nakon treće faze, kada je sigurno da su lokalne vrednosti komponenti ažurirane, validne i ispravnog tipa, bezbedno je izvršiti promene modela podataka. Tokom ove faze, lokalne vrednosti se koriste da bi se ažurirali objekti na serverskoj strani koji odgovaraju UI komponentama, tj. ažuriraju se *backing bean* komponente. *Backing bean* komponente su sa UI komponentama povezane korišćenjem *JSF Expression Language*-a. JSF implementacija vrši ovo ažuriranje modela pozivanjem *processUpdates()* metode za *UIViewRoot* komponentu. Ovim pozivom se ta metoda rekurzivno primenjuje na sve ulazne dete-komponente i na kraju se poziva *updateModel()* metoda svake UI komponente.
Ako se dogodi da je ažuriranje neke UI komponente neuspelo, generiše se poruka o grešci i smešta se u *FacesContext*, korišćenjem *addMessage()* metode. Svako neuspelo ažuriranje rezultuje prelaskom na fazu renderovanja odgovora i informisanjem korisnika o greškama.

I tokom ove faze, događaji mogu biti stavljeni u red događaja za procesiranje. Nakon što se obave sva potrebna ažuriranja, a pre prelaska na narednu fazu, JSF implementacija procesira ove događaje.

Bilo koja komponenta može da skrati životni ciklus i da preusmeri procesiranje na poslednju fazu, pozivom *renderResponse()* metode, ili da u potpunosti završi procesiranje. Ako aplikacija

ima potrebu da u ovoj fazi generiše odgovor koji ne sadrži JSF komponente, tada poziva metodu *FacesContext.responseComplete().*

7.5. Faza pokretanja aplikacije

Kako su u ovom momentu toka obrade, vrednosti UI komponenata konvertovane, validirane i primenjene na odgovarajuće *bean* objekte, sada se te vrednosti mogu upotrebiti za izvršavanje poslovne logike aplikacije. U okviru ove faze izvršava se *action()* metoda (*UICommand* komponente, kao što su dugme ili link) koji dovodi do slanja forme ili povezivanja sa drugom stranom. Drugim rečima, u ovoj fazi se vrši upravljanje događajima. JSF implementacija pokreće ovaj proces pozivom *processAplication()* metode za *UIViewRoot* komponentu.

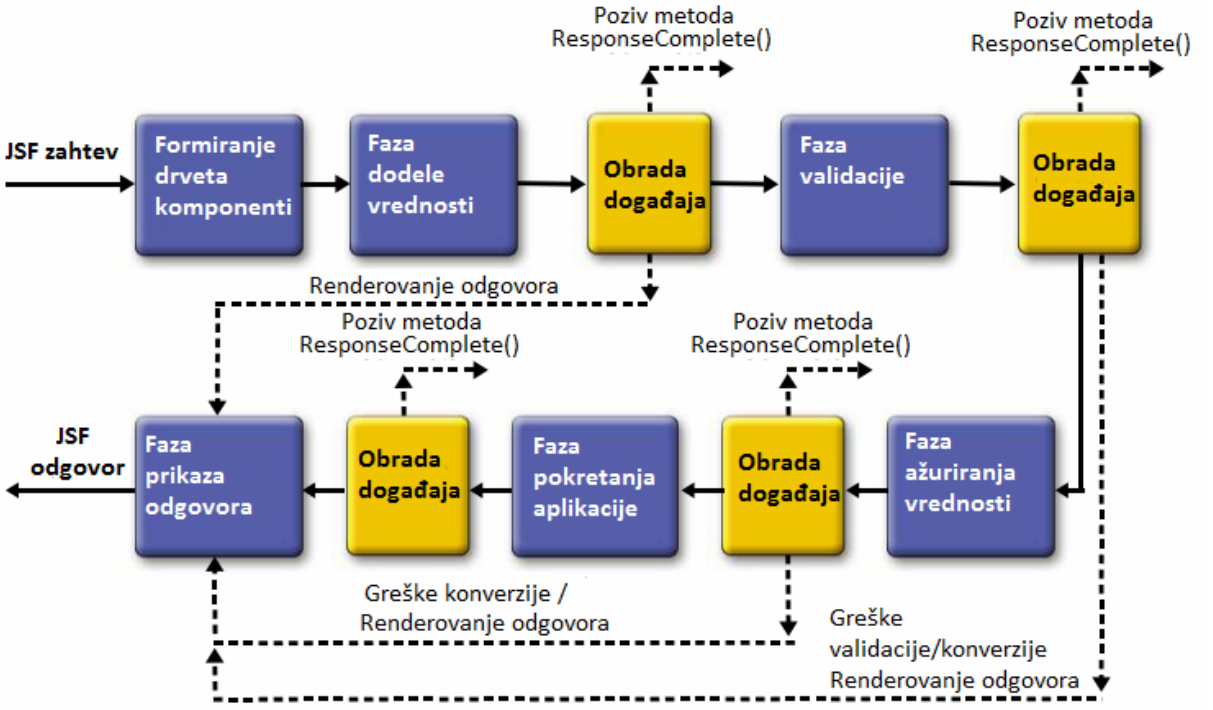
Ako aplikacija ima potrebu da u ovoj fazi generiše odgovor koji ne sadrži JSF komponente, tada poziva metod *FacesContext.responseComplete().* U suprotnom, prelazi se na poslednju fazu životnog ciklusa.

7.6. Faza prikazivanja odgovora

U poslednjoj fazi ciklusa, JSF implementacija delegira ovlašćenje za prikazivanje strane JSP kontejneru (ukoliko aplikacija koristi JSP strane).

Ako je u pitanju inicijalni zahtev, komponente koje su prikazane na strani će biti dodate stablu komponenti kada JSP kontejner pokrene stranu. Ako zahtev nije inicijalni, komponente su već dodate stablu pa nema potrebe da se ponovo dodaju. Ako je u pitanju ponovni zahtev i ako su se javile greške tokom jedne od faza (dodele vrednosti, validacije ili ažuriranja vrednosti), prikazaće se originalna strana. Nakon što je sadržaj stabla prikazan, stanje odgovora ostaje sačuvano, tako da bude dostupno prvoj fazi narednog zahteva.
Kada je poslednja faza ciklusa završena, web kontejner fizički prenosi rezultujuće bajtove, a zatim pretraživač izvrši renderovanje.

Evo i grafičkog prikaza životnog ciklusa JSF strane.



8. Kjučni koncepti JSF-a

Kao i mnoge druge tehnologije i JSF tehnologija ima svoj skup pojmova koji čine konceptualnu osnovu mogućnosti koje ona pruža. Da bi se razvila jedna JSF aplikacija, neophodno je razumeti svaki od sledećih ključnih koncepata:

1. UI komponente
2. Rendereri
3. *Backing bean*-ovi
4. Validatori
5. Konverteri
6. Poruke
7. Događaji i osluškivači događaja
8. Navigacija

Pre nego što pređemo na detaljniji pregled funkcija i svojstava vezanih za prethodno nabrojane pojmove, osvrnućemo se ukratko na svaki od njih. UI komponente koje su sadržane u okviru pogleda ažuriraju *backing bean*-ove i generišu događaje u zavisnosti od ulaznih podataka korisnika. Rendereri služe za prikaz komponenti, a takođe mogu da generišu događaje i poruke. Konverteri prevode i formatiraju vrednost komponente za prikaz i generišu poruke o greškama. Validatori verifikuju vrednosti komponenti i takođe mogu da generišu poruke o greškama. *Backing bean*-ovi sadrže osluškivače događaja i akcione metode (*action methods*). Akcione metode su vrsta osluškivača specijalizovani za navigaciju. Osluškivači događaja pokreću događaje i mogu da manipulišu pogledom ili da izvršavaju radnje nad objektima modela, čime se i sprovodi sva aplikativna logika. Akcione metode imaju sve mogućnosti kao i osluškivači događaja, s tim da pored toga imaju posebnu povratnu vrednost (*outcome*) koja se koristi za navigaciju. Sistem za navigaciju koristi tu povratnu vrednost za određivanje narednog pogleda koji će se prikazati korisniku. Događaji predstavljaju neku vrstu posledice korisnikovih akcija. Poruke ukazuju na greške ili druga obaveštenja.

8.1. UI komponente

UI komponente (komponente korisničkog interfejsa ili jednostavno kontrole) su objekti koji se održavaju na serveru i koji obezbeđuju određene funkcionalnosti za interakciju sa krajnjim korisnikom. Kada kažemo da su UI komponente *stateful* objekti, to se odnosi na svojstvo koje omogućava održavanje (čuvanje ili pamćenje) stanja tih objekata između zahteva korisnika. Osim ovog, postoji još jedno bitno svojstvo UI komponenti. Kada se kaže da su UI komponente *reusable*, to se odnosi na mogućnost njihovog ponovnog korišćenja. Jedna ista komponenta se može neograničen broj puta koristiti i to bez pisanja zamornog i ponavljajućeg koda. Ovo zaista olakšava posao programerima i skraćuje vreme izrade korisničkih interfejsa, jer kada se jednom napravi, JSF komponenta se samo uključuje u JSP stranicu.

JSF tehnologija omogućava jednostavno ugrađivanje već postojećih *(third-party)* komponenti u web aplikaciju. Korišćenje ovih komponenti otklanja potrebu za ponovnim pisanjem već postojećih komponenti, pa na taj način smanjuje vreme potrebno za izradu aplikacije.

> *Third-party* softverske komponente (komponente treće strane) su komponente razvijene ili da budu besplatno distribuirane ili su prodate od strane lica koje ne predstavlja izvornog proizvođača platforme nad kojom je sama komponenta izgrađena. (Više informacija o ovome može se naći u [11].)

Što se izgleda komponente tiče, to zavisi od toga kako je renderovana. Međutim, nezavisno od toga kako je renderovana, komponenta ima iste funkcionalnosti. Ova odlika - neutralnost u odnosu na renderer, ključna je prilikom razvoja novih komponenti. Ipak, izgradnja UI komponenti korišćenjem JSF-a više se zasniva na sastavljanju i konfigurisanju postojećih

komponenata, nego na pisanju zamornog koda, korišćenjem različitih tehnologija kao što su HTML, CSS, JavaScript i sl.
JSF specifikacija sadrži skup standardnih UI komponenti i obezbeđuje API za njihov razvoj. Pored toga, već postojeće komponente mogu se proširivati. Drugim rečima, postoje dve vrste UI komponenti:

1. Standardne UI komponente
2. *Custom* (Prilagođene) UI komponente

Standardne komponente JSF-a raspoređene su u dve biblioteke:

1. JSF Core Tag Library
2. JSF Html Tag Library

Da bi ove komponente mogle da se koriste, potrebno je uključiti ih u svaku JSP stranicu korošćenjem taglib direktive, na sledeći način:

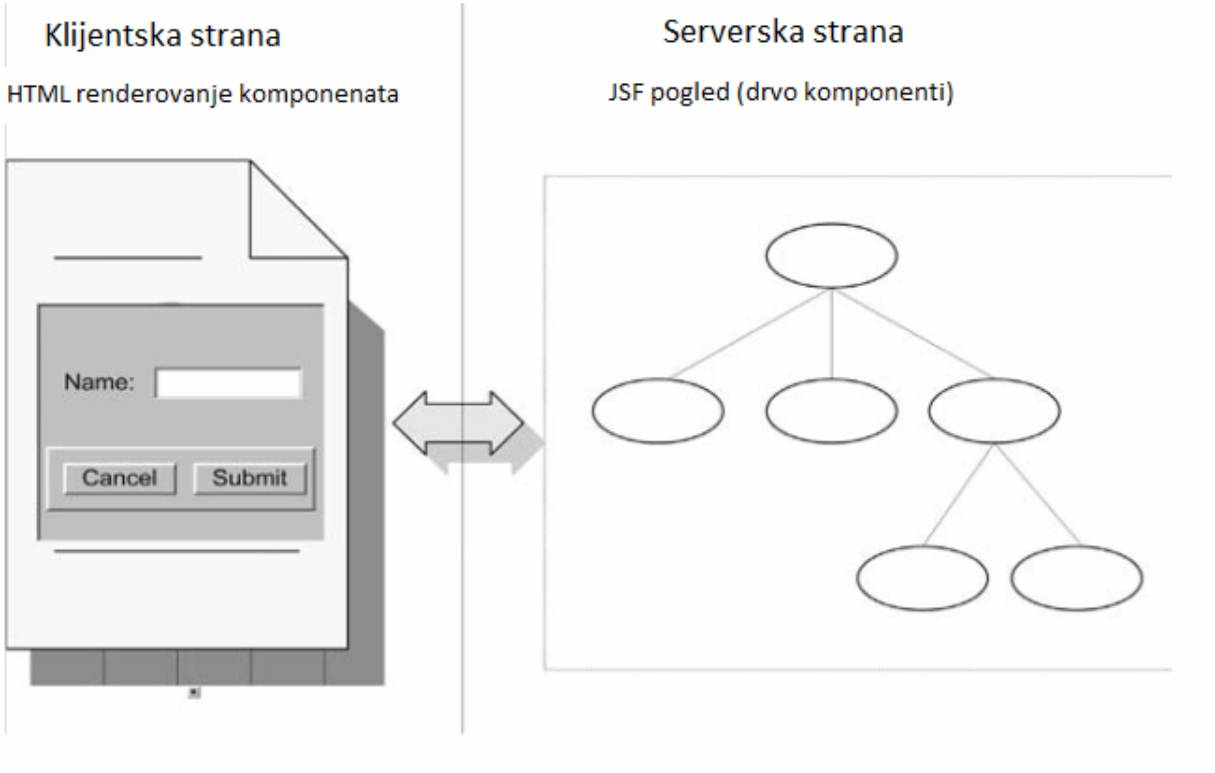
```
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
```

U sledećim tablicama navedeni su svi JSF Core i Html tagovi.
(Tablice su preuzete iz [2].)

JSF:CORE TAGS	JSF:HTML TAGS
actionListener	column
attribute	commandButton
convertDateTime	commandLink
convertNumber	dataTable
converter	form
facet	graphicImage
loadBundle	inputHidden
parameter	inputSecret
selectItem	inputText
selectItems	inputTextarea
subview	message
validateDoubleRange	messages
validateLength	outputLabel
validateLongRange	outputLink
validator	outputMessage
valueChangeListener	outputText
verbatim	panelGrid
view	panelGroup
	selectBooleanCheckbox
	selectManyCheckbox
	selectManyListbox
	selectManyMenu
	selectOneListbox
	selectOneMenu
	selectOneRadio

Jedna od ključnih razlika između *web*-zasnovanih i *desktop*-zasnovanih komponenti je u tome što prve nikada direktno ne komuniciraju sa mašinom korisnika. Recimo, ako se neispravno popuni forma *desktop* aplikacije i takva prosledi, stranica se neće prikazati ponovo, već se dobija obaveštenje o grešci, a sve vrednosti unete u formu ostaju. Kod web aplikacija, stranica se prikazuje ponovo, zajedno sa porukom o grešci, a samo izgleda kao da se stranica nije ponovo prikazala. Drugim rečima, komponente JSF-a automatski pamte vrednosti, odnosno stanja. JSF komponente imaju mogućnost da pamte vrednosti, odnosno stanja između zahteva, zahvaljujući tome što okruženje čuva (održava) drvo komponenti za datu stranicu. Ovo drvo UI komponenti, koje nazivamo još i pogled, je JSF-ova interna reprezentacija stranice i omogućava *parent-child* (roditelj-dete) odnose, kao što je recimo korišćenje forme u koju su ubačene druge komponente (labele, tekstualna polja i sl.).

Na primer:



Svaka kontrola u okviru drveta ima svoj jedinstveni identifikator. Ovaj identifikator komponente može da postavi sam programer, a ako to eksplicitno ne uradi, Id se generiše automatski.

8.2. Rendereri

Jedan od najvažnijih aspekata korisničkog interfejsa web aplikacije jeste kako ga korisnici vide odnosno kako izgleda. JSF obezbeđuje efikasan mehanizam za renderovanje odgovora na zahtev korisnika. Postoje dva oblika renderovanja. To su:

1. Direktno renderovanje – renderovanje u okviru same UI komponente
2. Delegirano renderovanje – renderovanje izvan UI komponente, korišćenjem *Render Kit*-a.

Prilikom delegiranog renderovanja, ovaj proces obavljaju posebne klase, koje se nazivaju rendereri. Rendereri su organizovani u *Render Kit*-ove. *Render Kit* je biblioteka renderera. Predstavljen je apstraktnom klasom *javax.faces.render.RenderKit*. *Render Kit* odnosno kolekcija renderera implementira izgled (*skin*) za određeni uređaj (telefon, PC ili *markup* jezik kao što je HTML, WML ili SVG). Velika prednost JSF-a je što se ne ograničava na određeni uređaj ili *markup*. Ipak, podrazumevani, tj. standardni Render Kit JSF specifikacije sadrži renderere za HTML. Svaka JSF implementacija mora da podržava ovaj *Render Kit*.

Renderer se može smatrati nekom vrstom prevodioca između klijentskog i serverskog sveta. To je klasa odgovorna za kodiranje i dekodiranje. Kada JSF implementacija prihvata zahtev korisnika, renderer obavlja dekodiranje. Dekodiranje je proces izvlačenja parametara HTTP zahteva i postavljanje vrednosti komponente (instance komponente na serveru) na datu vrednost parametra. S druge strane, kodiranje je proces prikaza komponenata korisniku, koji se odvija prilikom slanja odgovora na zahtev.

Ukoliko se vrši direktno renderovanje, UI komponenta mora da kodira i dekodira samu sebe prevazilaženjem (*overriding*) metoda renderovanja definisanih u okviru *UIComponentBase* klase. Reč je o metodama *decode()*, *encodeBegin()*, *encodeChildren()*, *encodeEnd()*. *Decode()* metoda se poziva nakon prihvaćenog zahteva, a metode kodiranja se pozivaju kada JSF priprema odgovor na zahtev. Ako komponenta koja se renderuje nema dete-komponente, treba implementirati samo *encodeEnd()* metodu. Kada se vrši direktno renderovanje komponente, još treba implementirati metodu *setRendererType()* tako da vrati *null*.

U današnje vreme postoji veliki broj uređaja koji imaju pristup internetu. Stoga se programeri suočavaju sa izazovom razvijanja komponenata koje mogu da se koriste na različitim platformama. Da bi se omogućilo funkcionisanje komponenata na različitim uređajima, neophodno je renderovati ih na više različitih načina. Za JSF to ne predstavlja problem jer omogućava razvoj renderera. Renderere kreiramo uglavnom u situacijama kada imamo posla sa *custom* UI komponentama. U tom slučaju, neophodno je registrovati renderer. To se vrši u okviru *faces.config* fajla. Međutim, u slučaju korišćenja standardnih JSF UI komponenti, podrazumeva se da svakoj od njih odgovara renderer standardnog HTML *Render Kit*-a. Na osnovu upotrebljenog taga, tačno se zna koji će renderer biti primenjen.
Na slici sa naredne strane može se videti lista standardnih JSF HTML renderera. (Tabela je preuzeta iz [2].)

STANDARDNI JSF HTML RENDERERI

RENDERER / TAGS	UICOMPONENTS
Button / <commandButton>	UICommand
Web Link / <commandHyperlink>	UICommand
Table / <dataTable>	UIData
Form / <form>	UIForm
Image / <graphicImage>	UIGraphic
Hidden / <inputHidden>	UIInput
Secret / <inputSecret>	UIInput
Input Text / <inputText>	UIInput
TextArea / <inputTextArea>	UIInput
Label / <outputLabel>	UIOutput
Output Link / <outputLink>	UIOutput
Output Text / <outputText>	UIOutput
Grid / <panelGrid>	UIPanel
Group / <panelGroup>	UIPanel
Checkbox / <selectBooleanCheckbox>	UISelectBoolean
Checkbox List / <selectManyCheckbox>	UISeletMany
Listbox / <selectManyListbox> <selectOneListbox>	UISelectMany / UISelectOne
Menu / <selectManyMenu> <selectOneMenu>	UISelectMany / UISelectOne
Radio / <selectOneRadio>	UISelectOne

8.3. Backing beans

U okviru JSF aplikacije, objekti koji imaju ulogu da vrše interakciju između modela i pogleda (podataka i korisničkog interfejsa), zovu se *backing bean* objekti. *Backing bean*-ovi sadrže atribute za preuzimanje unosa korisnika i osluškivače događaja koji upravljaju ovim atributima, kao i korisničkim interfejsom ili sprovode neke druge obrade zahteva. *Backing bean* klase su JavaBean klase koje sadrže kod koji se nalazi u pozadini aplikacije, tzv. *code-behind.*

JSF omogućava deklarativno pridruživanje *backing bean*-ova komponentama korisničkog interfejsa. Kad se kaže deklarativno, misli se bez korišćenja koda, odnosno u ovom slučaju korišćenjem *JSF Expression Language*-a.

Na primer:

```
<h:inputText value="#{prvaBean.ime}" id="IME" />
```

Ovim delom koda se povezuju vrednost *HtmlInputText* komponente sa *ime* atributom objekta *prvaBean* (*prvaBean* je instanca klase *PrvaBean*). Kad god se promeni vrednost komponente, promeni se i vrednost *prvaBean.ime* atributa i obrnuto. Drugim rečima, UI komponenta i odgovarajući *backing bean* atribut su automatski sinhronizovani. Ovo je ključno svojstvo JSF-a i predstavlja tipičan način povezivanja *backing bean* atributa sa vrednošću UI komponenata. Pored toga, moguće je direktno povezivanje *backing bean* atributa sa instancom komponente na serveru. U tom slučaju se ne koristi *value* već *binding* atribut UI komponente. Ovaj način povezivanja (*binding*) je koristan kada je komponentom potrebno manipulisati korišćenjem Java koda. Ovakav vid procesiranja se najčešće sprovodi preko osluškivača događaja.

Na primer:

```
<h:panelGrid id="controlPanel"
binding="#{helloBean.controlPanel}"
columns="20" border="1" cellspacing="0"/>
```

Binding atribut komponete korišćenjem JSF EL izraza povezuje *HtmlPanelGrid* komponentu sa *controlPanel* atributom *backing bean*-a *helloBean*.
Kada se korišćenjem osluškivača izvrše određene promene, one su vidljive prvi naredni put kada se stranica prikaže korisniku.

Jedan pogled može imati više od jednog *backing bean*-a. Postoje alati koji za novokreiranu stranicu aplikacije generišu *backing bean* klase automatski. Neki alati automatski generišu i povezivanja UI komponenata sa odgovarajućim *backing bean* atributima.

Backing bean-ovi često komuniciraju sa objektima modela (recimo, pomoćnim klasama koje pristupaju bazama podataka).
Da programeri ne bi gubili vreme na zamarajući kod kojim se kreiraju *backing bean*-ovi i objekti modela, JSF obezbeđuje deklarativni mehanizam koji olakšava ovaj postupak, a koji se naziva *Managed Bean Creation facility*. Ovime je moguće navesti koji će objekti biti dostupni tokom životnog ciklusa aplikacije.

Na primer:

```
<managed-bean>
  <managed-bean-name>prvaBean</managed-bean-name>
  <managed-bean-class>PrvaBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
```

Na ovaj način JSF kreira instancu klase PrvaBean po imenu prvaBean i čuva je u sesiji korisnika. Tako objekat prvaBean postaje dostupan za integrisanje sa UI komponentama. Svaki objekat koji je na ovaj način uključen u aplikaciju naziva se *managed bean*.

8.4. Validatori

Preko validatora se vrši provera da li je korisnik prosledio ispravnu vrednost preko neke UI komponente na stranici. Postoji nekoliko načina koje JSF koristi za upravljanje validacijom:

1. Korišćenje standardnih (ugrađenih) komponenata za validaciju
2. Korišćenje metoda validacije iz *backing bean*-ova
3. Kreiranje *custom* validatora

JSF se isporučuje zajedno sa skupom standardnih (ugrađenih tj. podrazumevanih) komponenata za validaciju. Te komponente su predstavljene određenim klasama, a svakoj klasi odgovara i jedan tag iz JSF *core* biblioteke. Ovaj podrazumevani skup validatora obuhvata:
LongRangeValidator, *DoubleRangeValidator* i *LengthValidator*, kojima odgovaraju redom tagovi: `<f:validateLongRange/>`, `<f: validateDoubleRange/>` i `</validateLength>`.
Backing bean metode validacije se lako implementiraju, ali su ograničene na korišćenje u okviru jedne aplikacije. Kreiranu metodu validacije pridružujemo odgovarajućoj komponenti korišćenjem kombinacije njenog *validator* atributa i *Expression Language*-a. Metode validacije moraju da imaju potpis u sledećem obliku:

public void *methodName*(FacesContext context ,
UIComponent component,
Object value)
throws ValidatorException

Kao što je već rečeno, *backing bean* metode su jednostavne za implementaciju. Međutim, ukoliko želimo validatore, koje možemo da napišemo jednom, a da ih koristimo bilo gde, najbolje je koristiti *custom* validatore, odnosno kreirati klase koje implementiraju interfejs *javax.faces.validator.Validator*. U okviru te klase neophodno je implementirati jedinstveni metod ovog interfejsa, metod *validate()*. Nakon toga, da bi ovaj validator mogao da se koristi, potrebno ga je registrovati u okviru *faces.config* fajla. Kreirani validator se pridružuje UI komponenti korišćenjem `<f: validator/>` taga i njegovog atributa `validatorId` koji mora da odgovara `<validator-id/>` tagu iz *faces.config*-a. Svakoj komponenti se može pridružiti kako jedan validator, tako i više njih.

Proces validacije se odvija na serveru. Kada se tokom ovog procesa pojavi greška, poruka o grešci se dodaje u *FacesContext*, u tekuću listu grešaka. U tom slučaju se iz faze validacije direktno prelazi na fazu prikazivanja odgovora, što znači da vrednosti *backing bean*-ova ostaju nepromenjene. U slučaju da je *immediate* atribut UI komponente postavljen na *true*, validacija se vrši u okviru druge faze ciklusa (faze dodele vrednosti).

8.5. Konverteri

Kada korisnik vrši interakciju sa JSF aplikacijom, interakcija se ustvari vrši sa izlazom renderovanja. Rendereri, kao što je već poznato daju posebnu reprezentaciju strane, pogodnu za određenog klijenta (kao što je web pretraživač). Da bi migli ovo da sprovedu, rendereri moraju imati posebno znanje o komponentama koje treba da prikažu. Međutim, komponente mogu biti povezane sa atributima *backing bean*-ova, a ovi atributi mogu biti bilo kog tipa (*String*-kojim se predstavlja ime, *Date*-kojim se predstavlja datum rođenja ili bilo kog drugog tipa). Pošto ne postoje ograničenja za tip, rendereri ne mogu unapred da znaju kako da prikažu objekat. Zbog toga konverteri imaju bitnu ulogu. Oni prevode objekat u *String* za prikaz i prevode uneti *String* nazad u odgovarajući objekat.

Proces konverzije se odvija u drugoj fazi životnog ciklusa JSF stranice.
Ako je *immediate* atribut neke ulazne komponente postavljen na *true*, onda se pored konverzije u ovoj fazi vrše i validacija i procesiranje događaja koji su pridruženi toj komponenti.

Kao i kod validacije, i kod konverzije postoje uobičajeni načini kojima se ovaj postupak sprovodi:

1. Korišćenjem *converter* atributa u okviru taga komponente
2. Ugnježdavanjem tagova standardnih konvertera unutar tagova komponente
3. Korišćenjem *custom* konvertera

Atribut *converter*, koji je podržan od strane nekih JSF tagova, omogućava deklarativno navođenje konvertera koji će se koristiti. Tagovi koji podržavaju ovaj atribut su: *inputText*, *inputSecret*, *inputHidden* i *outputText*. Na ovaj način se mogu koristiti konverteri koji su isporučeni sa implementacijom JSF-a, ali isto tako se mogu koristiti i registrovani *custom* konverteri.
U situacijama kada je potrebno podesiti dodatna svojstva konvertera, primenjuje se ugnježdavanje tagova konvertera unutar tagova komponenti.
JSF obezbeđuje *NumberConverter* i *DateTimeConverter* klase koje podržavaju atribute za finije podešavanje ponašanja tih konvertera.

Na primer:

Postavljanjem atributa *type*, kao u narednom primeru, korisnik se obavezuje da ispred unetog iznosa unese i oznaku valute.

```
<h:inputText value=”#{modifyInvoicePage.invoice.amount}”>
<f:convertNumber type=”currency”/>
</h:inputText>
```

Svaka JSF implementacija mora da obezbedi skup standardnih konvertera. Klase tih konvertera su date u sledećoj tabeli :
(Tabela je preuzeta iz [2].)

PODRAZUMEVANE KLASE KONVERTERA

CONVERTER CLASS	CONVERTER ID
BigDecimalConverter	BigDecimal
BigIntegerConverter	BigInteger
NumberConverter	Number
IntegerConverter	Integer
ShortConverter	Short
ByteConverter	Byte
CharacterConverter	Character
FloatConverter	Float
DoubleConverter	Double
BooleanConverter	Boolean
DateTimeConverter	DateTime

Pored ovoga, moguće je kreirati i *custom* konvertere. Custom konverteri se kreiraju definisanjem klasa koje implementiraju interfejs *javax.faces.converter.Converter*, odnosno koje implementiraju dve metode:

1. *getAsObject()* – *UIinput* komponenta poziva ovaj metod da bi se prosleđeni *String* konvertovao u tip odgovarajućeg *bean* atributa.
2. *getAsString()* – *UIinput* komponenta poziva ovaj metod da bi se izvršila konverzija u *String* za prikaz.

Nakon definisanja ove klase, neophodno je konverter registrovati da bi on postao upotrebljiv unutar aplikacije. Ovo se radi u okviru *faces.config* fajla.
Registrovani konverter se koristi upotrebom *converter* atributa komponente, koja to podržava ili upotrebom `<f:converter />` taga i njegovog `converterId` atributa koji mora da se poklapa sa `<converter-id>` tagom iz konfiguracionog fajla. *Custom* konverteri se, prilikom registrovanja, mogu konfigurisati tako da budu podrazumevani za određeni tip. Inače, u okviru JSF implementacije postoji skup standardnih konvertera koji omogućavaju konvertovanje i kada se u okviru *UIinput* komponente ne navede nijedan konverter. U tom slučaju se JSF u vreme izvršavanja sam pobrine za pronalaženje odgovarajućeg konvertera. U narednoj tabeli nalazi se spisak konvertera među kojima JSF traži pogodnog za određenu kontrolu.(Tabela je preuzeta iz [2].)

CONVERTER CLASS	CONVERTER ID	NATIVE TYPE	JAVA CLASS
BigDecimalConverter	BigDecimal		java.math.BigDecimal
BigIntegerConverter	BigInteger		java.math.BigInteger
NumberConverter	Number		java.lang.Number
IntegerConverter	Integer	int	java.lang.Integer
ShortConverter	Short	short	java.lang.Short
ByteConverter	Byte		java.lang.Byte
CharacterConverter	Character	char	java.lang.Character
FloatConverter	Float	float	java.lang.Float
DoubleConverter	Double	double	java.lang.Double
BooleanConverter	Boolean	boolean	java.lang.Boolean

Svakoj ulaznoj komponenti moguće je pridružiti jedan konverter. Ukoliko prilikom konverzije dođe do neke greške, te greške je moguće prikazati dodavanjem *message* ili *messages* taga u JSP stranicu.

8.6. Poruke

Još jedno važno pitanje prilikom razvoja korisničkog interfejsa aplikacije jeste kako rukovoditi greškama.

Greške se, pre svega, mogu podeliti na dve glavne kategorije:

1. Aplikativne greške (vezane za poslovu logiku, bazu podataka, greške pri konekciji i sl.)
2. Korisničke (*input*) greške (greške koje nastaju pri unosu, recimo prilikom unetog neispravnog teksta ili prilikom prosleđivanja praznog polja)

Aplikativne greške obično generišu skroz novu stranicu sa prikazanim greškama, dok korisničke obično utiču da se ponovo prikaže ista stranica, ali zajedno sa tekstom koji opisuje grešku. Ono o čemu treba voditi računa jeste da, za svaku stranicu, ista greška proizvede istu poruku.
JSF obezbeđuje poruke, koje pomažu da se rukovodi greškama. Poruka se sastoji od:

1. Rezimea poruke (kratkog pregleda tzv. *summary text*-a)
2. Detaljnog teksta (*Detailed text*)
3. Stepena jačine poruke (*Severity level*, koji može biti : *Info*, *Warning*, *Error* i *Fatal*)

Gotovo svaki deo aplikacije-uključujući UI komponente, validatore, konvertere ili osluškivače događaja, može da doda poruku tokom procesiranja zahteva. JSF čuva listu svih tekućih poruka u okviru *FacesContext*-a.
Poruke ne moraju uvek ukazivati na greške. One mogu biti i informativnog karaktera (na primer, neki osluškivač događaja može dodati poruku koja indicira da je novi element dodat u bazu podataka).
Poruka može biti povezana sa određenom komponentom kada treba da prikazuje korisničke greške, ili može biti samostalna kada treba da prikazuje aplikativne greške. Greške vezane za određenu UI komponentu, mogu se prikazati korišćenjem *HtmlMessage* komponente.

Na primer:

```
<h:message id="errors" for="IME" style="color: red"/>
```

Ovim tagom se prikazuju sve greške generisane za komponentu sa identifikatorom IME (koja se mora nalaziti na istoj stranici).

Pored toga, sve poruke ili one koje nisu vezane za određenu komponentu mogu se prikazati korišćenjem *HtmlMessages* komponente.

Očigledno je da poruke predstavljaju zgodan način za informisanje korisnika o greškama i za prikaz drugih obaveštenja. One su sastavni deo JSF-ove validacije i konverzije. Kad god se neki validator susretne sa neodgovarajućom vrednošću ili kad konverter naiđe na neodgovarajući tip, generiše se poruka o grešci. Poruke se takođe mogu definisati i u okviru metoda pridruženih osluškivačima, pa će prilikom izvršavanja tih metoda i biti prikazane.

8.7. Događaji i osluškivači

Događaji i osluškivači događaja predstavljaju veoma veliki i važan napredak u razvoju aplikacija jer oslobađaju programera od razmišljanja na nivou zahteva i odgovora.
Kod JSF aplikacija, integrisanje aplikativne logike je stvar dodeljivanja odgovarajućih osluškivača komponentama koje generišu događaje (događaje koje osluškivači razumeju).

Postoji četiri standardna tipa događaja:

1. Događaji promene vrednosti (*Value-chenge events*)

2. Akcioni događaji (*Action events*)
3. Događaji modela podataka (*Data model events*)
4. Fazni događaji (*Phase events*)

8.7.1. Događaji promene vrednosti

Kada korisnik izvrši promenu vrednosti neke ulazne komponente, tada ta komponenta generiše događaj promene vrednosti. Ovi događaji se obrađuju korišćenjem *value-change* osluškivača (osluškivača za promenu vrednosti). Osluškivači za promenu vrednosti se u okviru ulaznih komponenata koriste navođenjem *valueChangeListener* atributa, koji ukazuje na odgovarajuću metodu *backing bean*-a (*value-change* listener metodu) koja procesira generisani događaj.

Na primer:

```
<h: inputText
valueChangeListener="#{myBean.processValueChanged}"/>
```

Drugi način primene metoda koje procesiraju događaje promene vrednosti jeste registrovanje posebnih klasa, koje implementiraju interfejs javax.faces.event.ValueChangeListener.
Ipak, u većini slučejeva je dovoljno primenjivati pridruživanje *backing bean* metode komponenti.

8.7.2. Akcioni događaji

Akcioni događaji nastaju kada korisnik aktivira neku komandnu komponentu, recimo kada klikne dugme ili hiperlink. Komponente koje generišu ovu vrstu događaja nazivaju se *Action sources*.
Za obradu akcionih događaja koriste se osluškivači akcija (*action listeners*).
Postoje dve vrste osluškivača akcija:

1. Oni koji sprovode navigaciju

Ova vrsta oslučkivača koristi se navođenjem *action* atributa UI komponente. U okviru ovog *action* atributa, navodi se tzv. *outcome* (ishod, povratna vrednost), koji navigacioni sistem JSF-a koristi da odredi koja stranica treba sledeća da se prikaže korisniku. Pri tome, ta stranica može biti i ista ona sa koje je korisnik poslao zahtev.
Postoje 2 osnovna tipa *outcome*-a (ishoda). To su: statički i dinamički.

Statički ishod je string koji se eksplicitno navodi u okviru *action* atributa komponente. U tom slučaju, akcioni događaj ne prouzrokuje izvršavanje nijedne akcione metode.

Na primer:

```
<h:commandButton value="PRETHODNA" action="prva"/>
<h:commandButton value="SLEDEĆA" action="treca" />
```

Dinamički ishod je string koji kao povratnu vrednost imaju akcione metode *backing bean*-a.

Na primer:

```
<h:commandButton value="SLEDEĆA" action="#{prvaBean.sledeca}"/>
```

U zavisnosti od aplikativne logike, jedna akciona metoda može da vrati različite ishode (povratne vrednosti).
Dakle, kada korisnik klikne recimo na dugme ili hiperlink, to prouzrokuje akcioni događaj, a zatim se izvršava metoda *backing bean*-a (**action metod**) pridružena odgovarajućem *action* atributu komponente koja je prouzrokovala događaj.

2. Oni koji sprovode druge vrste akcija, nevezane za navigaciju

Ovi osluškivači se koriste navođenjem *actionListener* atributa unutar komponente.

Na primer:

```
<h:commandButton id="redisplayCommand" type="submit" value="Redisplay"
actionListener="#{myForm.doIt}"/>
```

Kada korisnik klikne na dugme ili hiperlink, to prouzrokuje akcioni događaj, a zatim se izvršava metoda *backing bean*-a (**action listener metod**) pridružena odgovarajućem *actionListener* atributu komponente koja je prouzrokovala događaj.
Nakon izvršavanja ove metode, stranica se prikazuje ponovo zajedno sa promenama nastalim izvršavanjem pozvane metode.

Za razliku od akcionih metoda koje rukovode navigacijom, ove druge metode imaju pristup komponentama koje su prouzrokovale događaj.
Akcione metode za navigaciju nemaju parametre, a imaju povratnu vrednost (*outcome*), dok je kod *action listener* metoda situacija obrnuta. One imaju *ActionEvent* parametar, ali nemaju povratnu vrednost.

Najčešće, korišćenje prethodno pomenutih metoda u okviru *backing bean*-ova je sasvim dovoljno za sprovođenje aplikativne logike, iako se pored toga osluškivači događaja mogu implementirati i definisanjem klase koja implementira *ActionListener* interfejs odnosno implementacijom njegove *processAction()* metode.
Ukoliko se koristi prvi način, tada je upotreba ograničena na jedinstveni osluškivač, dok se prilikom rada sa interfejsom može koristiti više njih.

8.7.3. Događaji modela podataka

Ovi događaji nastaju kada komponenta koja je povezana sa podacima modela (*data-aware* komponenta) procesira podatke modela. Najčešći način za pokretanje ove vrste događaja je korišćenje komponente koja prikazuje listu elemenata modela koji se mogu selektovati (na primer, *HtmlDataTable*).
Za razliku od *value-change* i *action event* osluškivača, *data model* osluškivači moraju da implementiraju Java interfejs.
Data model događaji se razlikuju od prethodno spomenutih jer njih ne iniciraju UI komponente, već instance *Data Model*-a (modela podataka), tj. objekti modela koje komponente interno koriste za rad sa podacima. S obzirom na ovu činjenicu, osluškivači se ne mogu pridruživati komponentama u okviru JSP strane, već se moraju registrovati Java kodom.

8.7.4. Fazni događaji

Kad god JSF aplikacija prihvati zahtev za stranicom, ona prolazi kroz proces koji se zove *Request Processing Lifecycle*, tj. životni ciklus procesiranja zahteva. Kao što je već poznato, tokom ovog procesa, JSF vrši razne akcije, kao što su: formiranje stabla komponenti, prevođenje parametara zahteva u vrednosti komponenata, validaciju, ažuriranje *backing bean*-ova i objekata modela, pozivanje osluškivača događaja i prikazivanje korisniku odgovora na zahtev. Fazni događaji se generišu pre i posle svakog od koraka životnog ciklusa. Oni su češće generisani od strane samog JSF-a, nego od strane UI komponente, ali dešava se da budu korisni i prilikom razvoja aplikacije. Fazni događaji zahtevaju implementaciju Java interfejsa.

8.8. Navigacija

Web aplikacija se sastoji od više stranica, pa je neophodno da postoji način na koji se one povezuju. Prelaženje sa jedne stranice aplikacije na drugu naziva se navigacija.

JSF ima veoma zgodan i jednostavan navigacioni sistem, jer ne zahteva kodiranje, već samo određena podešavanja. Procesom navigacije rukovodi *navigation handler* (instanca klase *javax.faces.application.NavigationHandler*).
Pravila navigacije (*navigation rule*), definišu se u okviru JSF-ovog konfiguracionog fajla (*faces.config* fajla) korišćenjem `<navigation-rule>` elementa. Za svaku stranicu, pravila navigacije definišu za koji se *outcome* (povratnu vrednost *action* metoda) na koju stranicu prelazi. Svako mapiranje *outcome*-a stranicom naziva se *navigation case.*

Na primer:

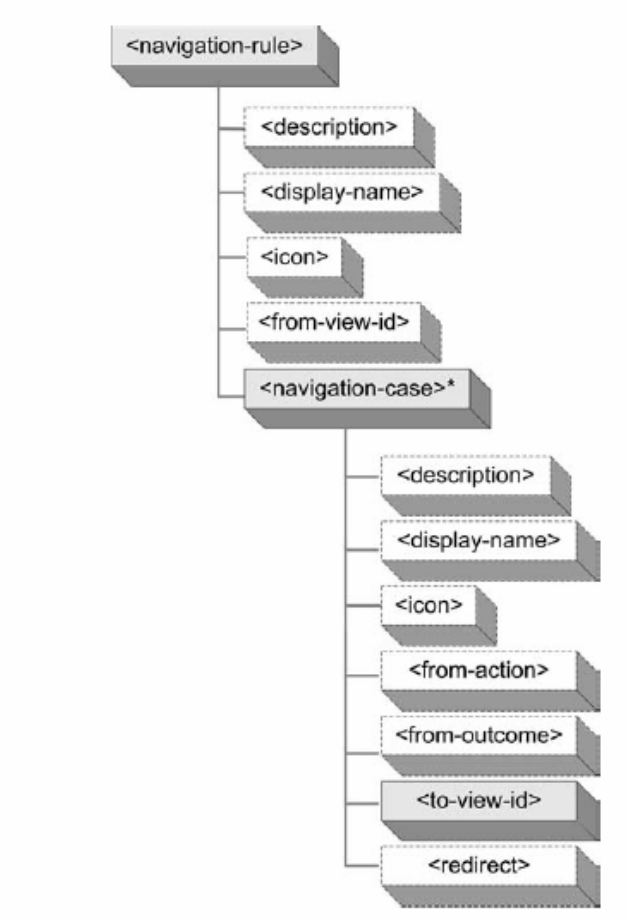
```
<navigation-rule>
  <from-view-id>/prva.jsp</from-view-id>
  <navigation-case>
   <from-outcome>druga</from-outcome>
   <to-view-id>/druga.jsp</to-view-id>
  </navigation-case>
 </navigation-rule>
 <navigation-rule>
  <from-view-id>/druga.jsp</from-view-id>
  <navigation-case>
   <from-outcome>treca</from-outcome>
   <to-view-id>/treca.jsp</to-view-id>
  </navigation-case>
 </navigation-rule>
 <navigation-rule>
  <from-view-id>/treca.jsp</from-view-id>
  <navigation-case>
   <from-outcome>druga</from-outcome>
   <to-view-id>/druga.jsp</to-view-id>
  </navigation-case>
 </navigation-rule>
 <navigation-rule>
  <from-view-id>/druga.jsp</from-view-id>
  <navigation-case>
   <from-outcome>prva</from-outcome>
   <to-view-id>/prva.jsp</to-view-id>
  </navigation-case>
 </navigation-rule>
```

Za jednu istu vrednost `<from-view-id>` elementa može se napraviti više navigacionih pravila. Tada JSF ta pravila tretira kao jedno veliko kombinovano pravilo. Ovo je korisno ukoliko u okviru aplikacije ima više konfiguracionih fajlova, što se može desiti kada više programera radi na različitim delovima iste aplikacije.

Ukoliko želimo da budemo sigurni da je neki *outcome* generisan baš od strane određenog metoda, onda koristimo `<from-action>` element u okviru elementa `<navigation-case>`. Ovo može biti korisno kada dva različita *action* metoda vraćaju isti *outcome*.
Do sada je bilo reči o navigacionim pravilima koja su vezana za jednu određenu stranicu. Pored ovoga, moguće je postavljati pravila za sve stranice ili za grupu stranica.
Pravilo navigacije je globalno ako se u okviru `<from-view-id>` elementa postavi zvezdica (*, *asterisk*).
Čest slučaj je pridruživanje pravila celom direktorijumu, tj. svim stranicama u okviru tog direktorijuma. To se radi na sledeći način:

```
<from-view-id>/ime_direktorijuma/*</from-view-id>
```

Ukoliko aplikacija ima veliki broj pravila navigacije, poželjno je smestiti ih u odvojen konfiguracioni fajl.
U okviru `<navigation-case>` i `<navigation-rule>` elemenata postoje elementi `<description>, <display-name>, <icon>` koji se mogu opciono ubaciti.
Na sledećoj slici može se videti raspored navigacionih elemenata: (Slika je preuzeta iz [1].)



9. Primer jednostavne aplikacije

Aplikacija koja će biti predstavljena u narednom primeru sadrži tri stranice i jedan *backing bean*. To su:

- Prva.jsp sa odgovarajućim prvaBean.java *backing bean*-om
- Druga.jsp
- Treca.jsp

Uprkos tome što je veoma jednostavna, njom se uspešno mogu demonstrirati neki bitni koncepti JSF-a.

Na prvoj stranici nalaze se dva ulazna polja u koje je potrebno uneti ime i prezime, kao i dugme koje vodi na narednu stranu.
Na drugoj stranici se nalaze dve *outputText* komponente, koje imaju ulogu da prikažu ime i prezime uneto na prethodnoj (prvoj) strani. Pored toga, na drugoj strani se nalaze još dva dugmeta, za prelazak na prethodnu i sledeću stranu.
I na kraju, treća strana je veoma slična drugoj, s tom razlikom što ne sadrži dva dugmeta, već samo jedno, za povratak na prvu stranu.
Evo i koda za prethodno spomenute strane.

Prva.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1" pageEncoding="utf-8"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Prva Strana</title>
</head>
<body>
<f:view>
<h:form>
```

```
     <h:outputText value="UNESITE VAŠE IME I PREZIME..."
style="COLOR: blue; FONT-WEIGHT: bold; FONT-SIZE: x-large; FONT-
FAMILY: 'Calibri';"/>
     <br>
     <h:outputLabel value="Ime:" />       
     <h:inputText value="#{prvaBean.ime}" id="IME" />
     <h:message id="errors" for="IME" style="color: red"/>
     <br>
     <h:outputLabel for="PREZIME" value="Prezime: "/>
     <h:inputText value="#{prvaBean.prezime}" id="PREZIME"/>
     <br>
     <br>
     <h:commandButton value="SLEDEĆA"
action="#{prvaBean.sledeca}"/>

</h:form>
</f:view>
</body>
</html>
```

Druga.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1" pageEncoding="utf-8"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Druga strana</title>
</head>
<body>
<f:view>
<h:form>

     <h:outputText value="Pozdrav "/>
     <h:outputText value="#{prvaBean.ime} "/>
     <h:outputText value="#{prvaBean.prezime}"/>
     <br>
     <br>
     <br>
     <br>
```

```
     <h:commandButton value="PRETHODNA" action="prva"/>
     <h:commandButton value="SLEDEĆA" action="treca" />

</h:form>
</f:view>
</body>
</html>
```

Treca.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Treca strana</title>
</head>
<body>
<f:view>
<h:form>

     <h:outputText value="Pozdrav "/>
     <h:outputText value="#{prvaBean.ime} "/>
     <h:outputText value="#{prvaBean.prezime}"/>
     <br>
     <br>
     <br>
     <h:commandButton value="PRETHODNA" action="druga"/>

</h:form>
</f:view>
</body>
</html>
```

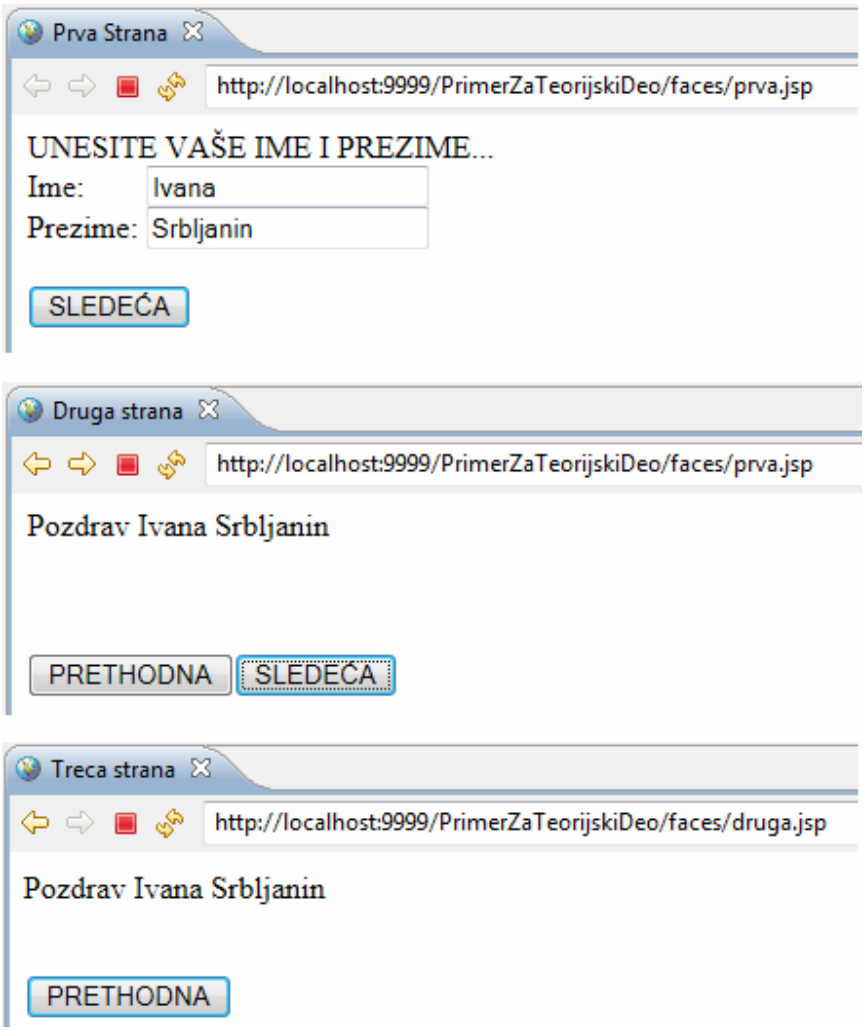
A evo i koda za **PrvaBean.java** klasu:

```
public class PrvaBean {

	private String ime;
	private String prezime;


	public String getIme() {
		return ime;
	}
	public void setIme(String ime) {
		this.ime = ime;
	}

	public String getPrezime() {
		return prezime;
	}
	public void setPrezime(String prezime) {
		this.prezime = prezime;
	}

	//action metod za navigaciju
	public String sledeca()
	{
		return "druga";
	}
}
```

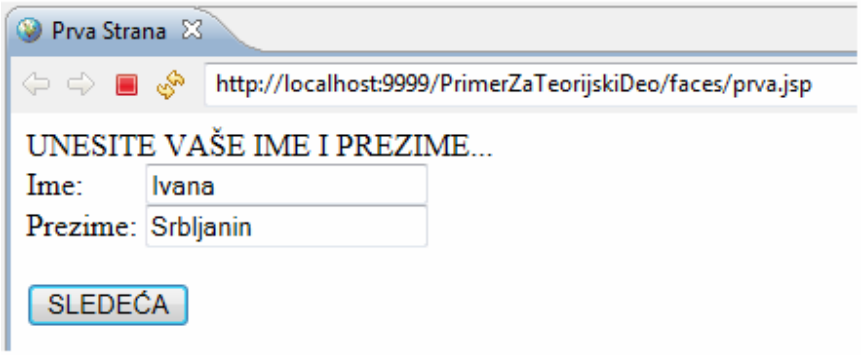
Najvažnije što će u ovom jednostavnom primeru biti prikazano je upotreba sesije.
Ukoliko je *backing bean* prve strane (prvaBean.java) definisan tako da mu je opseg važenja postavljen na *request*, uneto ime i prezime će samo jednom biti preneto na drugu stranicu. Već sledećim klikom na dugme PRETHODNA ili SLEDEĆA, ove vrednosti se gube. U praksi je najčešće potrebno da se unete vrednosti pamte, bez obzira koliko je puta korisnik promenio stranicu. Da bi se, bez obzira na navigaciju, održale unete vrednosti, neophodno je *backing bean* prve strane definisati tako da mu opseg važenja bude *session*. Ovo se, kao što je već poznato, vrši u okviru faces.config fajla.

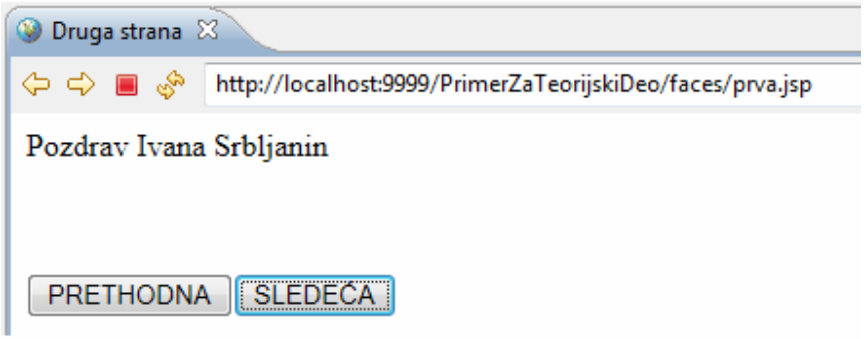
```
<managed-bean>
  <managed-bean-name>prvaBean</managed-bean-name>
  <managed-bean-class>PrvaBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
```

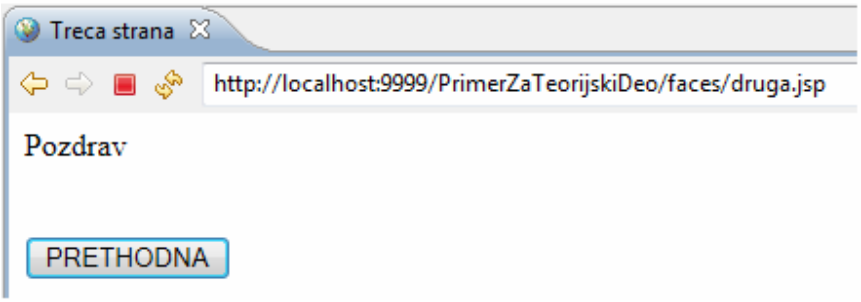
Ukoliko *bean* ima *session* opseg važenja (kao što je prethodno navedeno), tada će prolazak kroz stranice redosledom prva-druga-treća imati sledeći efekat:



Ukoliko, ipak, *bean* ima *request* opseg važenja, tada će prolazak kroz stranice redosledom prva-druga-treća imati sledeći efekat:







U okviru koda ove jednostavne aplikacije nalaze se neki delovi koji se u radu navode kao primeri za JSF EL, definisanje pravila navigacije i sl.

10. Teorijski opis prateće aplikacije

Prilikom izučavanja bilo koje nove oblasti, pa tako i ovde opisane JSF tehnologije, prilično je korisno razmotriti teorijske osnove na kojima se ona zasniva. Ipak, kada je u pitanju oblast kao što je programiranje, najbolje je učiti kroz praktičan rad. Zato će, nakon prethodno izloženog teorijskog dela o JSF-u, u ovom radu biti priložena i opisana jedna JSF aplikacija.

Aplikacija o kojoj će biti reči nosi naziv Opštinski info i uslužni servis. Ideja je da se kroz ovaj primer predstave ključni koncepti JSF-a i da se stekne jasnija slika o tome kako ovo okruženje zaista funkcioniše u praksi. Ova web aplikacija praktično predstavlja korisnički servis koji bi mogao da omogući ostvarivanje različitih usluga, kao što su naručivanje, rezervisanje i sl. Ovom vrstom interakcije između korisnika i aplikacije mogu se demonstrirati neke od mogućnosti, koje JSF tehnologija pruža. Ono što je u okviru ovog rada simulirano jeste poručivanje hrane preko interneta, s tim da aplikacija svakako ima mogućnost proširenja na druge, slične vrste usluga.

Pre nego što se počne izrada neke web aplikacije, potrebno je uraditi nekoliko stvari. Treba osmisliti njenu strukturu, obezbediti i podesiti potreban alat za njenu izradu i izabrati implementaciju tehnologije koja se koristi.
U izradi ove aplikacije primenjena je 1.2 verzija JSF tehnologije, a korišćena implementacija je Mojarra. Dodatno, korišćene su još dve biblioteke komponenti. To su: RichFaces i Ajax4Jsf.
Ove dve biblioteke obezbeđuju veliki broj UI komponenti, koje olakšavju izradu aplikacija time što u sebi poseduju već ugrađen Ajax. Ajax (Asynchronous JavaScript and XML) omogućava da se prilikom korisničkog zahteva ne vrši ponovno učitavanje kompletne web stranice, već samo neophodnog dela strane.
Dakle, pored standardnih biblioteka JSF-a, omogućeno je korišćenje još dve biblioteke komponenti, koje se na .jsp stranu uključuju direktivama:

```
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j" %>
<%@ taglib uri="http://richfaces.org/rich" prefix="rich" %>
```

Biblioteke JSF-a se preuzimaju sa interneta u vidu .jar fajlova.
Kao razvojno okruženje za izradu ove aplikacije koristi se Eclipse.
S obzirom da je reč o web aplikaciji koja se pokreće na serveru, bilo je potrebno izabrati i neki web server. U ovom slučaju, to je Apache Tomcat 7.0.27.
U velikom broju slučajeva, prilikom razvoja web aplikacija, baza podataka predstavlja neizostavan segment projekta. Zbog toga i ova aplikacija uključuje komunikaciju sa bazom, koja je izgrađena u okviru Oracle MySQL alata.
Dakle, ukratko, preduslovi za izradu ove web aplikacije su:

- Implementacija JSF-a
- Apache Tomcat Server
- Eclipse Indigo
- MySQL baza podataka

10.1. Model baze podataka

Prvi korak u izradi praktičnog dela rada bio je kreiranje baze podataka.
Baza se sastoji od 6 tabela:

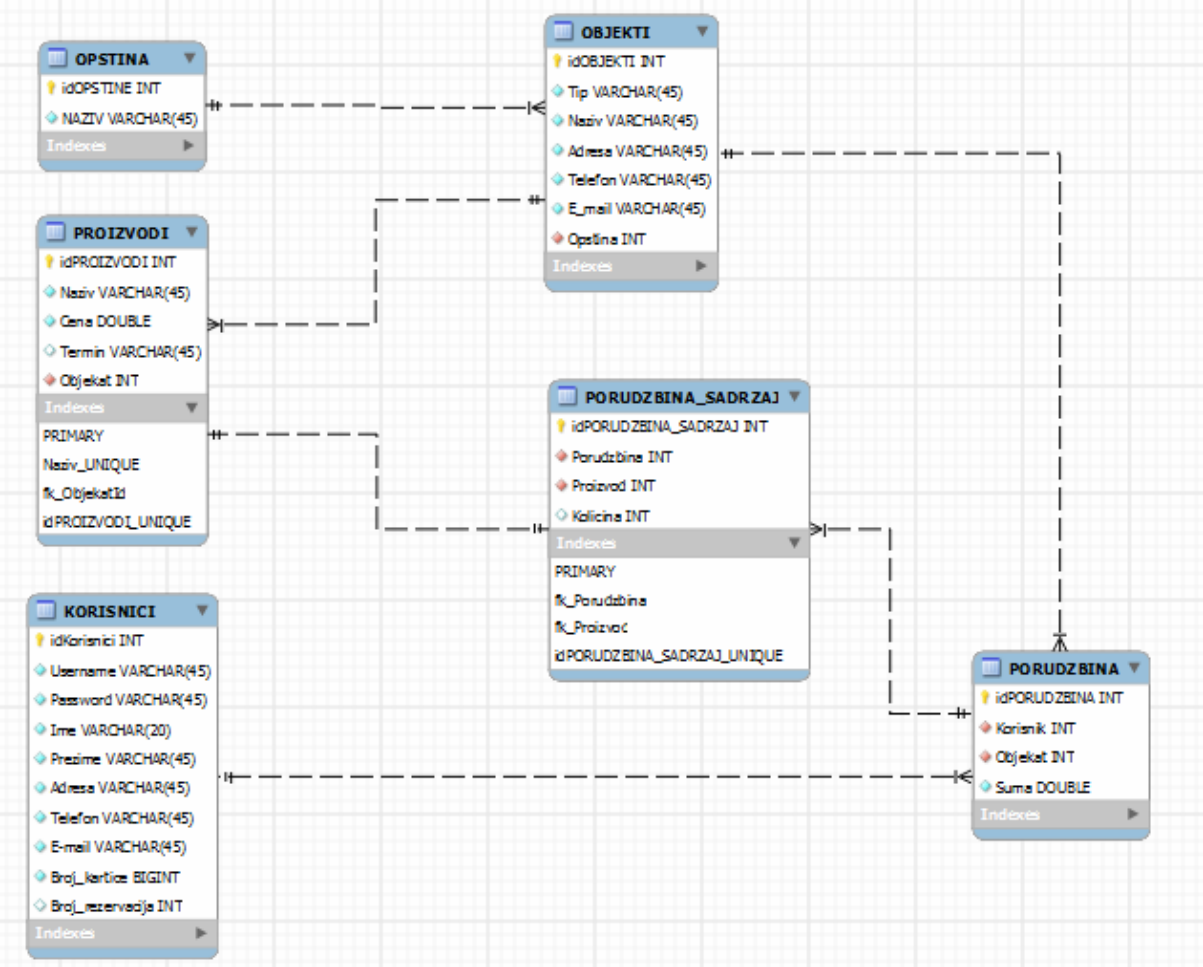
1. OPSTINA
2. OBJEKTI
3. PROIZVODI
4. KORISNICI
5. PORUDZBINA
6. PORUDZBINA_ SADRZAJ

Tabele OPSTINA, OBJEKTI, PROIZVODI su table iz kojih se podaci isključivo uzimaju i prikazuju na strani, dok u njih nije moguće kroz aplikaciju ubacivati podatke.
S druge strane, KORISNICI, PORUDZBINA i PORUDZBINA_SADRZAJ su tabele u koje se kroz aplikaciju podaci upisuju.

Tabele u bazi su povezane sprovođenjem sledećih pravila:

1. Na jednoj opštini se može naći više objekata.
2. Jedan objekat nudi više proizvoda.
3. Za jedan objekat, može se napraviti više porudžbina.
4. Jedan korisnik može napraviti više porudžbina.
5. Jedna porudžbina može sadržati više proizvoda (i to u proizvoljnoj količini).
6. Jedan proizvod predstavlja tačno jedan sadržaj porudžbine.

Detaljni grafički prikaz baze podataka predstavljen je sledećim dijagramom.



10.2. Struktura aplikacije

Nakon pripremljenog modela podataka, može se preći na izradu same aplikacije. To uključuje kreiranje .jsp stranica, kreiranje klasa za predstavljanje podataka iz baze, kreiranje *bean*-ova i podešavanje konfiguracionih fajlova.

U okviru ove aplikacije nalazi se šest .jsp strana. To su:

1. pocetna.jsp
2. ponuda.jsp
3. nalog.jsp
4. logovanje.jsp
5. registrovanje.jsp
6. masterStrana.jsp

Klase koje služe za predstavljanje podataka iz baze, nazvane su u skladu sa tabelama koje predstavljaju:

1. Opstina.java
2. Objekti.java
3. Proizvodi.java
4. Korisnici.java
5. Porudzbina.java
6. Porudzbina_sadrzaj.java

Pored ovih, tu je još jedna klasa korisna za prikaz porudžbine. To je PorudzbinaPrikaz.java.

Java klase proglašene *bean*-ovima su:

1. LogovanjeBean.java
2. RegistrovanjeBean.java
3. RegLogBean.java
4. PorudzbinaBean.java

Pored neophodnih konfiguracionih fajlova (web.xml i faces.config), u aplikaciji se koristi i jedan .css fajl (style.css), kao i nekoliko slika.

Ukratko, aplikacija je koncipirana na sledeći način.
Dolaskom na početnu stranu aplikacije (pocetna.jsp), korisnik dobija mogućnost da bira opštinu grada Beograda na kojoj želi da odabere prehrambeni objekat za poručivanje hrane. Klikom na kategoriju HRANA, prikazuju mu se objekti na izabranoj opštini. Dalje, klikom na željeni objekat, prikazuje se ponuda za isti (ponuda.jsp). Ukoliko želi da izvrši poručivanje, korisnik prvo mora da se uloguje (logovanje.jsp). Prethodno, ukoliko korisnik nema nalog, potrebno je da se registruje (registrovanje.jsp). Kada je korisnik ulogovan, osim mogućnosti poručivanja, može pogledati sopstveni nalog (nalog.jsp).
Stranica masterStrana.jsp predstavlja zaglavlje, odnosno obuhvata komponente koje treba da se nađu na više nego jednoj strani aplikacije.

Česta potreba prilikom razvoja web aplikacija je održavanje podataka dostupnim tokom više od jednog korisničkog zahteva. Ta se potreba i u ovoj aplikaciji ispunjava korišćenjem sesije. Postoje dve mogućnosti za realizaciju spomenutog. Jedna mogućnost je da se cela klasa proglasi *bean*-om sa *session* opsegom važenja. Tada objekti i promenljive (njihove vrednosti) definisani u okviru tog bean-a postaju dostupni tokom sesije korisnika.
Druga mogućnost je da *bean* klasa ima *request* opseg važenja, a da se po potrebi, određeni objekti smeštaju u sesiju.
U okviru ove aplikacije korišćene su obe mogućnosti, pa su *bean*-ovi, kojima je realizovan celokupan pozadinski kod, konfigurisani tako da RegistrovanjeBean.java i LogovanjeBean.java imaju *request*, a RegLogBean.java i PorudzbinaBean.java imaju *session* opseg važenja.
Svaki od pomenutih *bean*-ova namenjen je nekoj vrsti posla.

RegistrovanjeBean.java sadrži metod za registrovanje novog korisnika (public String registrujKorisnika())

LogovanjeBean.java sadrži:

- metod za logovanje (public String ulogujSe())
- metod za odjavljivanje (public String izloguj se())

PorudzbinaBean.java, odnosi se na mogućnost korisnika da, ukoliko je ulogovan, izvrši poručivanje. Sadrži:

- metod za prikaz opština (public List<SelectItem> getListOpstine())
- metod za prikaz objekata na izabranoj opštini (public void napuniObjete())
- metod za prikaz ponude izabranog objekta (public String prikaziPonudu())
- metod za prikaz trenutne porudžbine korisnika (public String prikaziModalPanel())
- metod za slanje porudžbine u bazu (public String posaljiPorudzbinuUBazu())
- metod za otkazivanje porudžbine (public String otkaziPorudzbinu())

RegLogBean.java se odnosi na mogućnost korisnika da, ukoliko je ulogovan, pogleda sopstveni nalog. Sadrži:

- metod za prikaz porudžbina (public void prikaziPorudzbineKratko())
- metod za prikaz detalja o izabranoj porudžbini (public void prikaziPorudzbine())
- metod za uklanjanje prikazanih porudžbina (public String skloniPorudzbine())

Kroz narednih nekoliko slika i objašnjenja biće izložene neke od funkcionalnosti aplikacije.

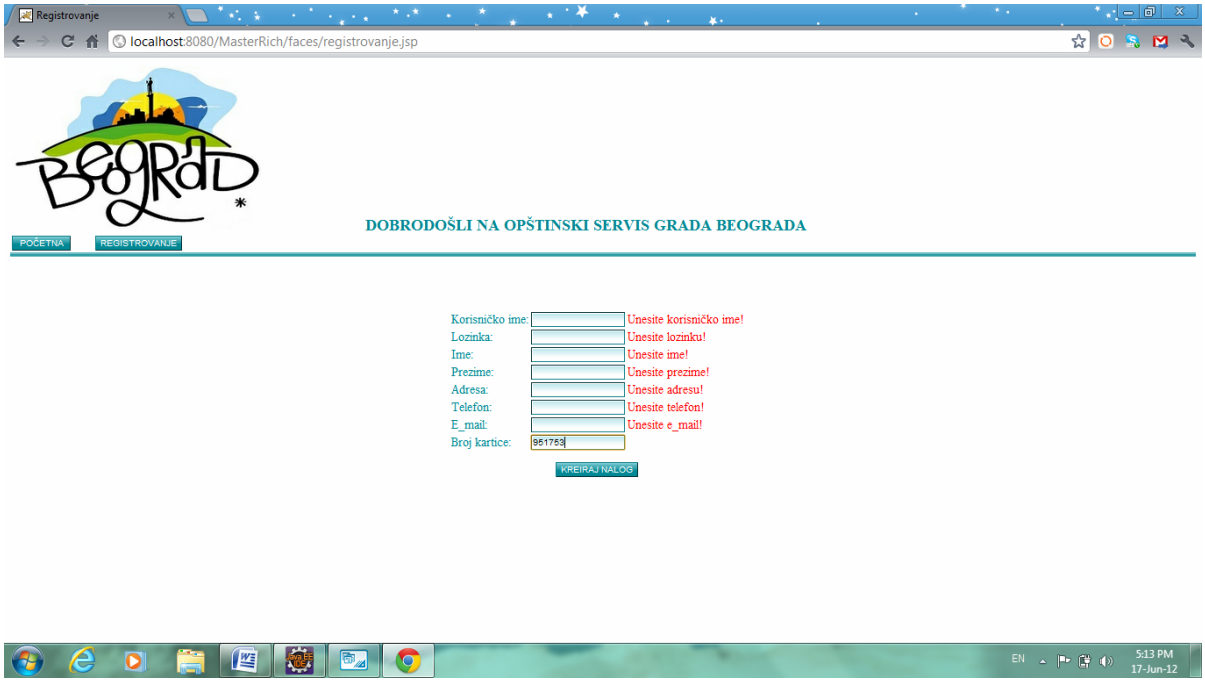
Na sledećoj slici može se videti kako izgleda početna strana aplikacije kada se odabere opština Vračar i kategorija HRANA.

Na početnoj strani aplikacije, korisnik, u okviru <h:selectOneManu/> komponente ima mogućnost izbora opštine, a zatim i kategorije. Klikom na <a4j:commandLink/>, koji se odnosi na kategoriju HRANA, a koji je predstavljen slikom, u okviru <rich:dataGrid/> komponente prikazuju se objekti za izabranu opštinu i kategoriju.

Navedene akcije bile su vezane za istu (početnu) stranu aplikacije. Sledeći korak je izbor željenog objekta i prelaz na sledeću stranu. Dakle, klikom na naziv objekta, korisnik prelazi na stranicu ponuda.jsp, na kojoj mu se, u okviru <rich:dataTable/> komponente, prikazuje ponuda baš za izabrani objekat. Prenošenje informacije o izabranom objektu vrši se korišćenjem <f:setPropertyActionListener/> komponente.

Na sledećoj slici se može videti stranica ponuda.jsp, ukoliko je korisnik izabrao FASTFOOD MOLE objekat.

Da bi uopšte imao mogućnost poručivanja, korisnik prethodno mora biti ulogovan, a da bi mogao da se uloguje, mora da se registruje.
Na sledećoj slici je stranica registrovanje.jsp, do koje se dolazi klikom na dugme zaglavlja, REGISTROVANJE.



Klikom na dugme KREIRAJ NALOG, podaci korisnika se šalju u bazu, pod uslovom da je uneto sve što je potrebno.

Sada, kada korisnik ima svoj nalog, može da se uloguje.

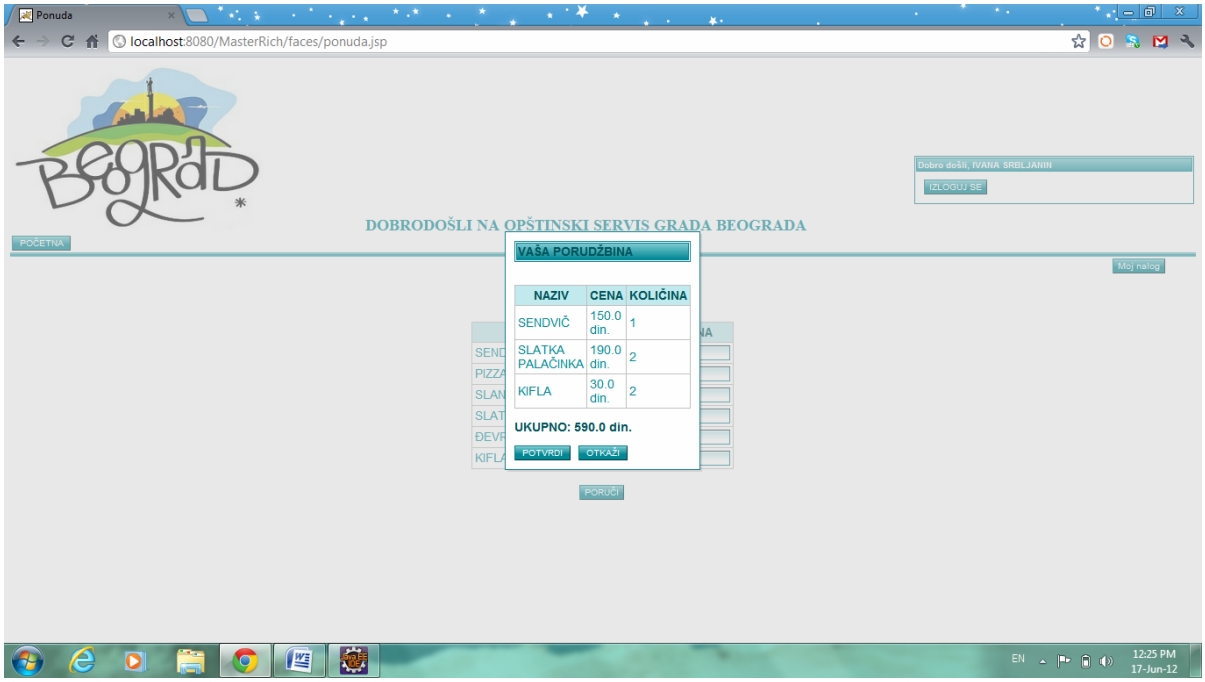
Logovanje.jsp je strana koja je uključena u stranice pocetna.jsp i ponuda.jsp korišćenjem <a4j:include/> komponente.

Prilikom logovanja, korisnik mora da vodi računa o ispravnosti korisničkog imena i lozinke, koje unosi, jer mu se u suprotnom izdaje poruka da pokuša ponovo.

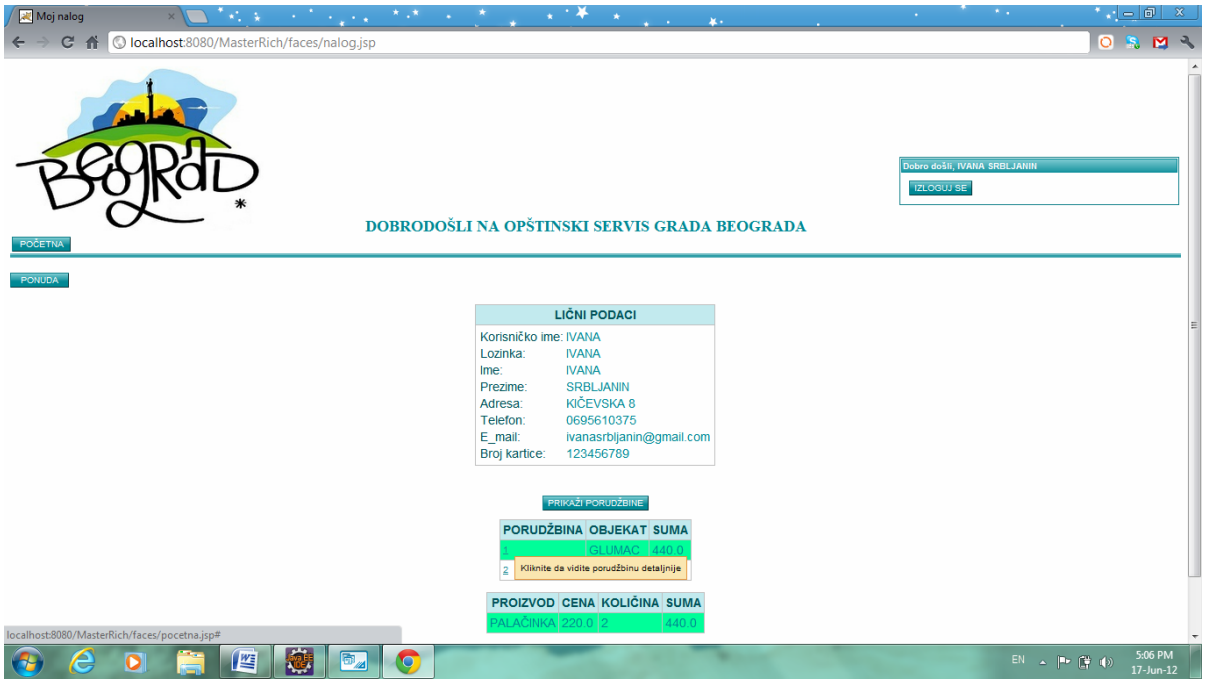
Kada se uloguje, korisnik stiče dve mogućnosti. Prvo, da izvrši poručivanje, a drugo da pogleda svoj nalog.

Na ranije prikazanoj slici ponuda.jsp, korisnik bira koji proizvod želi i u kojoj količini. Klikom na dugme PORUČI, prikazuje mu se <rich:modalPanel/> komponenta, sa trenutnom porudžbinom, kao i dugmićima POTVRDI (za slanje porudžbine u bazu) i OTKAŽI (za odustajanje od porudžbine).

Na sledećoj slici može se videti pomenuti modalPanel.



Na kraju, klikom na dugme MOJ NALOG, korisnik prelazi na novu stranicu, nalog.jsp, u okviru koje može videti svoje podatke, unete prilikom registrovanja, kao i sve dotadašnje ponude.



Ono što se tokom interakcije između stranica i korisnika veoma često koristi jeste komunikacija sa bazom. Zato će, za kraj, biti objašnjeno kako se to u okviru ove aplikacije realizuje.

JDBC API (Java Database Connectivity API) defineše interfejse i klase koje programeri koriste za konektovanje na bazu i izvršavanje upita.
JDBC drajver, koji je potrebno preuzeti sa interneta, u vidu .jar fajla, implementira ove interfejse i klase za određenog DBMS (Data Management System) proizvođača. Klasa koja definiše objekte kojima se Java aplikacija povezuje sa JDBC drajverom je JDBC DriverManager.
Da bi se na ovaj način uspostavila veza sa bazom podataka, potrebno je:

1. Definisati URL za bazu podataka.
   ```
   public String jdbcURL = "jdbc:mysql://localhost:3306/master_db";
   ```

2. Učitati odgovarajući drajver.
   ```
   Class.forName("com.mysql.jdbc.Driver");
   ```

3. Kreirati konekciju.
   ```
   conn = DriverManager.getConnection(jdbcURL, "root", "ivana");
   ```

4. Nakon povezivanja sa bazom, mogu se kreirati i izvršavati SQL upiti.
   ```
   Statement statement = conn.createStatement();
   ```

5. Skup rezultata upita smešta se u ResultSet.
   ```
   ResultSet rs = statement.executeQuery("select idKORISNICI, username, password, ime,
   prezime, adresa, telefon, e_mail, broj_kartice, broj_rezervacije from korisnici where
   username = '"+ getUsername().toUpperCase() + "' and password = '"+
   getPassword().toUpperCase()+"' ");
   ```

6. Po završetku rada sa bazom, potrebno je zatvoriti sve konekcije.
   ```
   statement.close();
   conn.close();
   ```

12. Zaključak

U ovom trenutku, kada su već izloženi i detaljnije opisani ključni koncepti Java Server Faces tehnologije, trebalo bi da je jasno čime je ova tehnologija okarakterisana. Ipak, osvrnimo se još jednom ukratko na osnovna svojstva JSF-a.

- JSF je standardno Java okruženje za jednostavniju i bržu izradu web aplikacija sa bogatim korisničkim interfejsom.
- Arhitektura JSF-a integriše MVC dizajnerski obrazac.
- JSF okruženje je izgrađeno nad bibliotekom klasa Servlet API.
- JSF je komponentno-zasnovano okruženje.
- JSF omogućava korišćenje *third-party* komponenti (komponenti treće strane).
- JSF predstavlja model programiranja vođen događajima.
- UI komponente su uskladištene na serveru i na serveru se obrađuju događaji kreirani od strane korisnika.
- JSF obezbeđuje rukovanje navigacijom.
- Odlikuje se proširivom arhitekturom. Proširiva arhitektura se odnosi na mogućnost dodavanja funkcionalnosti na osnove JSF-a.
- JSF ima obimnu podršku alata različitih proizvođača (Sun, Oracle, IBM).

Teorijski, JSF tehnologija se može koristiti za izradu aplikacija koje nisu namenjene webu, iako se u najvećem broju slučajeva, ipak, koristi baš za izradu web aplikacija.

Veoma pogodno svojstvo JSF okruženja jeste što je namenjeno svima onima koji su na neki način uključeni u razvoj softvera, a ne samo ekspertima kodiranja.
Web dizajneri mogu da dizajniraju izgled stranica web aplikacije koristeći biblioteke JSF-a.
Programeri aplikacija koriste JSF za programiranje objekata, konvertera, validatora, rukovanje događajima itd.
Zahvaljujući proširivoj i prilagodljivoj prirodi JSF-a, programeri komponenti mogu praviti komponente neohodne za specifične potrebe aplikacije.
Projektanti aplikacija, koji su odgovorni za kompletnu arhitekturu aplikacije definišu navigaciju među stranicama, obezbeđuju skalabilnost aplikacije, konfigurišu *beans* objekte.
Takođe, JSF je tehnologija pogodna za proizvođače (*vendore*) alata.

Literatura:

[1] Kito D. Mann: Java Server Faces in Action, Manning, 2005
[2] Bill Dudney, Jonathan Lehr, Bill Willis, LeRoy Mattingly: Mastering Java Server Faces, Wiley Publishing, 2004
[3] www.ibm.com/developerworks
[4] www.netbeans.org
[5] www.javaworld.com
[6] www.cet.rs
[7] www.vogella.de
[8] www.roseindia.net
[9] docs.oracle.com
[10] sr.wikipedia.org
[11] en.wikipedia.org