UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

MASTER RAD

PRIMENA KOMPONENTNOG PROGRAMIRANJA ZASNOVANOG NA ENTERPRISE JAVA ZRNIMA U EDUKATIVNE SVRHE

Mentor:
Prof. Dr. Dušan Tošić

Student:
Elazar Filip

Beograd , Jun 2011

SADRŽAJ

1. Uvod

Cilj ovog rada je da opiše kako se Enterprise Java zrna mogu primeniti u edukativne svrhe, odnosno kada i kako se ona mogu koristiti u razvoju edukativnih Java EE aplikacija. Podrazumeva se da je čitalac upoznat sa objektno-orijentisanim programiranjem i programskim jezikom Java, osnovama Veb tehnologija kao što su HTML i CSS, kao i sa osnovama relacionih baza podataka. Rad je koncipiran tako da su prvo predstavljeni osnovni pojmovi vezani za razvoj Java EE aplikacije, zatim sledi kraći teoretski prikaz Java EE zrna i na kraju je data ilustracija primene zrna u izradi edukativne aplikacije na konkretnom primeru.

1.1 Šta je Java Enterprise Edition

Java Enterprise Edition (Java EE) je jedna od Java platformi koja služi za razvoj i izvršavanje višeslojnih, skalabilnih mrežnih aplikacija. Kao i svaka Java platforma, ona sadrži Java virtuelnu mašinu i programski interfejs, pružajući okruženje u kojem mogu da se razvijaju i izvršavaju programi napisani na programskom jeziku Java.

1.2 Šta je Enterprise Java zrno

Enterprise Java zrno je višekratna serverska komponenta alikacije na Java Enterprise Edition (Java EE) platformi. Ono sadrži poslovnu logiku aplikacije, tj. kod čijim se izvršavanjem obezbeđuje funkcionalnost aplikacije. Enterprise Java zrno nije isto što i Java zrno, osnovna razlika je u tome što se Enterprise Java zrno izvršava u kontejneru za Enterprise Java zrna, koji je opisan u poglavlju 1.5. U daljem tekstu podrazumevaće se da je reč o Enterprise Java zrnima, koja su glavna tema ovog rada i koja ćemo skraćeno zvati zrna. U poglavljima 1.4 i 2.2 detaljnije su opisani pojmovi višeslojne aplikacije, skalabilnosti, raznovrsnosti korisnika aplikacije i transakcije, pa ćemo za sada reći samo da zrna obezbeđuju brži i jednostavniji razvoj višeslojnih, skalabilnih Java Veb aplikacija sa potrebom korišćenja transakcija i raznovrsnim korisnicima. Implementiraju se u obliku Java klasa, moraju imati podrazumevani konstruktor i podržavaju serijalizaciju. Instanca ili objekat klase Java zrna je konkretan

primerak klase sa sopstvenim identitetom i stanjem, tj. vrednostima atributa. Po konvenciji imenovanja metoda za pristupanje atributu klase, koriste se standardni nazivi metoda koji počinju sa *get* i *set* a završavaju se nazivom atributa koji počinje velikim slovom.

Tako na primer, u aplikaciji za kupovinu na Internetu, poslovna logika za poručivanje artikla moze biti implementirana u okviru zrna u metodi *poručiProizvod* čijim pozivanjem se proizvod poručuje. Slično, u drugoj metodi *troškoviPošiljke* može biti implementirana poslovna logika računanja troškova pošiljke na osnovu tipa i cene artikla, njegovih dimenzija, težine i udaljenosti poručioca. Pozivanjem ove metode korisniku se mogu prikazati troškovi pošiljke.

1.3 Pregled osnovnih pojmova

Java Veb Servleti

Java Veb Servleti su klase napisane u programskom jeziku Java. Ono što im je zajedničko ja da implementiraju *javax.servlet.Servlet* interfejs. Servleti mogu komunicirati sa klijentima korišćenjem protokola kao što su naprimer HTTP, FTP,TCP/IP, ali kako je za nas od posebnog značaja HTTP protokol, to će nam posebno interesantan biti *HttpServlet*, tj. abstraktna klasa *javax.servlet.http.HttpServlet.*

JavaServer Pages

JavaServer Pages (JSP) su dinamičke Veb strane zasnovane na JSP skript jeziku. Sastoje se od statičkog dela obično predstavljenjog HTML-om ili XML-om i JSP elemenata koji služe za prikazivanje dinamičkih sadržaja. Izvršavaju se na serverskoj strani i nisu zavisne od platforme tj. mogu da se izvšavaju na različitim tipovima servera. Po raspoređivanju JSP strana na server, vrši se njihovo prevođenje u Servlet klase koje se zatim prevode u bajtkod (*bytecode*) i izvršavaju isto kao i ostale Servlet klase.

JSP standardna biblioteka etiketa

JSP standardna biblioteka etiketa (*JSTL*) enkapsulira bazične funkcionalnosti zajedničke za mnoge Veb aplikacije u vidu etiketa. Ona omogućava optimizaciju implementacije koda stavljajući na raspolaganje širok spektar etiketa kao sto su npr. etikete za iteraciju ili proveru ispunjenosti uslova.

JSP etiketa ima svoj početak i završetak koji se označavaju navođenjem imena etikete u uglastim zagradama, i može ali ne mora imati telo. Na primer,

```
<c:if test=“neki test“>
    Telo etikete
</c:if>
```

je jedna etiketa koja ima telo, dok je

```
<ime_etikete:nešto/>
```

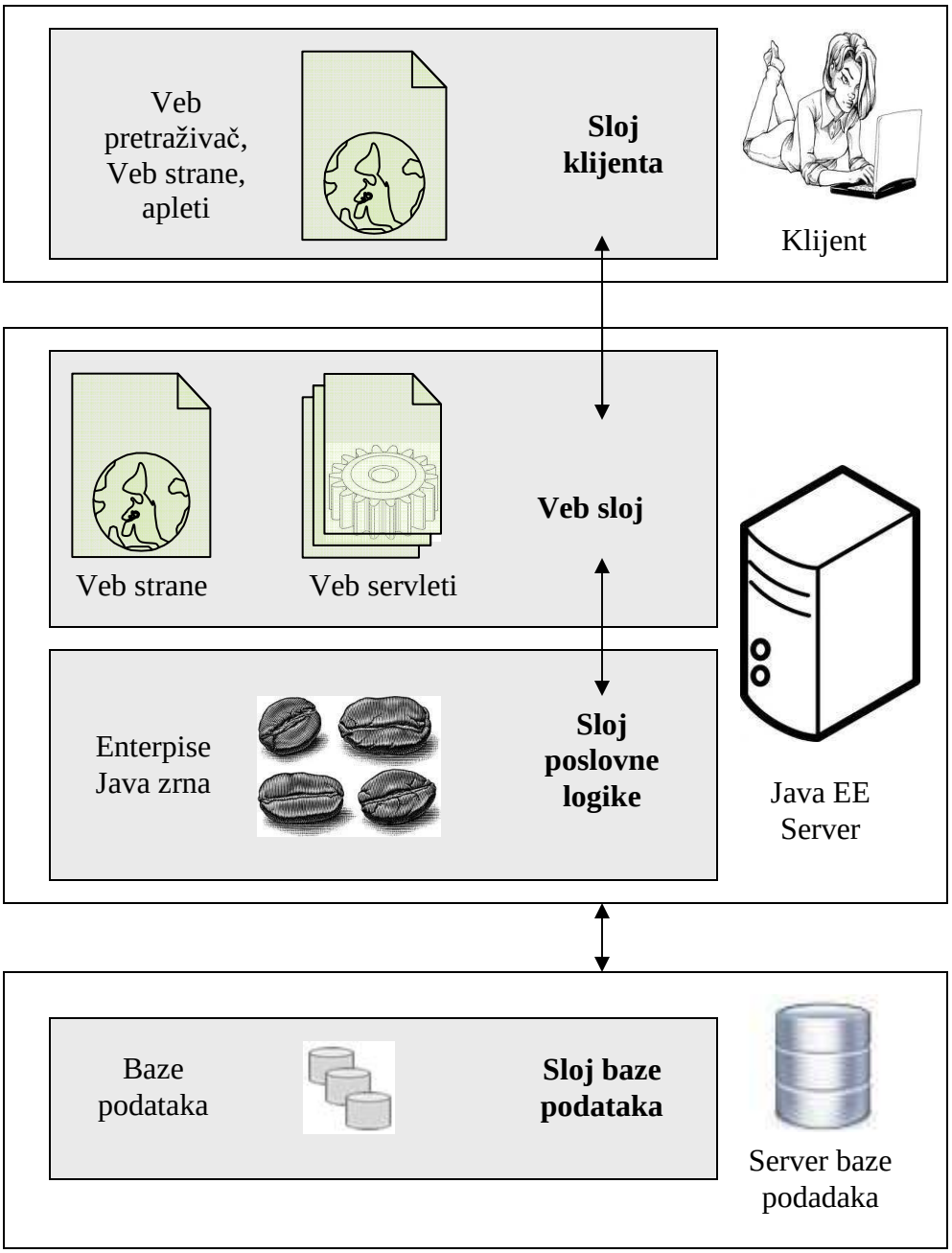
primer etikete bez tela.

Da bismo koristili određenu biblioteku etiketa potrebno je prethodno navesti gde se ta biblioteka nalazi. U JSP datoteci u kojoj želimo da koristimo etikete određene biblioteke etiketa potrebno je na početku datoteke napraviti referencu na odgovarajuću biblioteku etiketa. U ovom slučaju za c:if etiketu, to ćemo uraditi postavljanjem direktive:

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

na početak JSP datoteke u kojoj se etiketa koristi.

1.4 Višeslojna arhitektura Java EE aplikacije



slika 1.1 **Slojevi Java EE aplikacije**

I ako Java EE aplikacije mogu sadržati i četiri sloja, kao što se to vidi i na prikazanoj slici, smatra se da one generalno pripadaju troslojnim aplikacijama iz razloga njihove rasprostranjenosti na tri različite mašine, a to su mašina klijenta, Java EE server i server baze.

Na Java EE platformi obično se koristi distribuirani višeslojni model aplikacija. Aplikaciju možemo razložiti na više komponenata, vršeći podelu prema funkciji komponente i instalirati ih na odgovarajuće mašine u zavisnosti od toga kojem sloju aplikacije data komponenta pripada. Veb servleti i JSP strane su Veb komponente dok se statičke Veb strane, slike i apleti ne svrstavaju u Veb komponente. Enterprise Java zrna su komponente sloja poslovne logike. U Java EE arhitekturi, Veb komponente i statički Veb sadržaji se zajedno nazivaju Veb resursi. Veb modul je najmanja jedinica Veb resursa koja se može rasporediti i koristiti na Veb serveru. Veb modul se može distribuirati spakovan u JAR datoteku koja se tada naziva Veb arhiva (WAR). Zbog razlike u sadržaju WAR i JAR datoteka, WAR datoteke imaju ekstenziju *war*. Ovako opisan Veb modul potpuno je prenosiv.

Da bi se WAR datoteka koristila na serveru, ona mora da sadrži deskriptor raspoređivanja (*deployment descriptor*), koji je u osnovi XML datoteka koja, između ostalog sadrži podatke o sadržaju osnovnog direktorijuma Veb aplikacije i mapiranje imena resursa aplikacije sa imenima resursa servera.

Na slici 1.1 prikazana je podela višeslojne Java EE aplikacije po slojevima a to su:

- klijent-sloj na kojem se nalaze komponente koje se izvršavaju na klijentskoj mašini;
- Veb-sloj sa komponentama koje se nalaze na Java EE serveru;
- sloj poslovne logike koji se nalazi na Java EE serveru;
- sloj baze na kojem se nalaze softver i podaci smešteni na serveru baze;

1.5 Java EE kontejneri

Za svaku od komponenata višeslojne Java EE aplikacije, Java EE server obezbeđuje odgovarajući kontejner. Kontejneri predstavljaju interfejs između komponenata i funkcionalnosti Java platforme neophodnih za funkcionisanje datih komponenti. Da bi određena komponenta bila spremna za izvršavanje, prethodno mora biti raspoređena u odgovarajući kontejner shodno sloju aplikacije u kojem se nalazi.

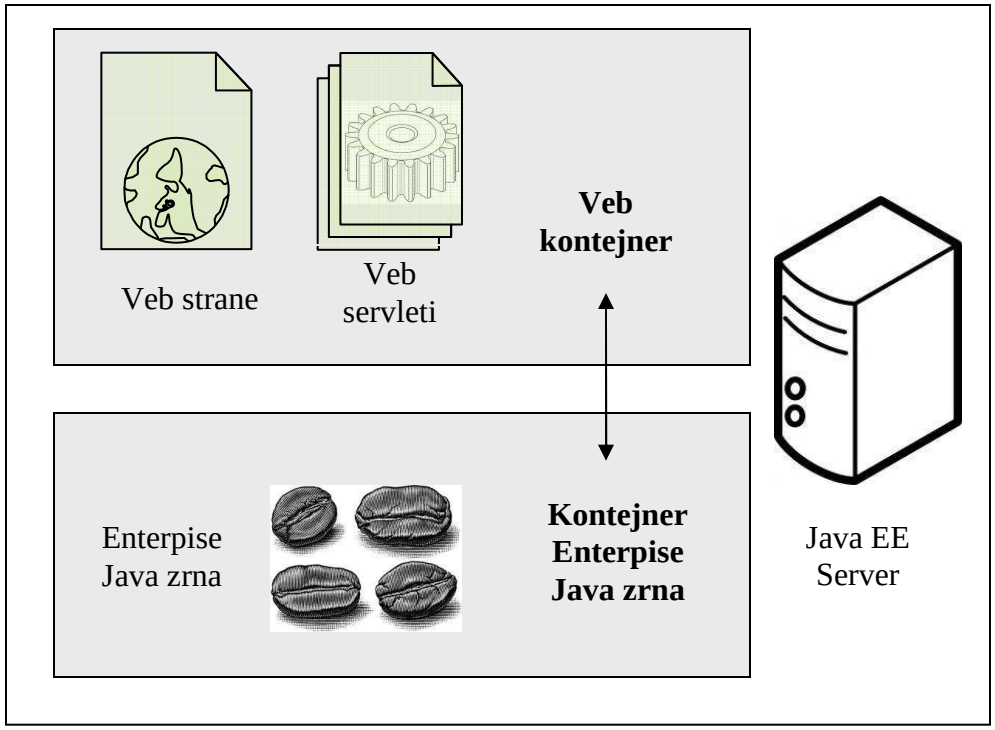
Na Java EE serveru imamo dva osnovna tipa kotejnera i to su:

- kontejner Enterprise Java zrna (EJB kontjener)
- Veb kontejner

Kontejner Enterpise Java zrna je zadužen za način izvršavanja Enterprise Java zrna a Veb kontejer je zadužen za način izvršavanja Veb strana i servleta.

U zavisnosti od konfiguracije kontejnera, zavisi ponašanje komponete koja je u njega raspoređena. Drugim rečima, komponente iste aplikacije se mogu ponašati različito u zavisnosti od toga kako su konfigurisani kontejneri u koje su respoređene. Tako, na primer, u jednom okruženju Enterprise Java zrna mogu imati određena prava pristupa podacima baze, dok u drugom okruženju ta prava pristupa mogu biti potpuno drugačija.

Kontejneri Java EE servera prikazani su na slici 1.2.



Slika 1.2 **Kontejneri Java EE servera**

Enterprise Java zrna se izvršavaju u kontejneru za Enterprise Java zrna te on predstavlja njihovo izvršno okruženje na Java EE serveru. Kontejner pruža zrnima funkcionalnosti platforme kao što su bezbednost i obavljanje transakcija.

Java EE bezbednosni model omogućuje konfiguraciju Veb komponenti, a Enterprise Java zrna omogućavaju pristup sistemskim resursima jedino korisnicima sa dozvoljenim pravom pristupa.

Java EE model transakcija omogućuje specifikaciju veza između metoda koje čine transakciju tako da su sve metode jedne transakcije tretirane kao jedna celina. Transakcija predstavlja jedno izvršenje logičke celine posla sastavljenog od više metoda koje je potrebno izvršiti u kompletu. Transakcije čuvaju konzistentnost podataka u slučaju konkurentnog pristupa resursima. One ili izvrše predviđene radnje, menjajući određene podatke tj. stanja, ili ako to nije moguće u datom trenutku, vraćaju sve u prvobitno stanje pre poziva transakcije. Na primer, ako je potrebno da se sa bankovnog računa *A* prebaci novac na bankovni račun *B*, trebalo bi da se skidanje novca sa računa A i uplata novca na račun B obave kao jedna logička celina tj. transakcija. Ne bi bilo dobro da novac bude skinut sa računa A, a iz bilo kog razloga ne bude uplaćen na račun B, pa bi iz tog razloga bilo bi poželjno koristiti transakcije.

2. Enterprise Java zrna

2.1 Koje su koristi od korišćenja Enterprise Java zrna

Enterprise Java zrna pojednostavljuju razvoj većih, distribuisanih aplikacija iz više razloga:

- Prvo, kontejner Enterprise Java zrna pruža sistemske funkcionalnosti kontrolišući instance zrna, transakcije, bezbednosne autorizacije, niti itd. pa osoba koja razvija softver ne mora mnogo da brine o pomenutim funkcionalnostima već može da se posveti rešavanju problema same poslovne logike. Inače, niti predstavljaju tok programa. Svaki java program ima barem jednu nit koja se kreira prilikom pokretanja aplikacije, ali za izvršavanje relativno nezavisnih delova programa može se koristiti više niti.
- Drugo, pošto zrna sadrže poslovnu logiku aplikacije, ona je jasno razdvojena od prezentacije. To dovodi do takozvanog *laganog klijenta* što je naročito bitno kod manjih uređaja kod kojih bi svaka rutina poslovne logike ili pristup bazi iz sloja klijenta opterećavala uređaj. Takođe, razgraničavanjem komponenti na ovaj način, Veb dizajner ne mora da zna na koji način se implementira određena poslovna logika, već može da se skoncentriše na svoj deo posla.
- Treće, ali ne i najmanje važno, pošto su Enterpirse Java zrna prenosiva, moguće je već postojeća zrna koristiti u razvoju novih aplikacija i na drugim Java EE serverima. Zrna se pakuju u EJB JAR datoteke koje predstavljaju nezavisne komponente. Dakle, zrna se mogu posmatrati kao komponente građe, koja su, kada su jednom nepravljena, na raspolaganju u novim projektima. Pri pravljenju nove aplikacije mogu se koristiti već gotovi moduli, kao što su EJB JAR datoteke, koje se postavljaju u EAR datoteku koja sadrži aplikaciju.

2.2 Kada koristiti Enterprise Java zrna?

Ako aplikacija koju razvijamo zadovoljava barem jedan od sledećih kriterijuma, onda treba razmisliti o upotrebi Enterprise Java zrna:

- aplikacija treba da bude skalabilna;

Skalabilna aplikacija ima karakteristiku da prilikom povećanja radnog opterećenja mašine ili mašina na kojoj se aplikacija izvršava, na primer zbog povećanja broja korisnika aplikacije, nije potrebno praviti veće izmene same aplikacije već se problem može rešiti povećanjem resursa mašine, odnosno mašina na kojima se aplikacija izvršava. Da bi omogućili rastući broj korisnika aplikacije, moguće je da će biti potrebno izvršiti distribuciju njenih komponenti na više mašina. Ne samo da Enterprise Java zrna

aplikacije mogu da se izvršavaju na različitim mašinama, već će njihova lokacija ostati transparentna odgovarajućem kontejneru.

- transakcije moraju osigurati integritet podataka;

Kontejner Eterprise Java zrna brine se o transakcijama, mehanizmu koji upravlja konkurentnom pristupu zajedničkim objektima. Primer transakcije naveden je u poglavlju 1.5 Java EE kontejneri.

- aplikacija može da ima raznovrsne korisnike;

Ovde se pre svega misli na takozvane *lagane klijente*, tj manje uređaje preko kojih se može pozivati aplikacija. Oni mogu biti raznovrsni i brojni. Na primer, mobilini telefon.

2.3 Vrste Enterprise Java zrna

Enterprise java zrna mogu biti:

1. Zrna sesije (*Session bean*)
2. Zrna vođena porukom (*Message-driven bean*)

1. Zrna sesije izvršavaju zadatake koje im klijent zada, tačnije, klijent poziva metode zrna čijim izvršavanjem zrno pruža traženu uslugu klijentu. Zrna sesije takođe mogu biti implementirana kao Veb servisi. Veb servisi su programske komponente koje komuniciraju sa klijentima koristeći otvorene XML standarde i Internet protokole, kao što je HTTP protokol.

2. Zrna vođena porukom služe za osluškivanje i reagovanje na određeni tip poruka.

Zrna sesije

Zrna sesije sadrže poslovnu logiku aplikacije i po potrebi mogu biti pozvana bilo lokalno, bilo sa neke druge mašine na serverskoj strani ili bilo putem Veb servisa. Zrna sesije se koriste pozivanjem metoda koje zrno sadrži. Po pozivanju odgovarajuće metode zrna izvšavaju se pridružene operacije i onom ko je pozvao zrno vraća se rezultat metode. Pozivač zrna ne zna na koji način se pozvana metoda izvršava jer poslovna logika ostaje spakovana u okviru samog zrna.

Tipovi zrna sesije

Postoje tri različita tipa zrna sesije i to su:

1. Zrna stanja (*Stateful beans*)
2. Zrna bez stanja (*Stateless beans*)
3. Zrna Jedinci (*Singleton beans*)

Zrna stanja

Kod zrna stanja, kao što i samo ime sugeriše, zrno predstavlja stanje klijentove sesije. Sesija je jedinstvena, vremenski ograničena veza između identifikovanog korisnika i Veb servera, pomoću koje se obavlja komunikacija sa korisnikom od trenutka njegovog pristupa Veb aplikaciji do trenutka isteka vremenskog ograničenja ili odjave korisnika. Stanje zrna pedstavljeno je vrednostima atirbuta konkretne instance. Ovo stanje još se naziva i stanje konverzacije, pošto je korisnik u stalnoj komunikaciji sa svojom instancom zrna.

Jednoj instanci zrna stanja pridružuje se samo jedan korisnik, tj. ono ne moze biti deljeno između više korisnika istovremeno. Kada korisnik okonča sesiju, ili kada ona istekne, njemu pridružena instanca zrna stanja se uništava i svi podaci o trenutnom stanju se gube.

Ovo ne predstavlja problem pošto obično nema potrebe da se podaci o stanju sesije čuvaju po njenom okončanju.

Zrna bez stanja

Zrna bez stanja nisu u stalnoj komunikaciji sa korisnikom, već samo onda kada korisnik pozove neku od metoda zrna. Instance zrna bez stanja nalaze se u kontejneru Enterprise Java zrna gde čekaju da budu uposlena. Po korisnikovom pozivu određene metode zrna, kontejner dodeljuje slobodnu instancu zrna korisniku. Ta instanca će sadžati podatke vezane za konkretnog korisnika dok traje izvršavanje metode koju je korisnik pozvao i već pri sledećem pozivu iste instance od strane novog korisnika ti podaci će biti zamenjeni podacima novog korisnika. Po izvršavanju metoda zrno više nije u vezi sa korisnikom i spremno je da bude uposleno ponovo, kao što su to i ostale slobodne instance zrna u kontejneru.

Zrno bez stanja može da implementira Veb servis dok zrno stanja to ne može.

S obzirom da instance zrna bez stanja mogu biti ponovo uposlena kad su slobodna, u aplikacijama sa većim brojem korisnika ona imaju bolje performase od zrna stanja. Obično je potreban manji broj zrna bez stanja nego zrna stanja za pružanje podrške istom broju korisnika.

Zrna jedinci

Za razliku od zrna stanja i zrna bez stanja, kada je u pitanju zrno jedinac, pravi se samo jedna instanca zrna za životni ciklus aplikacije. Ovoj instanci mogu konkurentno pristupati različiti korisnici, a instanca zadržava trenutno stanje između različitih poziva, osim ako ne dođe do pada ili spuštanja servera.

Kao i zrna bez stanja, zrna jedinci mogu implementirati Veb sevis.

Moguće je izvesti da se instanca zrna jedinca kreira pri pokretanju aplikacije i u tom slučaju ona može obaviti potrebne zadatke inicijalizacije prilikom pokretanja aplikacije. Takođe instanca može obaviti poslove čišćenja pre gašenja aplikacije pošto postoji tokom čitavog životnog ciklusa aplikacije.

Pristup zrnima sesije

Zrna sesije mogu, ali ne moraju, imati takozvane *poslovne interfejse* preko kojih korisnik može pozivati metode zrna. Korisnici koji ne koriste interfejse za pozivanje metoda zrna, mogu to uraditi direktno pozivanjem neke javne metode (*public method)* metode zrna. Poslovni interfejs zrna je standardni Java interfejs koji sadrži metode zrna sesije. Dobro dizajnirani interfejs i javne metode zrna štite korisnika od složenosti same poslovne logike koju metode zrna odrađuju. Način na koji se poslovna logika u okviru zrna obavlja, trebalo bi da može da se menja bez uticaja na korisnika. Ako pak dođe do promene u deklaraciji javne metoda zrna i potpisa metoda u interfejsu, onda je potrebno napraviti promene i u kodu koji poziva odgovarajuće metode.

Interfejs zrna sesije može biti

- lokalni (*local*)
- udaljeni (*remote*)

Jedno zrno sesije može imati više interfejsa. Koji tip interfejsa ćemo odabrati, kao pogodan, zavisi od više faktora, kao što je npr. distribucija komponenata. Ako će se komponente, koje pozivaju zrno, nalaziti na različitim mašinama, onda je potreban udaljeni interfejs, što daje veću fleksibilnost ali manju brzinu poziva. Sa druge strane, neka zrna su tesno povezana i često komuniciraju pa bi se lokalnim pozivima postigle bolje performanse.

Korisnike zrna sesije takođe mozemo podeliti na lokalne i udaljene.

Lokalni korisnik ima sledeće karakteristike:

- mora se nalaziti u istoj aplikaciji kao i zrna sesije kojima pristupa;
- može biti Veb komponenta(servlet, dinamička Veb strana) ili drugo zrno;
- lokacija zrna sesije čije metode koristi nije mu transparentna;

Java anotacije koriste se za označavanje vrste zrna kao i za tip interfejsa odgovarajućeg zrna.
Prilikom implementacije, klase zrna obeležavaju se anotacijom koja označava tip zrna, a u datoteci u kojoj se zrno implementira potrebno je uključiti odgovarajuću klasu koja tu vrstu zrna opisuje. U tabeli ispod, možete pogledati prikaz zrna, odgovarajućih anotacija i potrebnog uključivanja u datoteku zrna.

vrsta zrna	**Java anotacija**	**uključivanje odgovarajuće klase**
zrno bez stanja	*@Stateless*	i*mpo*rt *javax.ejb.Stateless*
zrno stanja	*@Stateful*	i*mpo*rt *javax.ejb.Stateful*
zrno jedinac	*@Singleton*	i*mpo*rt *javax.ejb.Singleton*
zrno vođeno porukom	*@MessageDriven*	*i*mport *javax.ejb.MessageDriven*

Slično, u tabeli ispod dat je prikaz interfejsa, odgovarajućih anotacija i potrebnog uključivanja u datoteku interfejsa.

tip interfejsa	**Java anotacija**	**Uključivanje odgovarajućeg interfejsa**
lokalni interfejs	*@Local*	*Import javax.ejb.local*
udaljeni interfejs	*@Remote*	*Import.javax.ejb.remote*

Za više detalja o Java notacijama pogledati literaturu [1].

Tako, na primer, za zrno bez stanja NekoZrno

```
@Stateless
public class NekoZrno implements LokalniInterfejs{...}
```

interfejs će biti

```
@Local
public interface LokalniInterfejs{...}
```

Da bi korisnik mogao da koristi instancu zrna sesije ili njegovog interfejsa, prvo mora da napravi referencu na tu instancu. To se može uraditi ili preko ubrizgavanja zavisnosti (*dependence injection*) ili koristeći *Java Naming and Directory Interface*.
Java Naming and Directory Interface (JNDI) je deo Java platforme koji aplikacijama zasnovanim na Java tehnologiji obezbeđuje jedinstven interfejs za višestruko imenovanje i servise za rad sa direktorijumima. JNDI API pruža aplikacijama mogućnost pretrage objekta koristeći atribute objekta i njegovo preuzimanje i skladištenje.
Za više detalja o JNDI API pogledati literaturu [1].
Ubrizgavanje zavisnosti je funkcionalnost Java EE platforme koja omogućava olakšano korišćenje Enterprise Java zrna. Mogućnost bezbednog ubrizgavanja zavisnosti i izbora implementacije interfejsa u fazi raspoređivanja aplikacije na server, naručito je značajna zbog zbližavanja Veb sloja aplikacije sa funkcionalnostima koje pruža kontejner Enterprise Java zrna, kao što su kontrola transakcija. Za više detalja o ubrizgavanju zavisnosti pogledati literaturu [1].

Jednostavnije je koristiti ubrizgavanje zavisnosti.

Da bismo napravili referencu ubrizgavanjem zavisnosti, koristimo anotaciju

```
@EJB
```

Za primer *NekoZrno*, na mestu gde želimo da koristimo metode lokalnog interfejsa, prethodno ćemo ubaciti sledeću anotaciju

```
@EJB
LokalniInterfejs lokalniInterfejs;
```

Instanca lokalniInterfejs sada je spremna za poziv metode interfejsa.

Slično, ako zrno ima javne metode, moze im se pristupiti direktno ubacivanjem sledeće anotacije

```
@EJB
NekoZrno nekoZrno;
```

Instanca *NekoZrno* sada je spremna za poziv javne metode.

Udaljeni korisnik ima sledeće karakteristike:

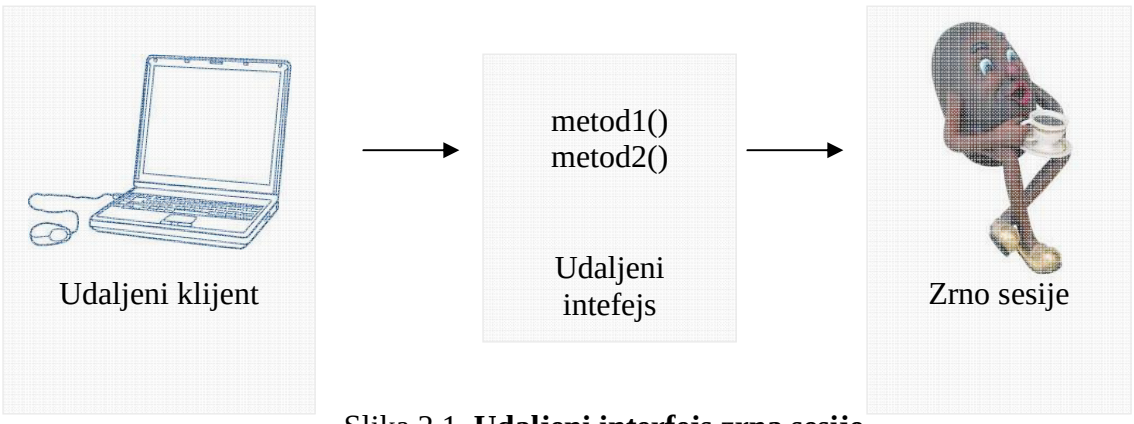
- može se nalaziti na drugoj mašini ili drugoj Java virtuelnoj mašini;
- može biti Veb komponenta, klijent-aplikacija(*application client*), ili drugo zrno;
- zrno kojem pristupa treba da ima udaljeni interfejs;

Udaljeni interfejsi obeležavaju se anotacijom

```
@Remote
```

Udaljeni interfejsi referenciraju se pomoću ubirzgavanja zavisnosti na isti način kao i lokalni interfejsi

```
@EJB
UdaljeniInterfejs udaljeniInterfejs;
```



Slika 2.1 **Udaljeni interfejs zrna sesije**

Aplikacije koje se nalaze van Java EE server okruženja moraju koristiti JNDI za referenciranje udaljenog interfejsa. Na primer,

```
UdaljeniInterfejs udaljeniInterfejs = (UdaljeniInterfejs)
    InitalContext.lookup("java:global/ime_aplikacije/UdaljeniInterfejs");
```

Zrna vođena porukom

Zrna vođena porukom omogućavaju Java EE aplikacijama da obrađuju poruke asihrono. Kod sinhronih operacija dolazi do blokiranja procesa sve dok se pozvana operacija ne završi, dok kod asihronih operacija po pozivanju nema blokiranja, nego se proces nastavlja. Zrna vođena porukom obično funkcionišu kao osluškivač *Java Message Service (JMS)* poruka. *Java Message Service API* je standard za slanje poruka koji pruža mogućnost komponentama Java EE aplikacija da kreiraju, šalju i primaju poruke, omogućavajući komunikaciju koja je pouzdana i asinhrona. Za opis osluškivača događaja pogledati literaturu [8], Poglavlje 2.6 - *Event Handling.* Dakle, zrna vođena porukom osluškuju i primaju JMS poruke. Poruka može biti poslata od strane bilo koje Java EE komponente, npr. od strane durgog Enterpise zrna. Po primanju JMS poruke, zrna vođena porukom preduzimaju određene akcije obično obrađujući primljenu poruku.
Zrna vođena porukom imaju sledeće karakteristike:

- izvršavaju se po prijemu poruke;
- pozivaju se asihrono;
- ne predstavljaju podatke u bazi, ali se mogu koristiti za menjanje podataka baze;
- nemaju stanje, tj. ne pamte podatke o klijentu;

Komponente koje pozivaju zrno vođeno porukom ga ne lociraju, vec koristeći JMS šalju poruku na lokaciju za koju zrno vođenom porukom ima osluškivač. Takva lokacija se može izabrati na serveru za svako zrno vođeno porukom. Kada poruka stigne, kontejner poziva metodu zrna *onMessage* da obradi poruku. Poruka se dalje može obraditi u zrnu u skladu sa poslovnom logikom, može se pozvati neko zrno sesije da obradi podatke iz poruke ili da ih sačuva u bazu.

Možemo uočiti nekoliko sličnosti između zrna vođenih porukom i zrna bez stanja:

- instanca zrna vođenog porukom ne pamti stanje klijenta, tj. podatke o klijentu;
- Sve instance zrna vođenih porukom u kontejneru su ekvivalentne;
- Jedna instanca zrna vođenih porukom može opsluživati više klijenata, jednog po jednog, naravno;

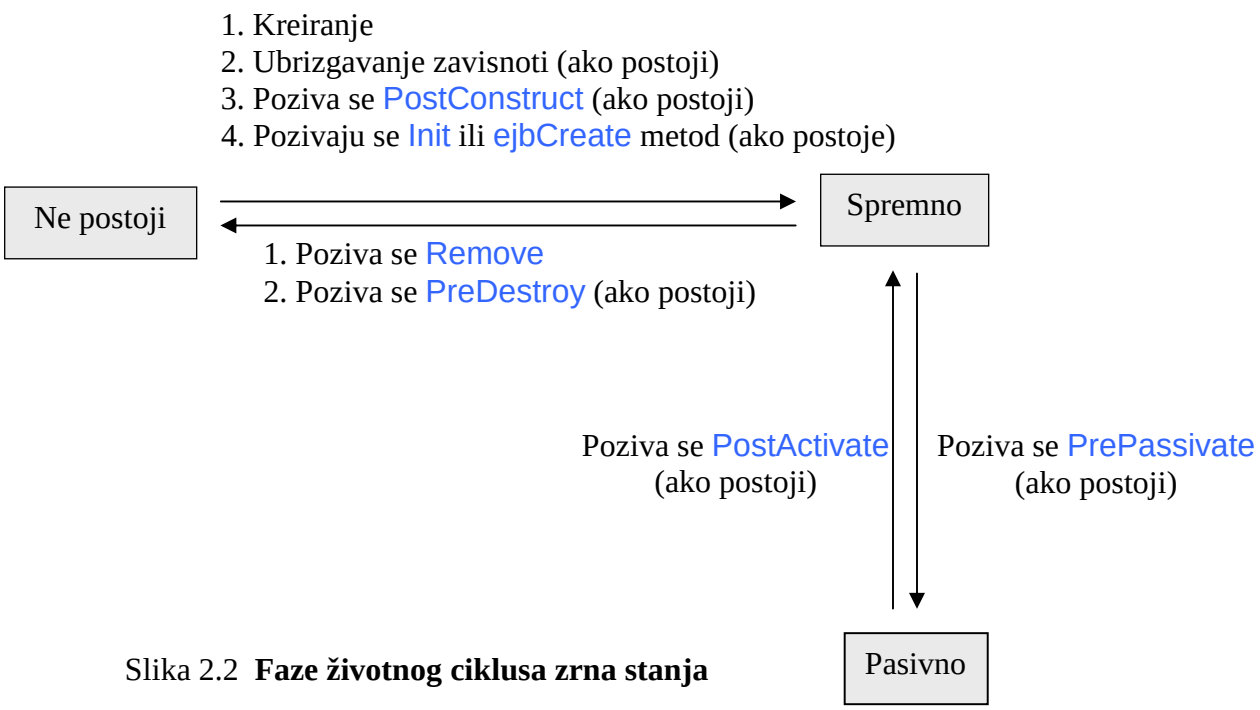
Pomoću zrna sesije takođe možemo slati JMS porkuke, ali ne možemo da ih primamo asihrono. Ako želimo asihrono primanje poruka, onda ćemo se odlučiti za zrna vođena porukom.

2.4 Životni ciklus Enterpise Java zrna

Enterprise Java zrno prolazi kroz različita stanja tokom svog života. Svaka od različitih vrsta zrna, (zrno stanja, zrno bez stanja, zrno jedinac, zrno vođeno porukom) ima svoj životni put.

Životni cikus zrna stanja

Slika 2.2 ilustruje faze životnog ciklusa zrna stanja. Referenciranjem zrna stanja inicira se ubrizgavanje eventualnih zavisnosti ako ih ima, ubrizgavanje se vrši od strane kontejnera a zatim se poziva metoda obeležena sa @PostConstruct, ako takava metoda postoji. Po izvršavanju Init metode, ako postoji, instanca zrna postaje spremna za izvršavanje metoda ukoliko budu pozvane od strane klijenta.



Slika 2.2 **Faze životnog ciklusa zrna stanja**

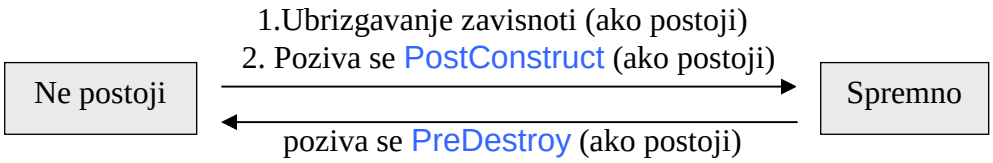
Dok se nalazi u fazi Spremno, kontejner može odlučiti da izmesti instancu zrna iz radne memorije stavljajući je u pasivno stanje. Pri tome, prilikom izbora instance za stavljanje u pasivno stanje, kontejner obično bira onu koja najduže nije bila korišćena. Tom prilikom, ako postoji, poziva se metoda obeležena sa @PrePassivate. Ako klijent pozove neku od metoda zrna dok je njegova instanca u pasivnom stanju, kontejner ponovo aktivira instancu zrna, poziva metodu označenu sa @PostActivate, ako ona postoji i instanca je ponovo u fazi Spremno.

Na kraju životnog ciklusa, klijent poziva metodu označenu sa @Remove i to je jedina metoda koju poziva klijent, sve ostale metode poziva kontejner zrna. Poslednja u nizu je metoda obeležena sa @PreDestroy, ukoliko postoji.

Životni cikus zrna bez stanja

Životni ciklus zrna bez stanja ima samo dva stanja, instanca ili postoji i spremna je za pozivanje metoda zrna ili ne postoji.

Kontejner zrna pravi i održava kolekciju instanci zrna bez stanja, ubrizgava zavisnosti i poziva metodu označenu sa @PostConstruct, ukoliko ona postoji. Kada se to obavi, instanca zrna je u fazi Spremno.

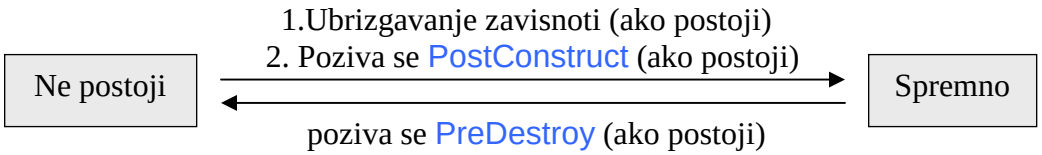


Slika 2.3 **Faze životnog ciklusa zrna bez stanja**

Na kraju životnog ciklusa, kontejner poziva metodu označenu sa @PreDestroy ukoliko ona postoji i tu se život instance zrna bez stanja završava.

Životni cikus zrna jedinca

Isto kao kod zrna bez stanja, zrno jedinac ima samo 2 faze svog životnog ciklusa. Instanca ovog zrna ili ne postoji ili postoji i spremna je za pozive metoda zrna. Kontejner zrna započinje životni ciklus zrna jedinca praveći instancu ovog zrna. Ukoliko je zrno jedinac označeno deklaracijom @Startup, pravljenje ove instance događa se prilikom raspoređivanja(*deployment*) aplikacije na serveru. Po ubrizgavanju eventualnih zavisnosti i pozivu metode označene sa @PostConstruct od strane kontejnera, instanca prelazi u fazu Spremno.



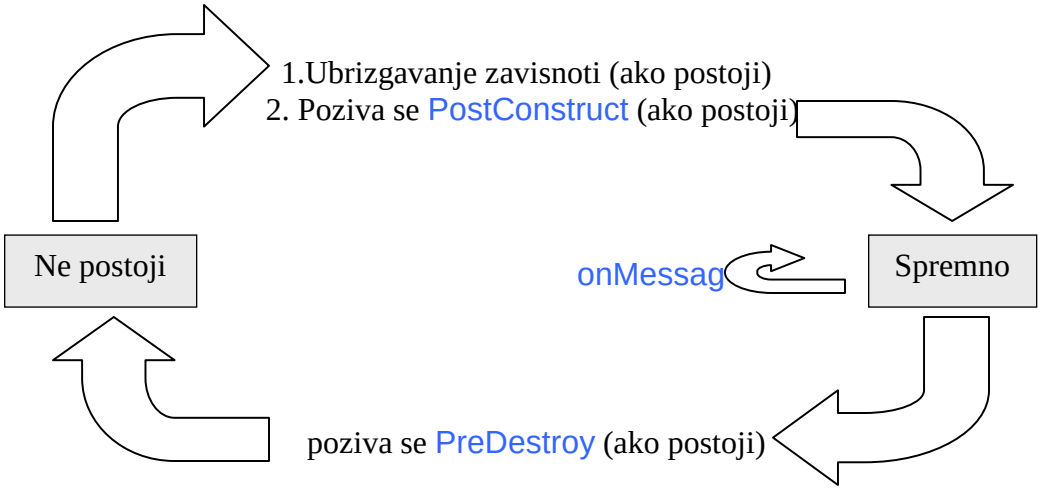
Slika 2.4 **Faze životnog ciklusa zrna jedinca**

Životni ciklus zrna jedinca završava se kontejnerovim pozivom metode označene sa @PreDestroy, ukoliko ona postoji i to označava kraj instance ovog zrna.

Životni cikus zrna vođenog porukom

Kontejner zrna pravi kolekciju instanci zrna vođenih porukom. Za svaku od tih instanci kontejner obavlja zadatke ubrizgavanja zavisnosti i poziva metodu označenu sa @PostConstruct, ukoliko ona postoji. Time zrno prelazi u fazu Spremno.

Instanca zrna vođenog porukom, koja se nalazi u fazi Spremno, služi da nad njom bude pozvana metoda *onMessage*. Pozivanjem ove metode instanca zrna opravdava svrhu svog postojanja.



Slika 2.5 **Faze životnog ciklusa zrna vođenog porukom**

Kao i kod zrna sesije bez stanja, na kraju životnog ciklusa kontejner poziva metodu označenu sa @PreDestroy i život instace zrna se završava.

2.5 Klase entiteta i njima odgovarajuće klase fasade

Klasa entiteta je Java klasa koja predstavlja neki entitet i najčešće odgovara tabeli relacione baze podataka. U tom slučaju svaka instanca klase entiteta odgovara jednom redu u odgovarajućoj tabeli baze. Atributi klase entiteta mapiraju se sa poljima u odgovarajućoj tabeli baze. Primarni ključ u klasi entiteta obeležava se Java anotacijom @id

Na primer:

```
@id
private Long pirmarniKljuč;
```

Klase entiteta moraju zadovoljavati sledeće uslove:

- klasa mora biti obeležena javax.persistence.Entity anotacijom;
- klasa mora imati javni ili zaštićeni konstruktor bez argumenata. Pred toga dozvoljeni su i drugi konstruktori;
- klasa, njene metode i atributi ne smeju biti deklarisani kao konačni (*final)*;
- ako će instance klase biti prenošene po vrednosti, recimo udaljenim interfejsom nekog zrna sesije, tada klasa mora implementirati *Serializable* interface;
- Atributi klase moraju biti deklarisani kao privatni, zaštićeni ili privatni u okviru paketa u kome se nalaze, i može im se pristupati jedino putem za to predviđenih metoda klase;

Često se za klase entiteta implementiraju njima odgovarajuće klase fasade. Klase fasade implementiraju se kao zrna bez stanja i kao takva na raspolaganju su im funkcionalnosti kontejnera Enterpirse Java zrna. Implementiranjem klasa fasada izbegava se tesna povezanost izmedju komponenata pa se one mogu razmestiti po različitim mašinama ako za to ima potrebe, a komunikacija se može obavljati preko udaljenog interfejsa klase fasade, odnosno zrna bez stanja.

Na primer, ako imamo klasu entiteta *entities.Student.java*

Da bi klasa *ejb.StudentFacade.java* postala klasa fasade za klasu *Student*, potrebno je u datoteci *StudentFacade* upisati sledeće

```
import entities.Student;
import javax..ejb.Stateless;
@Stateless
public classStudentFacade{...}
```

3. **Izrada edukativne Veb aplikacije**

3.1 O samoj aplikaciji

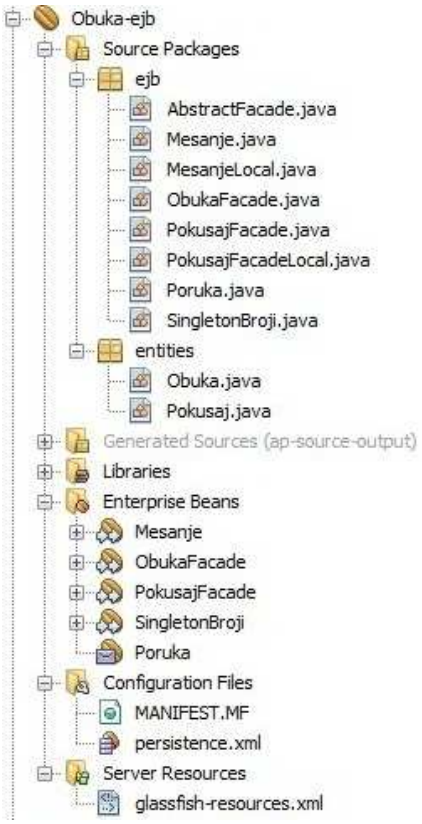
Razvijena Veb aplikacija predstavlja centar za obuku. Postoje dva različita tipa korisnika centra, a to su:

- Obični korisnici
- Profesori

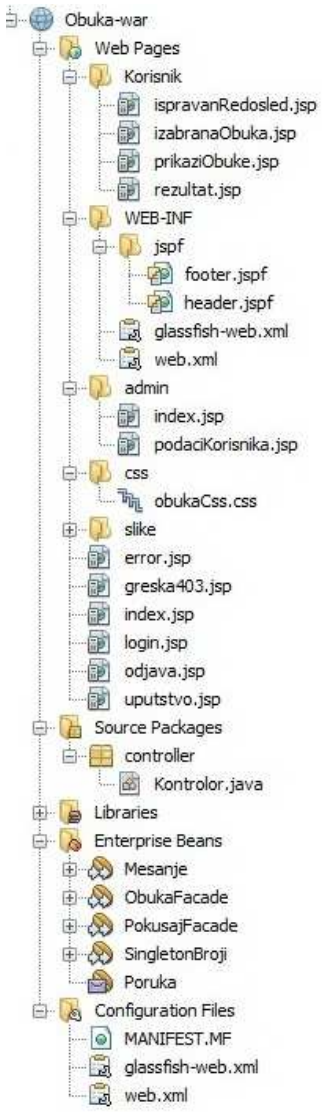
Obični korisnici centra logovanjem na sistem imaju pristup obukama koje su trenutno na raspolaganju, tj. koje se trenutno nalaze u bazi. Obuke se sastoje od postavljanja određenih pojmova u hronološki redosled. Na primer po izboru obuke za podizanje operativnog sistema od korisnika se traži da postavi faze podizanja operativnog sistema koje su prikazane u slučajnom redosledu hronološki. Ukoliko korisnik uspešno obavi zadatak, pruža mu se mogućnost odabira nove obuke, a u suprotnom može pokušati ponovo ili pogledati ispravno rešenje.

Profesori logovanjem na sistem imaju pregled običnih korisnika koji su trenutno aktivni. Ako se neki od korisnika izloguje, istekne mu sesija ili se prijavi na sistem, profesor ima uvid u to. Takođe, profesor može za svakog od aktivnih korisnika pogledati koliko je uspešnih i neuspešnih obuka korisnik do sada uradio i kakav mu je ukupan učinak.

EJB modul aplikacije prikazan je na slici 3.1, a Veb modul na slici 3.2, na kojima se vidi kakva je organizacija datoteka i kako su datoteke raspoređene u okviru modula.



Slika 3.1 **EJB modul aplikacije**



Slika 3.2 **Veb modul aplikacije**

3.2 Instalacija razvojnog okruženja

Za izradu Java EE aplikacije od softvera korišćeno je sledeće:

- Java Development Kit (JDK) 6
- Java EE 6 SDK
- NetBeans IDE 7.0
- GlassFish Server Open Source Edition 3.1

Instalacije su jednostavne, prvo je potrebno instalirati JDK6 a zatim je najlakše sa sajta *netbeans.org* preuzeti i instalirati paket koji sadrži NetBeans IDE 7.0, Java EE6 SDK i GlassFish Server 3.1.

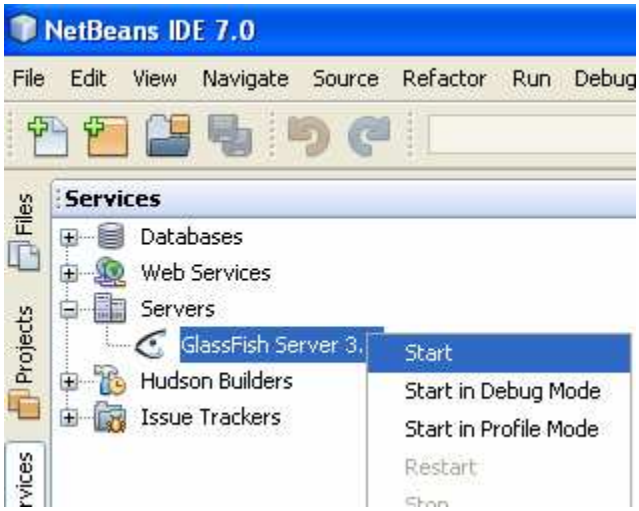
GlassFish je softverski server za Java EE platformu. On podržava Java EE API specifikacije, među kojima su JMS, Veb servisi i XML. Takođe, GlassFish svojim Veb i EJB kontejnerima predstavlja interfejs između Java EE komponenti i sistemskih funkcionalnosti Java platforme.
Pored GlassFish servera ima i drugih popularnih servera kao što su Apache Geronimo i Tomcat koji se mogu koristiti u istu svrhu. Pošto je za razvoj ove aplikacije izabran GlassFish server, naredna poglavlja su zasnovana na korišćenju ovog softvera.

3.3 Pokretanje aplikacije

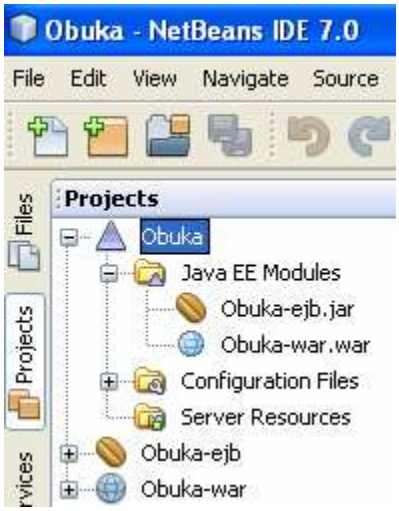
Sve datoteke sa izvornim kodom spakovane su u datoteku *PrimenaZrna.zip* koji raspakivanjem daje direktorijum *Obuka* u kojem se nalazi izvorni kod i direktorijum *Derbi_baza* koji sadrži podatke baze. Aplikacija je smeštena u EAR datoteku *Obuka*, koja sadrži EJB JAR modul *Obuka-ejb* i WAR modul *Obuka-war*. U modulu *Obuka-ejb* nalaze se Enterpise Java zrna i klase entiteta a u modulu *Obuka-war* nalaze se klasa Servleta, JSP strane i CSS datoteka. Naravno, svaki od modula sadrži i odgovarajuće deskriptore.

Da bi se aplikacija pokrenula po prvi put, potrebno je prethodno izvršiti povezivanje sa bazom *centar* i registrovati nove korisnike aplikacije na lokalnoj kopiji GlassFish servera. Da bi se to obavilo mogu se pratiti sledeći koraci:

1. Prvo je potrebno startovati GlassFish server, a to se može uraditi u *Services* prozoru NetBeans okruženja, desnim klikom na GlassFish Server pa zatim odabirom opcije *Start* (Slika 3.3)

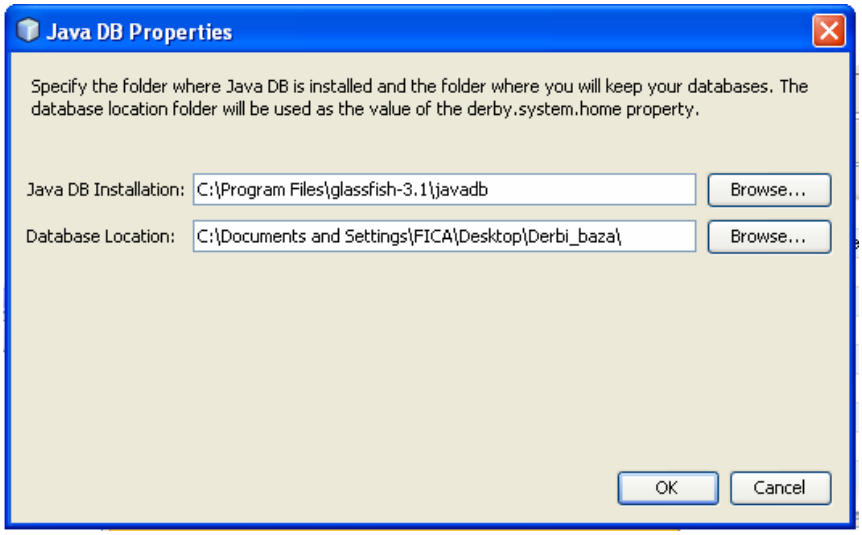


Slika 3.3 **Startovanje GlassFish servera**



Slika 3.4 **Moduli aplikacije**

2. Za povezivanje sa bazom podataka *centar*, treba kliknuti na + kod "Databases" pa zatim desnim klikom na "Java DB" izaberati opciju *properties.* U polje *Database location* treba uneti putanju do direktorijuma *Derbi_baza* koji se dobija raspakivanjem datoteke *PrimenaZrna.zip*. (Slika 3.5)



Slika 3.5 **Unošenje lokacije Derbi baze**

3. Potrebno je registrovati nove korisnike na lokalnom primerku GlassFish servera. Pogledati poglavlje "3.6 Registracija novih korisnika".

4. Odlaskom na *File -> Open project* možemo otvoriti projekat u NetBeans okruženju.(Samo izvršimo navigaciju do mesta gde smo raspakivanjem dobili direktorijum *Obuka* i NetBeans će prepoznati projekat).

5. Rasporedimo aplikaciju na server Desnim klikom na *Obuka* i odabirom opcije *deploy.*

6. I konačno, aplikaciju možemo pokrenuti desnim klikom miša na *Obuka* i zatim odabirom opcije *run.*

7. Veb pretraživač bi trebalo da bude otvoren od strane okruženja. Za slučaj da se to nije desilo otvorićemo Veb pretraživač i uneti localhost:8080/Obuka-war/ u adresnu liniju pretraživača.

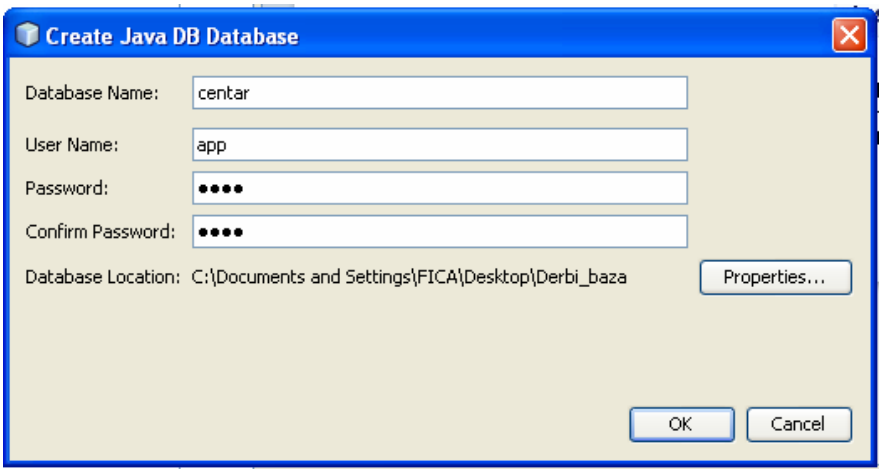
Za istovremeno logovanje više korisnika sa istog računara, u cilju testiranja aplikacije, najbolje je koristiti različite Veb pretraživače.

Kada smo jednom završili sa opisanim koracima, svako naredno pokretanje aplikacije je jednostavano. Koraci 2-4 nam više nisu potrebni.

Opisan postupak odgovara pokretanju aplikacije na lokalnom računaru. Naravno krajnji korisnik neće pokretati aplikaciju već će po dobijanju naloga za pristup, aplikaciji koja se nalazi na udaljenom serveru pristupati putem Interneta.

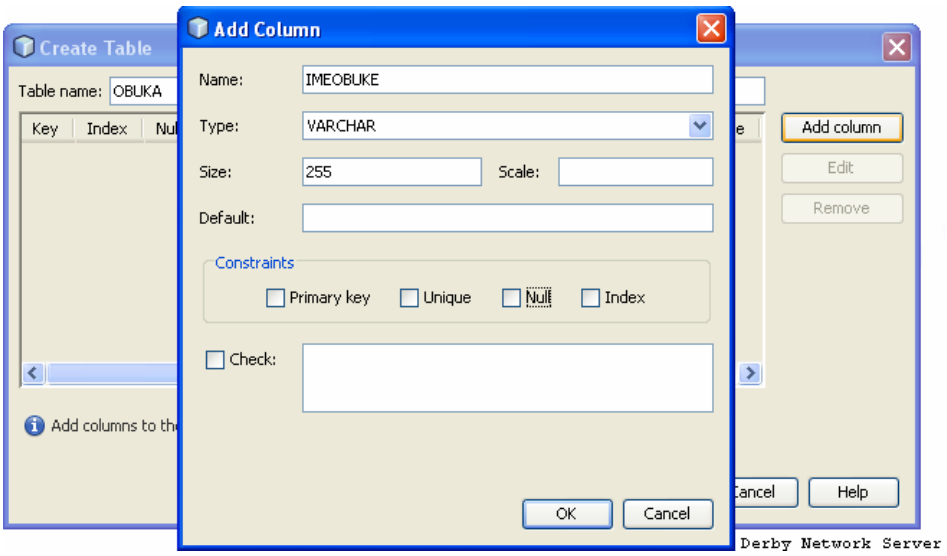
3.4 Baza podataka

Za bazu podataka u izradi aplikacije korišćena je *Java DB (Derby)* baza.
Java DB baza je distribucija *Apache Derby* sistema. *Java DB server* napisan je u Javi i podržava SQL, JDBC API i Java EE tehnologiju.
Java DB baza je spakovana u okviru GlassFish servera a nalazi se takođe i u JDK 6.
Za kreiranje nove baze u Netbeans-u potrebno je otići u Service prozor. Desnim klikom miša na *Java DB* moguće je pokrenuti i zaustaviti server, kao i kreirati novu bazu.



Slika 3.6 **Kreiranje Java DB baze**

Po kreiranju baze, pravljenje novih tabela moguce je bilo putem SQL-a ili pomoću NetBeans-ovog dijaloga



Slika 3.7 **Pravljene tabela i dodavanje kolona tabele**

Dodavanje kolona u tabelu preko NetBeans-ovog dijaloga takođe je jednostavno. Korišćene su 3 tabele:

OBUKA

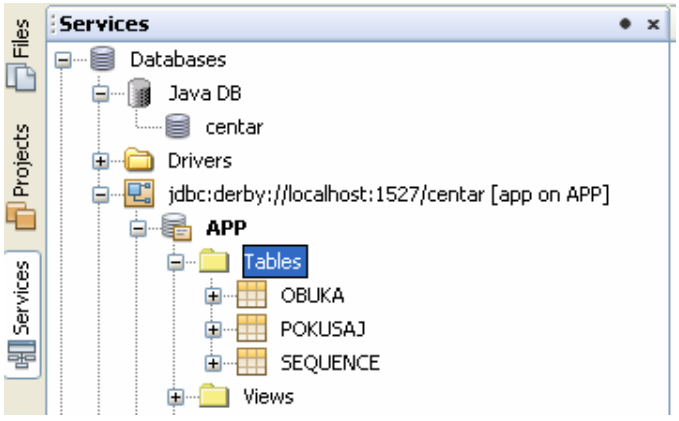
POKUSAJ

SEQUENCE

U prvoj tabeli OBUKA se čuvaju podaci o imenu obuke, broju elemenata koje obuka sadrži, kao i sami elementi obuke.

Druga tabela je tabela POKUSAJ u kojoj se beleži svaki pokusaj korisnika. U ovoj tabeli čuvaju se podaci o tome koji korisnik je obavio koju obuku, kao i to da li je pokušaj bio uspešan ili ne.

Tabelu SEQUENCE koristimo prilikom generisanja primarnih ključeva tabele POKUSAJ



Slika 3.8 **Baza *centar* i njene tabele**

Persistence unit definiše skup svih klasa entiteta koje su predstavljene tabelama u jednoj relacionoj bazi podataka. i definiše se u XML datoteci *persistence.xml*, koja se nalazi u *Obuka-ejb* modulu. Definisaćemo novi *persistence unit* i nazvati ga *Obuka-ejbPU*. Prilikom raspoređivanja aplikacije na server, iz datoteke *persistence.xml* biće pročitane informacije vezane za to koja baza podataka će se koristiti u aplikaciji.
Da bismo naznačili da ćemo koristiti novonapravljenu bazu *centar*, u datoteci *persistence.xml* navešćemo sledeće:

```
<jta-data-source>jdbc/centar</jta-data-source>
```

Takođe u istoj datoteci navešćemo i da se to odnosi na sve klase entiteta (*Obuka.java* i *Pokusaj.java*)

```
<exclude-unlisted-classes>false</exclude-unlisted-classes>
```

Klasi entiteta *Obuka.java* odgovara tabela OBUKA a klasi entiteta *Pokusaj.java* odgovara tabela POKUSAJ u bazi. Jedan objekat klase entiteta predstavlja jednu vrstu u odgovarajućoj tabeli baze.
Takođe biće potrebno da u zrnima sesije, koja pravimo za svaku od klasa entiteta, navedemo da ćemo koristiti *Obuka-ejbPU*, a to možemo uraditi upisom sledeće linije koda:

```
@PersistenceContext(unitName = "Obuka-ejbPU")
```

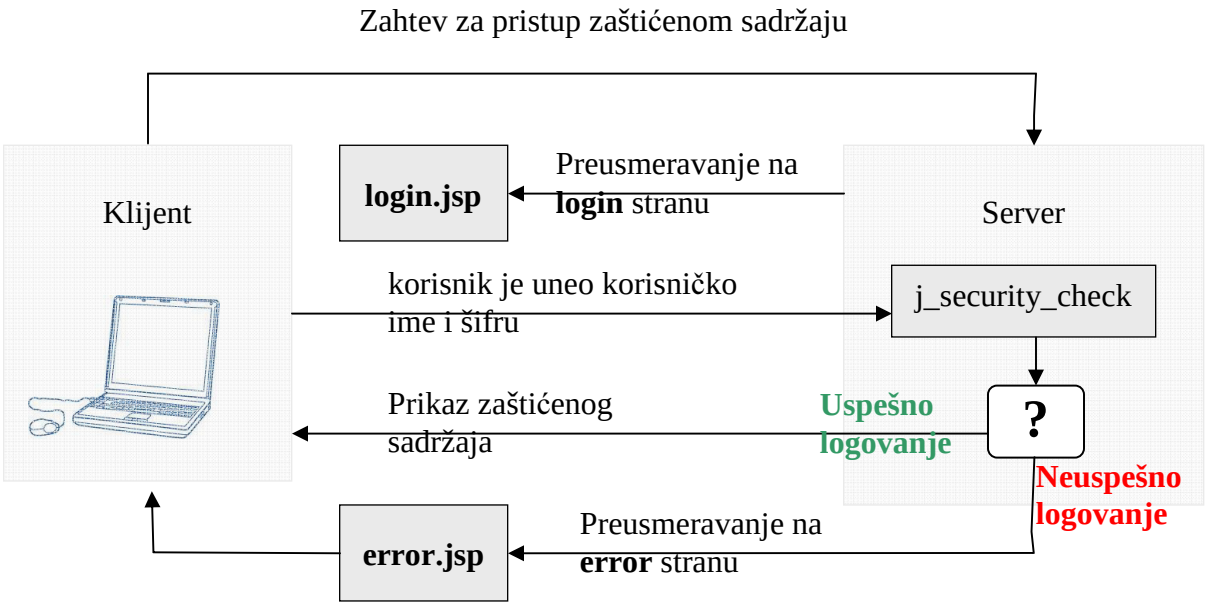
u datoteke *ObukaFacade.java* i *PokusajFacade.java.*

O zrnima sesije, koja pravimo za svaku od klasa entiteta, bice reči kasnije.

3.5 Autentifikacija korisnika

Java EE platforma podržava različite tipove autentifikacije, a za potrebe ove aplikacije korišćena je autentifikacija zasnovana na formi za logovanje (*form based authentication*). Proces autentifikacije odvija se u četiri koraka:

1. korisnik zahteva pristup zaštićenom sadržaju;
2. server prepoznaje da je zatražen pristup zaštićenom sadržaju i preusmerava korisnika na login stranu;
3. korisnik unosi svoje korisničko ime i šifru koristeći formu za logovanje;
4. ukoliko su uneti podaci odgovarajući prikazuje se zahtevani sadržaj a u suprotnom prikazuje se poruka o grešci prilikom logovanja;



Slika 3.9 **Proces autentifikacije zasnovane na formi za logovanje**

Implementacija autentifikacije zasnovane na formi sastoji se od pravljenja strane koja sadrži login formu, *login.jsp* i strane koja će biti prikazana u slučaju neuspešnog logovanja, *error.jsp* (ove datoteke se mogu nazvati i drugačije) ali i od unošenja odgovarajućih informacija u deskriptor raspoređivanja *web.xml*. Da bi se kontejner servleta obavestio o autentifikaciji, u deskriptor *web.xml* potrebno je upisati sledeće (nije bitno u koji deo datoteke):

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/error.jsp</form-error-page>
```

```
        </form-login-config>
    </login-config>
```

Kako imamo dve različite vrste korisnika aplikacije a to su obični korisnici i profesori, u deskriptoru raspoređivanja moramo definisati ove korisničke grupe pa je u *web.xml* upisano sledeće:

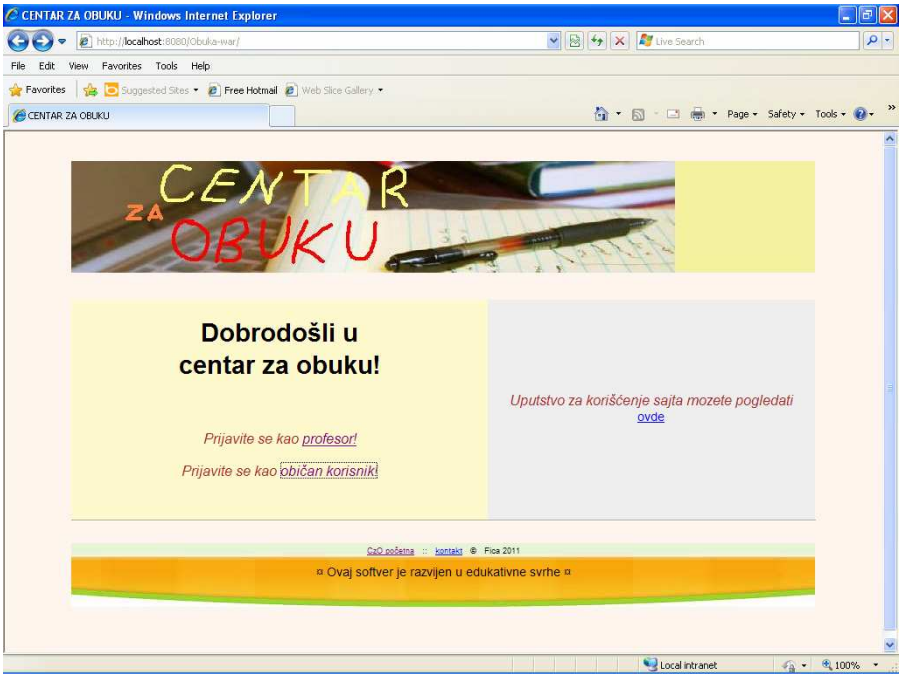
```
<security-role>
        <description>Prostor za Profesore</description>
        <role-name>admin</role-name>
</security-role>
<security-role>
        <description>Prostor za obicne korisnike</description>
        <role-name>Korisnik</role-name>
</security-role>
```

Da bismo definisali prava pristupa za svaku od ove dve grupe u deskriptor *web.xml* upisujemo sledeće:

```
<security-constraint>
        <display-name>admin</display-name>
        <web-resource-collection>
            <web-resource-name>
                Profesorska_zona
            </web-resource-name>
            <description>
                Sadrzaji kojima mogu pristupati samo profesori
            </description>
            <url-pattern>/admin/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <description/>
            <role-name>admin</role-name>
        </auth-constraint>
</security-constraint>
<security-constraint>
        <display-name>Korisnik</display-name>
        <web-resource-collection>
            <web-resource-name>
                Zona_obicnih_korisnika
            </web-resource-name>
            <description>
                Sadrzaji kojima mogu  pristupati obicni korisnici
            </description>
            <url-pattern>/Korisnik/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <description/>
            <role-name>Korisnik</role-name>
```

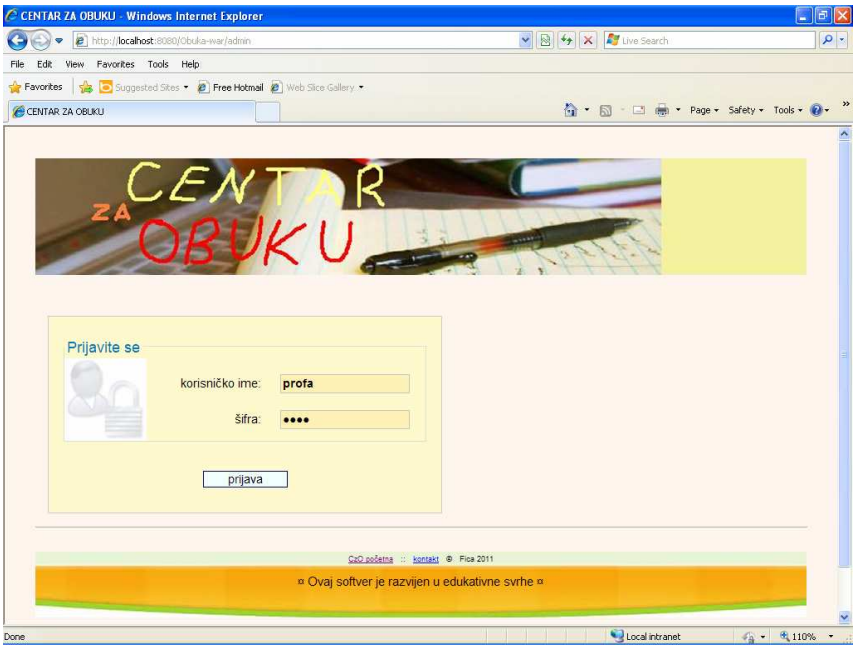
```
        </auth-constraint>
    </security-constraint>
```

Na početnoj Veb strani aplikacije korisniku se nudi da se prijavi na sistem kao profesor ili kao običan korisnik (Slika 3.10).



Slika 3.10 **Početna strana**

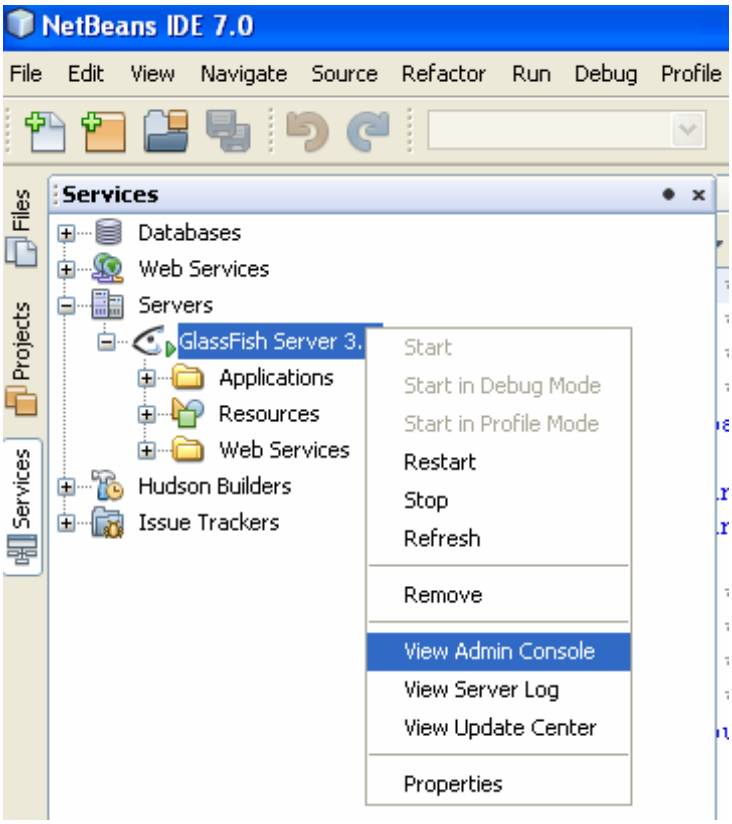
Zatim, po kliku na link za prijavu, od korisnika se traži da se uloguje unoseći svoje korisničko ime i šifru(Slika 3.11).



Slika 3.11 **Prijava na sistem**

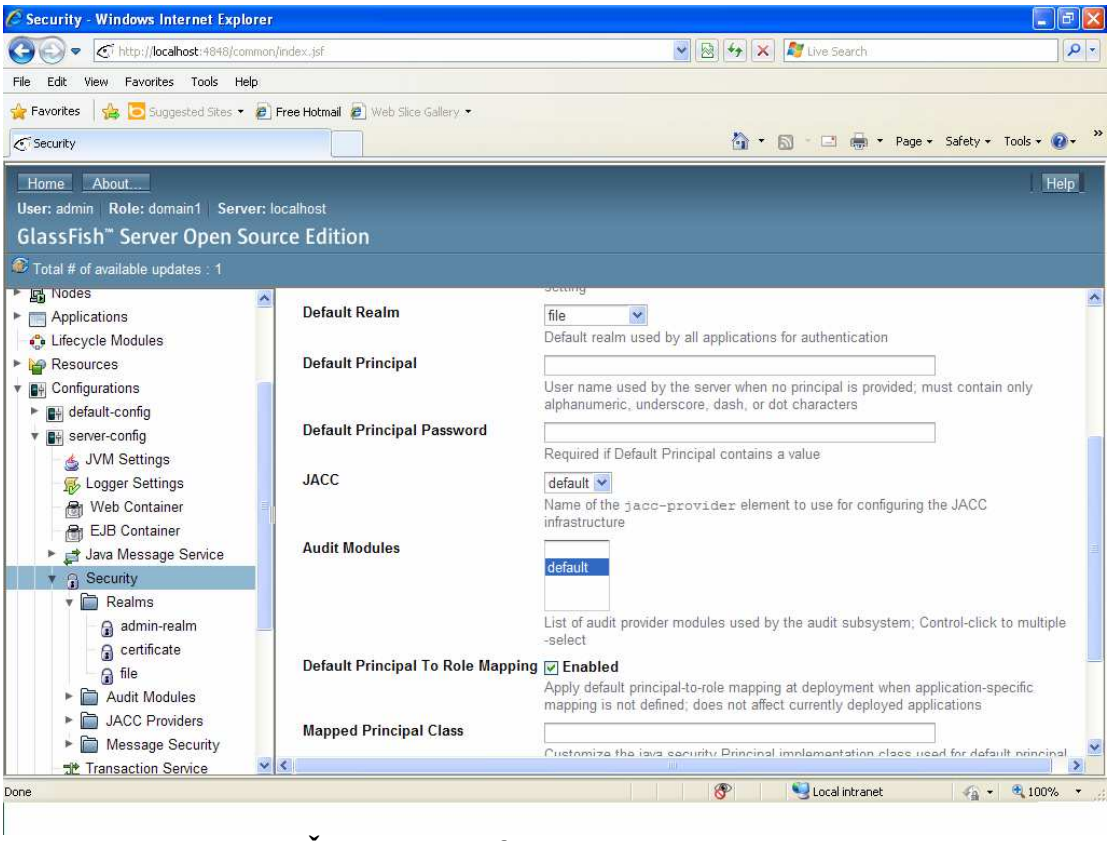
3.6 Registracija novih korisnika

Pri implementaciji aplikacije korišćen je *File realm* koji predstavlja tekstualnu datoteku u kojoj GlassFish server čuva šifre i korisnička imena za logovanje korisnika. Da bismo napravili korisnički nalog prvo je potrebno da otvorimo admin konzolu GlassFish servera. Admin konzola je korisničnki interfejs za administraciju servera zasnovan na Veb pretraživaču. Ona se obično nalazi na portu 4848 pa joj možemo pristupiti iz Veb pretraživača odlaskom na: http://localhost:4848/ ili iz NetBeans okruženja odlaskom u *Services* prozor, klikom na + da bi nam se otvorila *Servers* lista, startovanjem GlassFish servera i zatim odabirom *View Admin Console* posle desnog klika na *GlassFish Server* (Slika 3.12)



Slika 3.12 **Otvaranje admin konzole u Veb pretraživaču**

Po otvaranju admin kozole u pretraživaču, klikom na redom,
Configurations -> server config ->Security
dolazimo do Security sekcije.
Tu je potrebno izabrati:
Default Principal To Role Mapping (Slika 3.13) i potom kliknuti na *save.*



Slika 3.13 **Štikliranje Default Principal To Role Mapping**

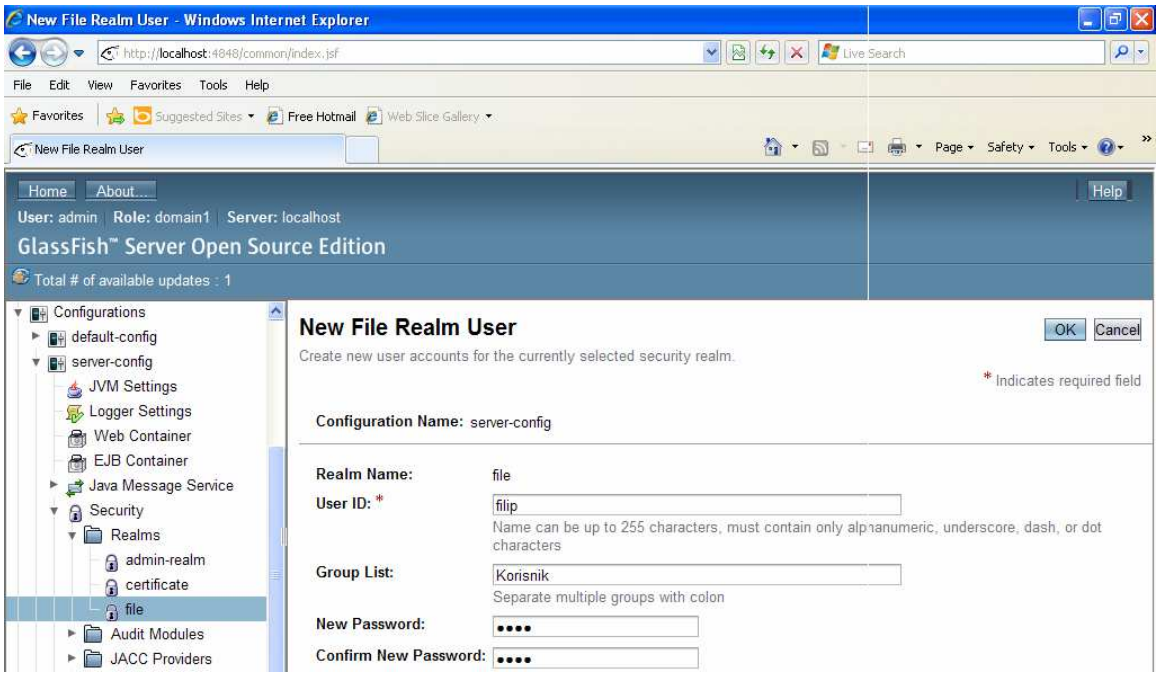
Dalje, klikom na *Realms -> file* možemo editovati *file realm*.
Sada novog korisnika možemo registrovati klikom na *Manage users* pa zatim na *New*

Kada smo u prozoru "New File Realm User"(Slika 3.14):
Izabraćemo *User id* i *password* po želji.

Da bismo registrovali običnog korisnika u polje *Group List* stavićemo:
Korisnik

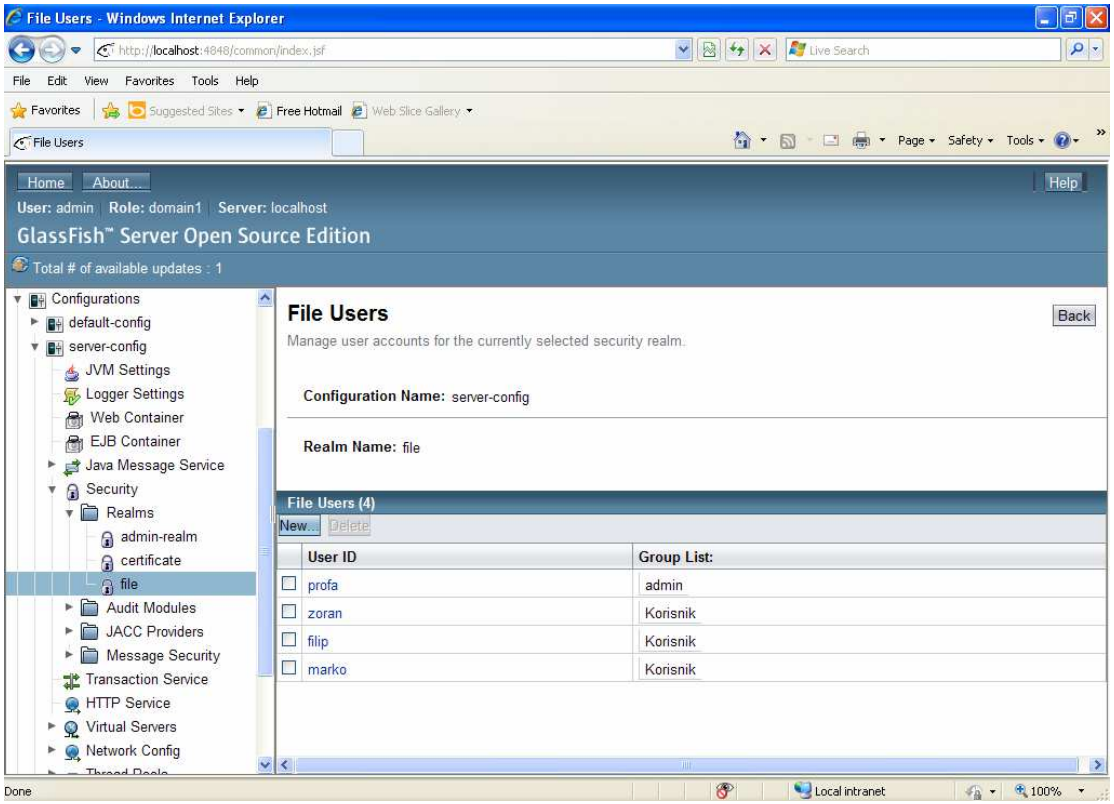
Da bismo registrovali novog profesora u polje *Group List* stavićemo**:**
admin

Po završetku pravljenja novih korisnika restartovaćemo GlassFish server.



Slika 3.14 **Pravljenje običnog korisnika**

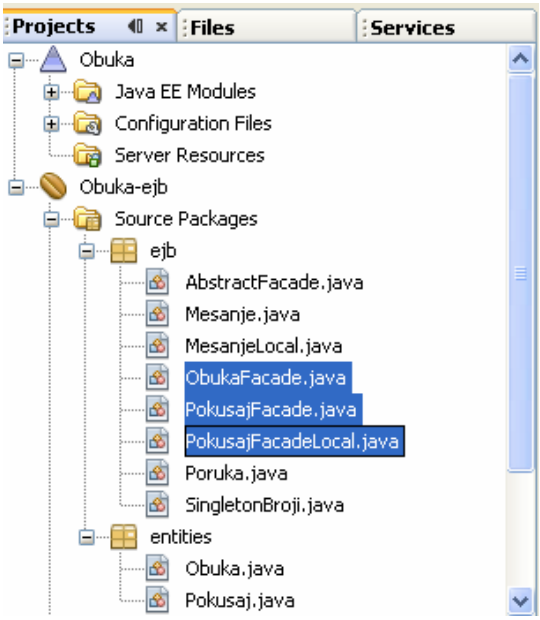
Po završetku registracije novih korisnika trebalo bi da to izgleda kao na slici 3.15



Slika 3.15 **Registrovani korisnici**

3.7 Zrna sesije za klase entiteta

Zrna sesije se koriste kao takozvane *klase fasade* za klase entiteta. Preko njih se vrši kreiranje, čitanje, ažuriranje i brisanje instanci klase entiteta (*CRUD Create-Read-Update-Delete*). Već smo pomenuli prednosti korišćenja zrna kao što su funkcionalnosti koje kontejner Enterprise Java zrna pruža.
Za potrebe aplikacije za svaku od klasa entiteta, *Pokusaj.java* i *Obuka.java* napravljena je odgovarajuća klasa fasade koja je ustvari zrno sesije. Tako za klasu entiteta Pokusaj.java imamo zrno sesije *PokusajFacade.java* a za Obuku.java tu je *ObukaFacade.java*. U pitanju su zrna sesije bez stanja jer instance ove klase fasade, po obavljenom poslu, postaju slobodne i na raspolaganju novim korisnicima. Takođe za zrno sesije *PokusajFacade.java* napravljen je lokalni interfejs *PokusajFacadeLocal.java*.
Zrna sesije i njima odgovarajući interfejsi smešteni su u paket EJB (Slika 3.16)



3.16 **Zrna sesije za klase entiteta sa lokalnim interfejsom**

EntityManager je Interfejs koji se koristi za interakciju sa instancama entiteta koje se nalaze u trajnim sadržajima, na primer u bazi podataka ili XML datoteci. *Entity Manager* je predstavljen javax.persistence.*EntityManager* instancom.
U klasama *ObukaFacade.java* i *PokusajFacade.java* pravi se *EntityManager* instanca tako što se koristi anotacija @PersistenceContext .

```
@PersistenceContext(unitName = "Obuka-ejbPU")
 private EntityManager em;
```

Entity Manager je kontrolisan od strane EJB kontejnera, a GlassFish server ga otvara i zatvara po potrebi. JPA je deo Java EE platforme koji mapiranjem Java objekata i podataka u relacionoj bazi omogućuje progameru da kroz Java objekte pristupa bazi. *Entity Manager* je integralna komponenta JPA-a (*Java Persistence API*).

Parametar *unitName* određuje *Persistence unit* a mi koristimo onaj koji smo definisali u deskriptoru *persistence.xml* a to je Obuka-ejbPU.
Tako metode zrna *ObukaFacade.java* pozivamo iz klase *Kontrolor.java* koja predstavlja Veb servlet aplikacije. Prvo koristimo anotaciju @EJB za ubrizgavanje zavisnosti:

```
@EJB
 private ObukaFacade obukaFacade;
```

Zatim u *nisci id_obuke* čuvamo id obuke koju je korisnik odabrao i koristeći metodu

```
public Obuka find(Object id)
```

zrna *ObukaFacade* pronalazimo u bazi obuku sa datim identifikatorom i smeštamo je u objekat klase *Obuka*.

```
String id_obuke = request.getParameter("odabranaObukaId");
Obuka izabranaObuka = obukaFacade.find(Long.valueOf(id_obuke));
```

Za zrno sesije *PokusajFacade* koristimo lokalni interfejs *PokusajFacadeLoca*l pa klasu definišemo sa :

```
public class PokusajFacade extends AbstractFacade<Pokusaj>
implements PokusajFacadeLocal) {...}
```

gde je *AbstractFacade* klasa koja sadrži standardne metode zajedničke za sve klase fasade klasa entiteta.

U klasi *Kontrolor.java* vršimo ubrizgavanje zavisnosti:

```
@EJB
private PokusajFacadeLocal pokusajFacade;
```

i zatim koristimo metodu lokalnog interfejsa *PokusajFacadeLocal*

```
//metoda vraca sve pokusaje korisnika sa datim korisnickim imenom
public List<Pokusaj> pokusajiKorisnika(String korisnickoIme){
        cb = em.getCriteriaBuilder();
        CriteriaQuery cq = cb.createQuery(Pokusaj.class);
        Root<Pokusaj> pokusaj = cq.from(Pokusaj.class);
        cq.select(pokusaj);
        cq.where(cb.equal(pokusaj.get(Pokusaj_.korisnickoIme),
        korisnickoIme));
        return em.createQuery(cq).getResultList();
}
```

pozivajući je preko instance lokalnog interfejsa:

```
List<Pokusaj> listaPokusaja =
pokusajFacade.pokusajiKorisnika(izabraniKorisnik);
```

U ovoj metodi korišćena je *Entity Manager* instanca *em* o kojoj je ranije već bilo reči. *EntityMagager API* koristi se za kreiranje, uništavanje i pronalaženje entiteta po primarnom ključu i za vršenje upita nad entitetima. Metoda em.getCriteriaBuilder() vraća *CriteriaBuilder* instancu *cb* za kreiranje *CriteriaQuery* objekta, zatim se izvršavanjem metode cb.createQuery(Pokusaj.class) pravi kriterijum upita za pretragu u kojoj će se tražiti pokušaji. Dalje se gradi upit tako da se pretraga vrši po atributu korisnickoIme koji je mapiran sa poljem KORISNICKOIME u tabeli POKUSAJ baze. Jedan red u tabeli baze odgovara jednoj instanci odgovarajuće klase. Po kreiranju kriterijuma pretage poziva se metod em.createQuery(cq) koji pravi upit nad bazom. Konačno, metod vraća sve pokušaje korisnika sa datim korisničkim imenom.

3.8 Servlet klasa i JSP strane

Servlet klasa *Kontrolor.java* smeštena je u Obuka-war modulu u paketu *controller*. U servlet klasi *Kontrolor.java* obrađujemo HTTP zahteve i, u zavisnosti od url obrasca, po obradi zahteva vršimo redirekciju ka odgovarajućem sadržaju JSP strana.
Veb servlete označavamo anotacijom @WebServlet

```
@WebServlet(name = "Kontrolor", loadOnStartup = 1, urlPatterns =
                        {"/Korisnik/index",
                        "/Korisnik/izabranaObuka",
                         "/Korisnik/provera",
                         "/admin",
                         "/admin/statistika"})
public class Kontrolor extends HttpServlet {...}
```

Ukoliko parametar loadOnStartup ima vrednost veću od 0, a u ovom slučaju ima, instanca klase servleta se pravi i inicijalizuje pri raspoređivanju aplikacije na GlassFish server.
urlPatterns definiše url obrasce na koje servlet klasa reaguje, preduzimajući odgovarajuće radnje i vršeći redirekciju.

Jedna instanca servleta može opsluživati više zahteva istovremeno gde se za svaki zahtev pokreće po jedna Java nit. Da ne bi pri svakom zahtevu korisnika stalno vršli čitanje istih podataka iz baze, prilikom inicijalizacije servleta, korisno je učitati iz baze obuke koje su na raspolaganju i smestiti ih u promenljivu *obuke* sa dosegom (*scope*) konteksta aplikacije. Obuke učitavamo preko instance klase *ObukaFacade* koja predsavlja zrno fasade za klasu entiteta *Obuka*.

```
@Override
public void init(ServletConfig servletConfig) throws ServletException {
```

```
        // inicijalizuje servlet sa informacijama konfiguracije
        super.init(servletConfig);
        // postavi obuke u servlet kontekst tj. u kontekst aplikacije
        getServletContext().setAttribute("obuke", obukaFacade.findAll());
    }
```

Na ovaj način štede se resursi servera.

Pored *init* metode, u servlet klasi imamo još dve metode, prva je:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException {...}
```

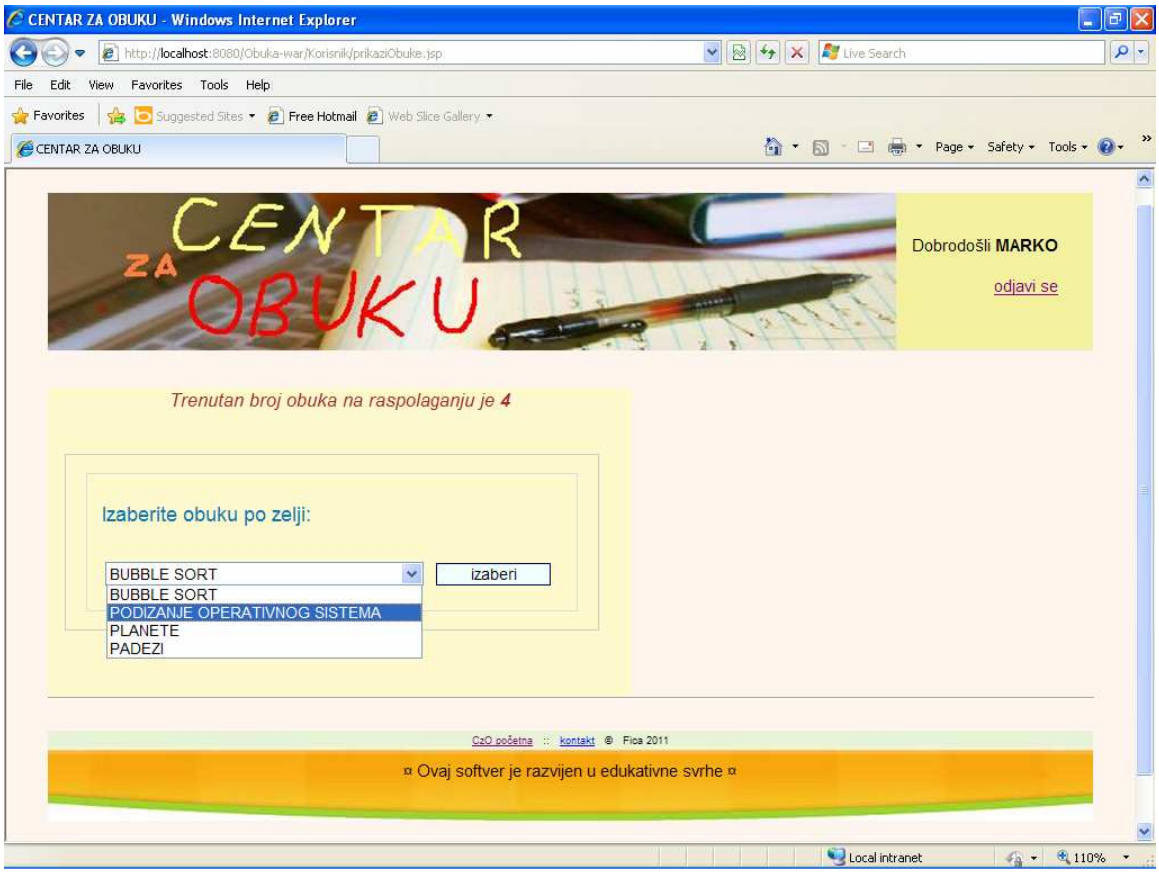
i služi za obrađivanje *HTTP get* zahteva, a druga je :

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {...}
```

i koristimo je za obrađivanje *HTTP pos* zahteva.

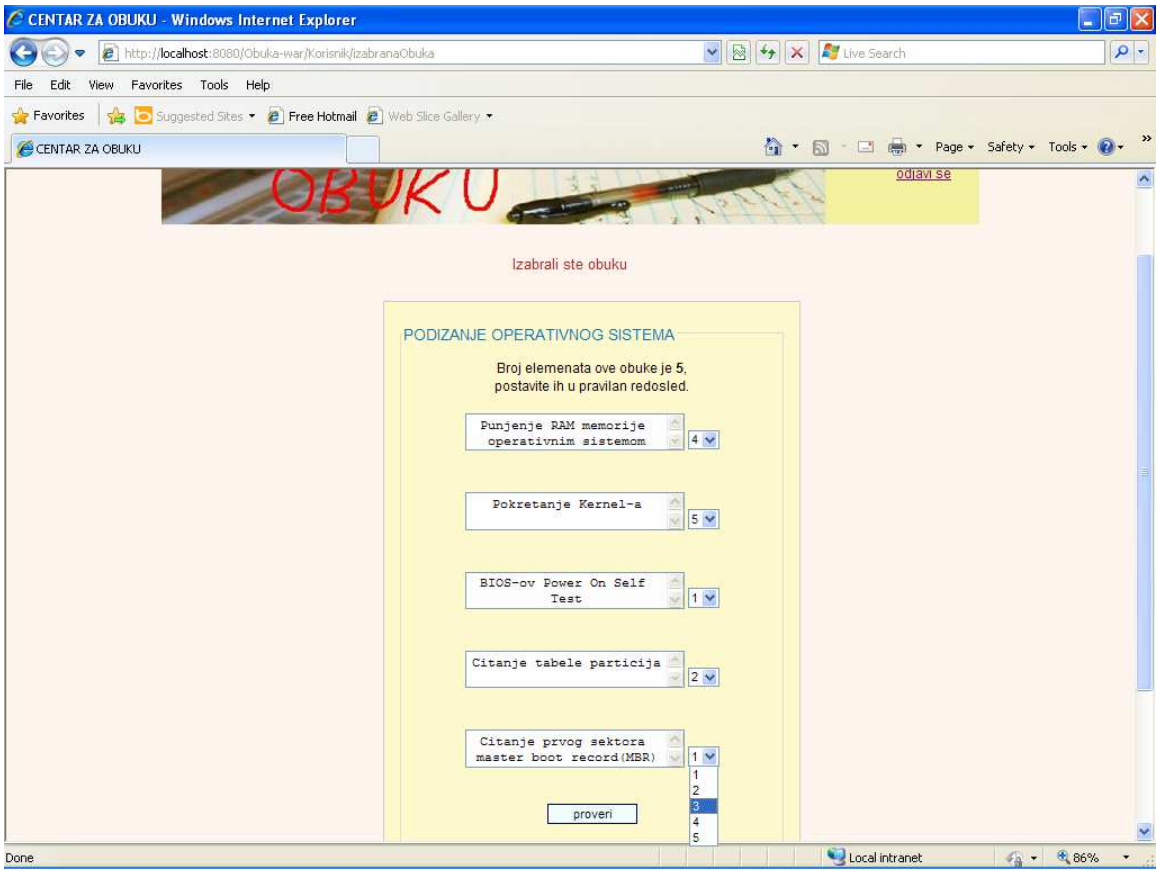
U metodi doGet ispitujemo url obrasce "/Korisnik/index" i "/admin" koji označavaju da se običan korisniki, tj. profesor uspešno ulogovao na sistem, a preostali obrasci ispituju se u doPost metodi.

Kada se običan korisnik uloguje, servlet vrši redirekciju i prikazuje se sadržaj JSP strane *prikaziObuke.jsp* (Slika 3.17)



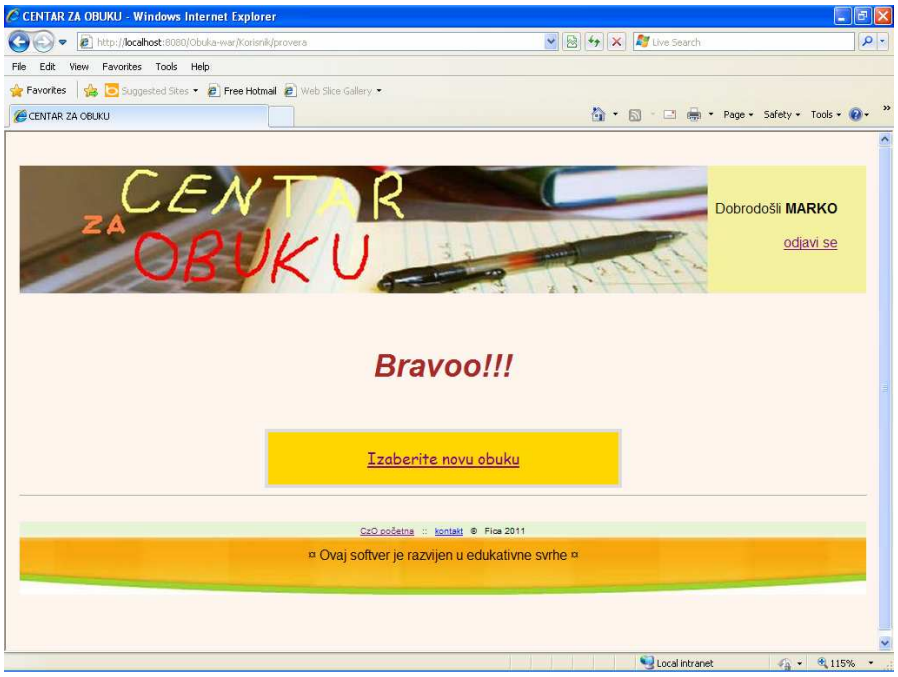
Slika 3.17 **Biranje obuke**

Kada korisnik izabere obuku, elementi izabrane metode se mešaju pomoću zrna sesije bez stanja *Mesanje.java* i prikazuju u slučajnom poretku (Slika 3.18).



Slika 3.18 **Prikazivanje izabrane obuke**

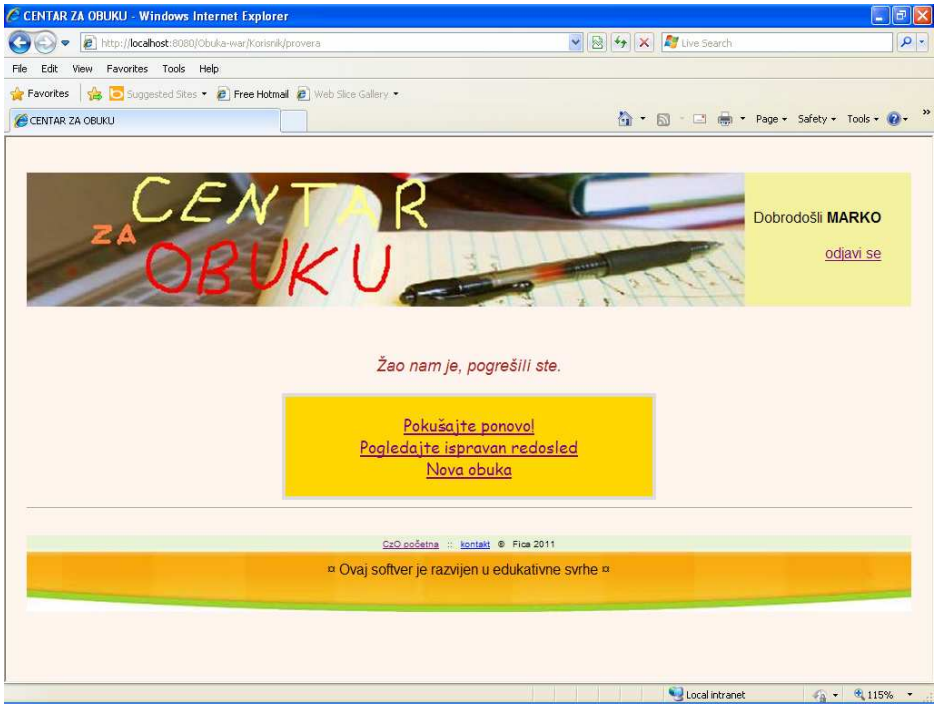
Kada korisnik klikne na dugme *proveri*, vrši se provera da li je njegov unos tačan ili nije. Provera da li je korisnik odabrao dva ista redna broja obavlja se na strani klijenta pomoću *javascript* metoda *proveriUnos* i ukoliko to jeste slučaj korisnik se opominje da proveri unos putem prozorčeta za dijalog (*dialog box*). U suprotnom, iz servleta se šalje poruka zrnu vođenom porukom *Poruka.java* za beleženje pokušaja u bazu a korisniku se prikazuje rezultat.



Slika 3.19 **Prikazivanje rezultata u slučaju uspeha**

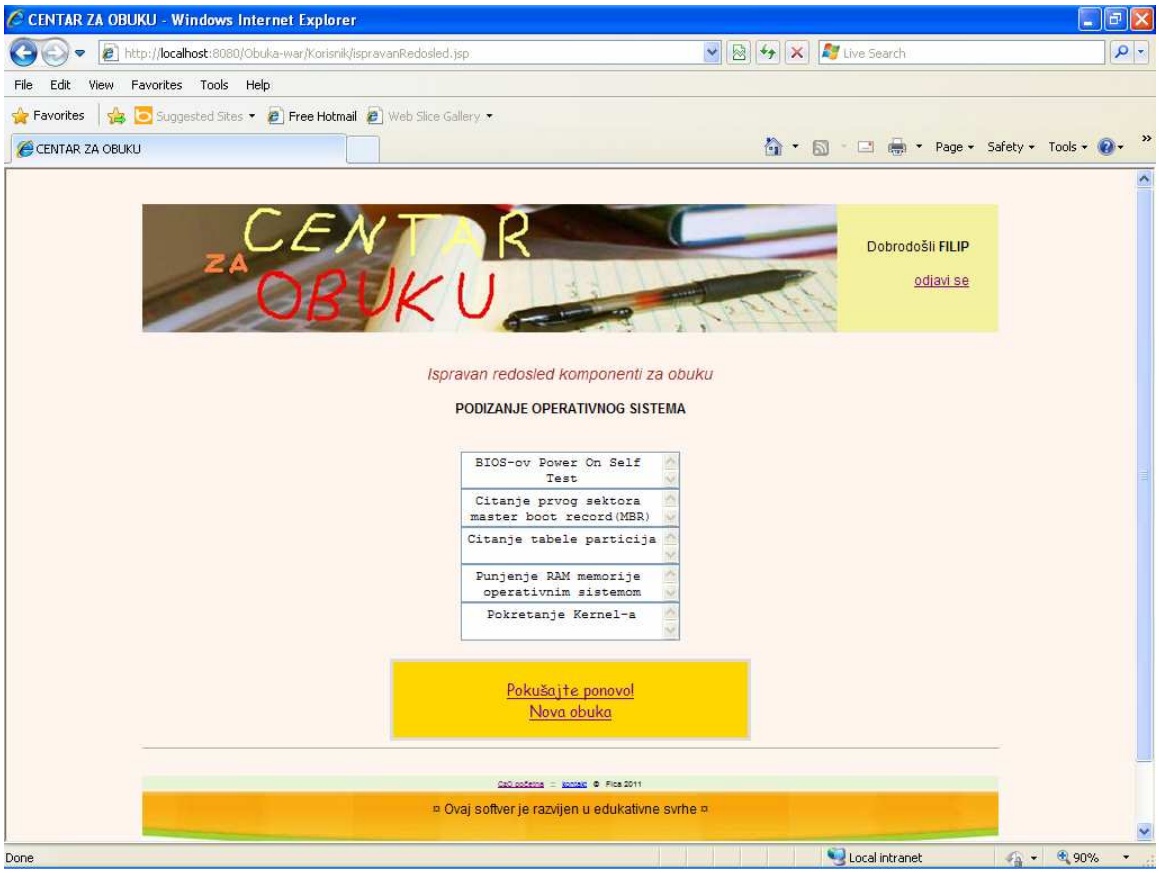
Ukoliko je odgovor tačan, korisnik može izabrati novu obuku(Slika 3.19).

Ukoliko odgovor nije tačan, korisnik može pokušati ponovo, izabrati novu obuku ili pogledati ispravan redosled elemenata (Slika 3.20).



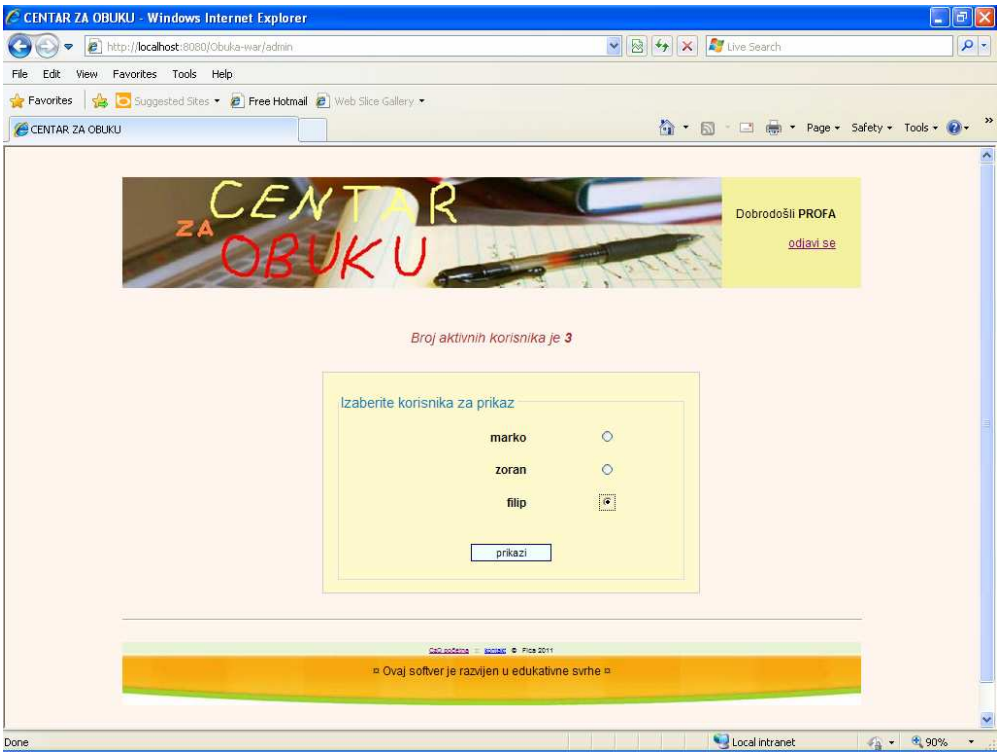
Slika 3.20 **Prikazivanje rezultata i opcija u slučaju neuspeha**

Praćenjem linka “Pogledajte ispravan redosled” prikazuje se ispravan poredak elemenata obuke (Slika 3.21)



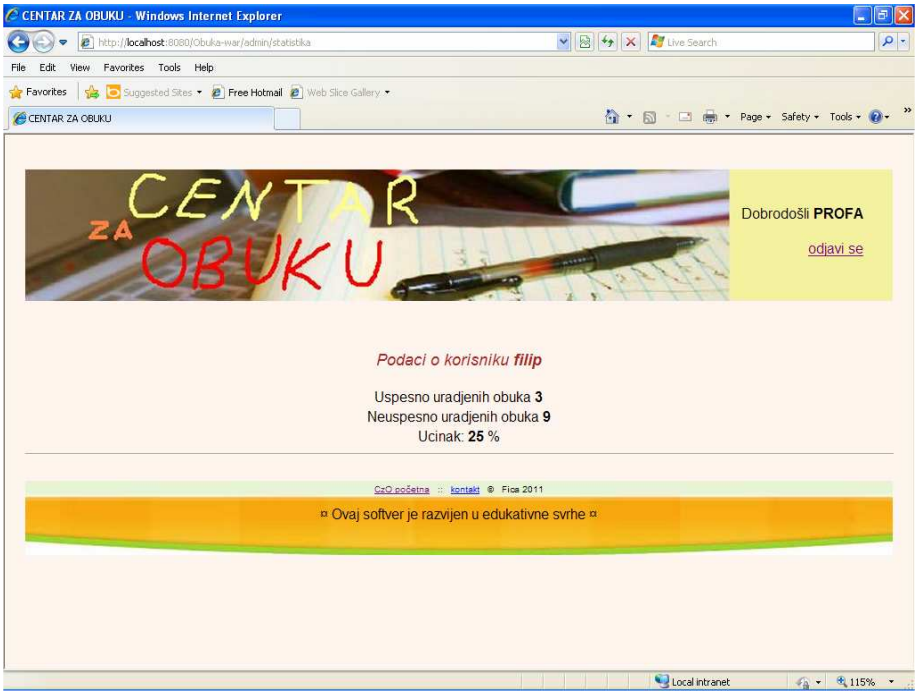
Slika 3.21 **Prikazivanje ispravnog redosleda komponenti**

Kada su u pitanju profesori, stvari izgledaju drugačije.
Ako se profesor uloguje na sistem, dobija informaciju koliko ima trenutno aktivnih korisnika (korisnika koji su u tom trenutku ulogovani) i koji su to korisnici (Slika 3.22)



Slika 3.22 **Pregled trenutno ulogovanih korisnika**

Profesor može odabrati korisnika čiju statistiku učinka želi da pogleda. Profesorovim klikom na dugme “Prikaži“ prikazuju se podaci o izabranom korisniku (Slika.3.23)



Slika 3.23 **Statistika učinka korisnika**

Sve korišćene JSP strane imaju zaglavlje (*header)* i podnožje (*footer)*. Zaglavlje sadrži odgovarajuće direktive za korišćene biblioteka etiketa. Zaglavlje i podnožje su smesteni u *jspf* folder WAR modula i definisani su u *web.xml* deskriptoru:

```
<jsp-config>
    <jsp-property-group>
        <include-prelude>/WEB-INF/jspf/header.jspf</include-
        prelude>
        <include-coda>/WEB-INF/jspf/footer.jspf</include-coda>
    </jsp-property-group>
</jsp-config>
```

Ilustracije radi navešćemo kod jedne manje JSP strane *prikaziObuke.jsp* na kojoj korisnik bira obuku od ponuđenih obuka koje su trenutno na raspolaganju u bazi. Može se primetiti korišćenje promenljive *obuke* sa dosegom u celoj aplikaciji kao i korišćenje JSP etikete *c:forEach* pomoću koje se prolazi kroz listu obuka promenljive *obuke*. Takođe može se uočiti da je CSS izdovojen u posebanu datoteku.

```
<div id="levaKolona">
    <div id="uputstvoTekst">
        Trenutan broj obuka na raspolaganju je
        <strong>${obuke.size()}</strong>
    </div>
    </br>
    <form name="biranjeObuke" action="izabranaObuka" method="post">
        <fieldset>
            <p>
                <legend>
                      Izaberite obuku po
                    &#382;elji:
                </legend>
            </p>
            <select name="odabranaObukaId">
                <c:forEach var="obuka" items="${obuke}">
                    <option value= "${obuka.id}">
                        ${obuka.imeObuke}
                    </option>
                    </br>
                </c:forEach>
            </select>
            <input type="submit" value="izaberi" class="dugme"/>
        </fieldset>
    </form>
</div>
```

3.9 Korišćena zrna

Sva Enterprise Java zrna nalaze se u EJB modulu Obuka-ejb u paketu ejb. Korišćena su sledeća zrna:

Zrna sesije bez stanja za klase entiteta:

- PokusajFacade.java
- ObukaFacade.java

Zrno sesije bez stanja:

- Mesanje.java

Zrno sesije jedinac (singleton):

- SingletonBroji.java

Zrno vođeno porukom:

- Poruka.java

Korišćena zrna sesije za klase entiteta su već opisana u poglavlju "3.7 Zrna sesije za klase entiteta".

Opisaćemo ukratko primenu preostalih korišćena zrna.

Mesanje.java

Zrno sesije bez stanja *Mesanje.java* korisiti se za mešanje komponenata obuke koju korisnik izabere i one treba da budu prikazane korisniku u slučajnom redosledu svaki put kada korisnik izabere neku obuku. Ako kasnije bude došlo do promene u poslovnoj logici, tj. pojavi se potreba da se komponente postavljaju ne u slučajnom nego u nekom drugom specijalnom poretku, potrebno je napraviti izmenu jedino u samom zrnu, a ostatak aplikacije ostaje zaštićen ne trpeći nikakve promene.

Za klasu *Mesanje.java* napravljen je lokalni interfejs *MesanjeLocal.java* pa u servlet klasu Kontrolor.java, koja se nalazi u WAR modulu u paketu *controller*, najpre vršimo ubrizgavanje zavisnosti:

```
@EJB
private MesanjeLocal mesanje;
```

Zatim pozivamo metode lokalnog interfejsa za postavljanje broja elementa niza komponenti izabrane obuke i pravljenje niza izmešanih indeksa komponenti:

```
mesanje.setBrojElemenata(broj_elemenata_izabrane_obuke);
int[] izmesanNiz = mesanje.getIzmesanNiz();
```

Po obavljenom poslu instanca zrna *Mesanje* nam ostaje na raspolaganju, tj. kontejner Enterprise Java zrna je može uposliti ponovo pri sledećem pozivu neke od metoda zrna.

SingletonBroji.java

Zrno sesije jedinac *SingletonBroji.java* koristimo za evidenciju o korisnicima koji su trenutno aktivni na sistemu. Ovo zrno omogućuje profesorima uvid u to koliko korisnika je trenutno aktivno i koji su to korisnici.
Da bi imali evidenciju o aktivnim korisnicima potrebno je beležiti kada se neki od korisnika uloguje ili izloguje sa sistema. U tu svrhu koristićemo Java kolekciju mapa koja će za ključ imati korisničko ime korisnika a za vrednost pridruženu ključu id sesiju korisnika.

```
private static Map<String, String> ulogovaniKorisnici = new
HashMap<String, String>();
```

Biće potrebno da kontrolišemo metode za kreiranje i uništavanje sesija pa naša klasa implementira *HttpSessionListener*

```
@Singleton
@LocalBean  // zrno je dostupno bez lokalnog interfejsa
@WebListener
public class SingletonBroji implements HttpSessionListener {... }
```

Pošto se svaki put, kada neki posetilac poseti početnu Veb stranicu, pravi nova sesija, mi za potrebe naše evidencije nećemo koristiti metodu *sessionCreated* HttpSessionListener-a, već ćemo, kada se korisnik uloguje iz servlet klase *Kontrolor.java*, pozvati metodu zrna SingletonBroji za upisivanje novog korisnika u mapu ulogovanih korisnika.

```
singletonBroji.upisiUlogovanogKorisnika(username, sessionId);
```

Moguće je da su neki korisnici ulogovani više puta koristeci istovremeno više Veb pretraživča pa pri upisivanju korisnika u mapu prvo proveravamo da li je dati korisnik već u mapi i ako nije onda ga upisujemo

```
public void upisiUlogovanogKorisnika(String username, String sessionId) {
        if (!ulogovaniKorisnici.containsKey(username)) {
        ulogovaniKorisnici.put(username, sessionId);
        }
}
```

Kada se korisnik izloguje, izbacujemo ga iz mape ulogovanih korisnika. To radimo koristeći njegov identifikator sesije na sledeći način:

```
@Override
public void sessionDestroyed(HttpSessionEvent se) {
```

```
        HttpSession sesija = (HttpSession) se.getSource();
         if (ulogovaniKorisnici.containsValue(sesija.getId())) {
        ulogovaniKorisnici.values().remove(sesija.getId());
         }
}
```

Kada se profesor uloguje, iz klase servleta *Kontrolor.java*, pozivamo metode zrna *SingletonBroji.java* koje vraćaju mapu trenutno ulogovanih korisnika i njihov broj:

```
int brojAktivnihKorisnika = singletonBroji.brojUlogovanihKorisnika();
Map<String, String> ulogovaniKorisnici =
singletonBroji.ulogovaniKorisnici();
```

Kada nema ulogovanih korisnika, metoda brojUlogovanihKorisnika vraća nulu a profesor se tada obaveštava da na sistemu trenutno nema aktivnih korisnika.

```
public int brojUlogovanihKorisnika() {
        return (ulogovaniKorisnici == null ? 0 : ulogovaniKorisnici.size());
 }
```

Pošto se klasa *singletonBroji.java* ne nalazi u WAR modulu gde je servlet klasa, već u EJB modulu, definisaćemo osluškivač u Veb deskriptoru *web.xml*:

```
<listener>
                <listener-class>ejb.SingletonBroji</listener-class>
 </listener>
```

Jednom napravljena instanca zrna *singletonBroji* postojaće sve do spuštanja ili eventualnog pada servera, tj. do zaustavljanja aplikacije na serveru.

Poruka.java

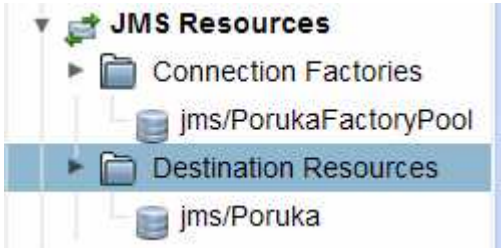
Zrno vođeno porukom *Poruka.java* koristimo za beleženje rezultata koje korisnik ostvari tokom obuke. Svaki put kada korisnik odradi neku od obuka, iz servlet klase *Kontrolor.java* pošalje se poruka sa podacima o rezultatu obavljene obuke. Pošto je u pitanju asinhrona komunikacija, nema blokiranja procesa, tj. servlet klasa ne čeka na zrno da izvrši potrebno upisvanje u bazu, već odmah nastavlja dalje omogućavajući korisniku da vidi rezultat obuke koju je obavio. Takođe, ukoliko kontejner Enterprise java zrna nema ni jednu slobodnu instancu zrna *Poruka.java* u trenutku slanja poruke, to neće uticati na servlet klasu. Ako primljena poruka nije odgovarajućeg formata, ili ako upisivanje u bazu bude neuspešno, beleži se upozorenje o tome, u za to predviđenu datoteku.

Klasa *Poruka.java* označena je anotacijom @MessageDriven koja govori kontejneru da je u pitanju zrno vođeno porukom kao i informacije o JMS konfiguraciji. Vrednost

promenljive mappedName generiše se od imena klase pa je u ovom slučaju to "jms/Poruka"

```
@MessageDriven(mappedName = "jms/Poruka", activationConfig = {
@ActivationConfigProperty(propertyName = "acknowledgeMode",
propertyValue = "Auto-acknowledge"),
@ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue")
})
public class Poruka implements MessageListener {...}
```

JMS resursi se prave na GlassFish serveru prilkom raspoređivanja aplikacije na server.



Slika 3.24 **JMS Resursi na GlassFish serveru**

JMS resursi su mapirani sa JNDI imenom destinacije odakle zrno dobija poruke pa u deskriptoru *glassfish-web.xml* imamo:

```
<message-destination-ref>
  <message-destination-ref-name>jms/Poruka</message-destination-ref-name>
      <jndi-name>jms/Poruka</jndi-name>
</message-destination-ref>
```

U servlet klasi *Kontrolor.java*, odakle šaljemo poruke, da bi koristili napravljene JMS resurse poslužićemo se @Resource anotacijom navodeći njima mapirana imena.

```
@Resource(mappedName = "jms/Poruka")
private Queue poruka;
@Resource(mappedName = "jms/PorukaFactory")
private ConnectionFactory porukaFactory;
```

Kada poruka stigne na destinaciju kontejner, poziva metodu zrna *onMessage* koja proverava da li je poruka odgovarajućeg formata i ako jeste vrši se upisivanje u bazu pomoću metode *sacuva.*

```
if (message instanceof ObjectMessage) {
        msg = (ObjectMessage) message;
        Pokusaj p = (Pokusaj) msg.getObject();
        sacuvaj(p);
}
```

Po obavljenom poslu instanca zrna *Poruka.java* ostaje na raspolaganju kontejneru Enterprise Java zrna spremna da, po potrebi, bude ponovo uposlena u slučaju primanja nove poruke.

4. Zaključak

Enterprise Java zrna imaju vrlo širok spektar primene u razvoju Java EE aplikacija tako da mogu imati mnogobrojne primene i u edukativne svrhe. Edukativne aplikacije često treba da budu skalabilne. Potrebno je vršiti čuvanje podataka u bazi pa korišćenje kontejnera Enterpise Java zrna za upravljanje transakcijama u mnogome olakšava posao. Kako su danas i sve češće u upotrebi manji uređaji, takozvani *lagani klijenti* sa kojih se pristupa Veb aplikacijama, može se reći da su, kada su u pitanju edukativne svrhe, sva tri kriterijuma iz poglavlja 2.2 za odlučivanje o korišćenju Enterprise Java zrna opravdana. U razvijenoj aplikaciji koja je opisana u ovom radu, kako bi se prikazala upotreba Enterprise Java zrna, odabrano je njihovo intenzivno korišćenje, ponekad na uštrb jednostavnosti aplikacije.

5. Kratak terminološki rečnik

bajtkod.....*bytecode*
deskriptor raspoređivanja..... .....*deployment descriptor*
doseg.....*scop*e
etiketa.....*tag*
forma za logovanje..... .....*form based authentication*
javna metoda.....*public method*
konačni.....*final*
lokalni.....*local*
prozorče za dijalog.....*dialog box*
raspoređivanje.....*deployment*
ubrizgavanja zavisnosti.....*dependence injection*
udaljeni.....*remote*
Zrna bez stanja.....*Stateless beans*
Zrna Jedinci.....*Singleton beans*r
Zrna sesije.....*Session bean*s
Zrna stanja.....*Stateful beans*
Zrna vođena porukom.....*Message-driven beans*

.

6. Literatura

[1] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, Chinmayee Srivathsa - „Java EE 6 Tutorial“ , Oracle, 2011

[2] Charles Lyons – „Sun Certified Web Component Developer (SCWCD) Study Companion, Second Edition with Java EE 6 Preview“, Garner Press UK, 2009

[3] www.netbeans.org - Java EE & Java Web Learning Trail

[4] Ian Evans - „Your First Cup: An Introduction to the Java EE Platform“, Oracle, 2011

[5] „Enterprise JavaBeans Specification Documentation 3.0 Final Release“, Oracle, 2010

[6] „GlassFish Server Open Source Edition 3.1 Quick Start Guide“, Oracle 2011

[7] Adam Bien – „EJB 3: From legacy technology to secret weapon“, JavaWorld.com, 2008

[8] David Flanagan – „*Java™ Foundation Classes in a Nutshell*“,

O'Reilly Media, 1999

7. Napomena

Kompletan izvorni kod razvijene aplikacije opisane u ovom radu priložen je uz sam rad.