UNIVERZITET METROPOLITAN BEOGRAD

CS230 - DISTRIBUIRANI SISTEMI

JMS i zrna vođena porukama

Lekcija 10

PRIRUČNIK ZA STUDENTE

CS230 - DISTRIBUIRANI SISTEMI

Lekcija 10

JMS I ZRNA VOĐENA PporukaMA

UVOD

JMS je strandardan Java API za upravljanje porukama.

Java Message Service (JMS) predstavlja standardni *Java EE API* (Application Programming Interface) za upravljanje porukama koji omogućava, u formi labave povezanosti komponenata, asinhronu komunikaciju između Java EE komponenata.

Savremena Java razvojna okruženja, a među njima je i *NetBeans IDE*, omogućavaju odličnu podršku koja je od velike pomoći za kreiranje Java EE aplikacija koje se baziraju na punoj prednosti primene *JMS API,* generišući većinu specifičnog JMS koda pri čemu se fokus programera ne rasipa i stavlja na razvoj poslovne logike konkretne aplikacije.

Za potpuno razumevanje principa i koncepata *JMS API,* koji će biti korišćeni prilikom razvoja Java EE aplikacija, u ovoj lekciji će teme biti pažljivo birane, analizirane i diskutovane. Posebna pažnja će biti stavljena na podršku izlaganju u formi pažljivo biranih i detaljno razrađenih i objašnjenih pokaznih primera.

U navedenom svetlu, lekcija će se posebno baviti sledećim temama:

- Uvod u JMS - biće predstavljen JMS API i dva domena JMS poruka;
- Kreiranje JMS resursa - biće detaljno obrađen koncept JMS destinacije. Dalje, biće započet projekat kao podrška izlaganju u lekciji i u okviru projekta biće pokazano kako se dodaje JMS destinacija u aplikativni server. Na kraju, ovog dela lekcije, biće prikazan način provere kreiranih JMS resursa u GlassFish veb konzoli;
- Implementacija JMS proizvođača - poseban akcenat će biti stavljen na kreiranje i implementaciju klasa modela i kontrolera JMS proizvođača. U ovom delu lekcije će biti kreirane i JSF stranice neophodne za testiranje aplikacije koja će biti kreirana;
- Korišćenje JSM poruka pomoću zrna vođenih porukama - ovaj deo lekcije će se baviti kreiranjem i implementacijom zrna vođenih porukama, kao sredstva za preuzimanje poruka sa prethodno kreirane JMS destinacije.

Savladavanjem ove lekcije student će biti osposobljen da u potpunosti razume i koristi *Java Messaging System* i *zrna vođena porukama* u Java EE aplikacijama. Takođe, student će ovladati dodatnim naprednim alatom za razvoj distribuiranih veb sistema.

Poglavlje 1

Uvod u JMS

JMS API

Primenom, JMS API, u savremenim razvojnim okruženjima, veoma je olakšan razvoj aplikacija.

Java Message Service (JMS) predstavlja standardni Java EE API (Application Programming Interface) za upravljanje porukama koji omogućava, u formi labave povezanosti komponenata, asinhronu komunikaciju između Java EE komponenata.

Veoma je važno, prilikom kreiranja Java EE veb distribuiranih aplikacija, koristiti svu moguću pomoć koji mogu da ponuda savremena Java razvojna okruženja. *NetBeans IDE,* kao jedno od njih, uključuje veoma moćnu podršku kao pomoć u kreiranju aplikacija koje koriste sve prednosti primene Java *Message Service (JMS) API* - ja. Kroz primenu ovakvog razvojnog okruženja, velika količina programskog koda može biti automatski generisana. Ovo je od posebnog značaja za programere. To praktično znači, da je njihov posao olakšan i rasterećen i da može biti fokusiran isključivo na razvoj poslovne logike umesto da se dodatno rasipa na razvoj JMS specifičnog koda.

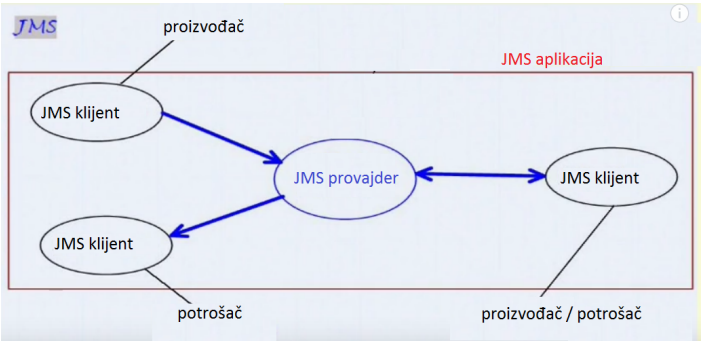
Posebno je važno istaći kako aplikacije crpe sve prednosti upotrebe *JMS API.* Aplikacije koje se oslanjaju na Java servis za upravljanjeporukama ne komuniciraju između sebe direktno. Umesto toga *proizvođači JMS poruka* (JMS message producers) šalju poruku na destinaciju (*redovi sa porukama* - JMS queue ili teme - topic) i *JMS potrošači* (JMS consumers) primaju poruke sa neke od destinacija.

Postoje dva domena poruka (messaging domains) koje je moguće koristiti kada se radi sa JMS porukama:

- point-to-point (PTP) messaging - U ovom slučaju JMS poruka se obrađuje od strane samo jednog JMS primaoca;
- publish/subscribe (pub/sub) messaging - U ovom slučaju svi primaoci poruka koji su registrovani za specifičnu temu (topic) primaju i obrađuju sve poruke koje su u vezi sa datom temom.

> JMS aplikacije koje koriste PTP način razmene poruka koriste redove sa porukama (JMS queue), dok JMS aplikacije koje koriste pub/sub način razmene poruka koriste teme (topic) kao JMS destinacije.

Sledećom slikom moguće je prikazati scenarije razmene poruka u JMS aplikaciji.



Slika 1.1 Razmena poruka u JMS aplikaciji [izvor: autor]

VIDEO MATERIJAL

Video materijal pod nazivom "Introduction to JMS"

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Kreiranje JMS resursa

JMS DESTINACIJA - UVODNA RAZMATRANJA

Pre razmene poruka neophodno je kreirati JMS destinaciju u aplikativnom serveru.

U ovom delu lekcije će biti govora o preduzimanju svih neophodnih aktivnosti koje prethode procesu razmene poruka između komponenata Java EE aplikacije. Ovde je posebno važno naglasiti da pre svakog započinjanja procesa razmene JMS poruka, neophodno je obaviti dodavanje JMS destinacije (JMS destinations) u aplikativni server koji se koristi za kreiranje i izvršavanje date Java EE aplikacije. Kada se koristi aplikativni server *GlassFish* u Java EE aplikacijama, ovaj zadatak je dodatno olakšan. Ovo praktično znači da je moguće kreirati JMS destinaciju direktno iz bilo kojeg Java EE projekta primenom nekog od savremenih Java razvojnih okruženja, kao što je na primer *NetBeans IDE*.

> Starije verzije Java EE platforme zahtevale su kreiranje JMS produkcije konekcije (JMS connection factory) kao dodatka JMS destinacijama. Sa Java EE 7 ovaj korak nije više potrebno izvoditi iz razloga što su svi podržani aplikativni serveri snabdeveni podrazumevanom JMS produkcijom konekcije.

U praksi, *JMS destinacija* predstavlja lokaciju posrednika na kojoj proizvođači poruka (JMS producers) postavljaju poruke i sa koje potrošači poruka (JMS consumers) preuzimaju postavljene poruke. Ukoliko se koristi PTP domen poruka kao JMS destinacija javljaju se redovi za poruke (message queues). Sa druge strane, ukoliko se koristi pub/sub domen poruka kao JMS destinacija javlja se tema poruka (message topic).

O redovima za poruke i temama moguće je proširiti analizu i zaokružiti izlaganje prilaganjem odgovarajućem video materijala.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

KREIRANJE JMS PROJEKTA

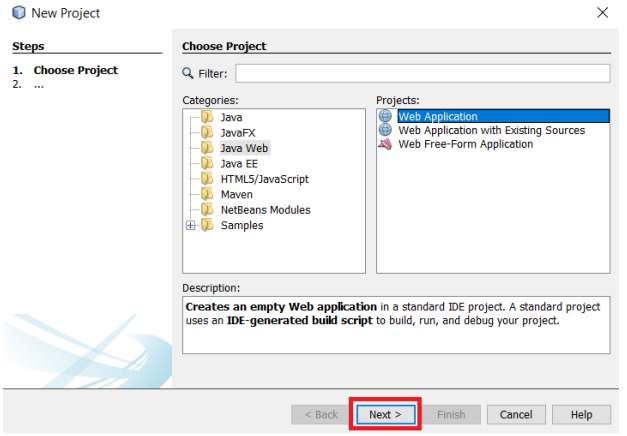
U ovom delu lekcije je akcenat na primeni savremenog razvojnog okruženja za kreiranje destinacije

U prethodnom izlaganju je, kao posebno važno, istaknuto da *JMS destinacija* predstavlja lokaciju posrednika na kojoj *proizvođači poruka (JMS producers)* postavljaju poruke i sa koje *potrošači poruka (JMS consumers)* preuzimaju postavljene poruke. U navedenom svetlu, neophodno je pokazati kako se kreira i koristi *JMS destinacija* u Java EE aplikacijama.

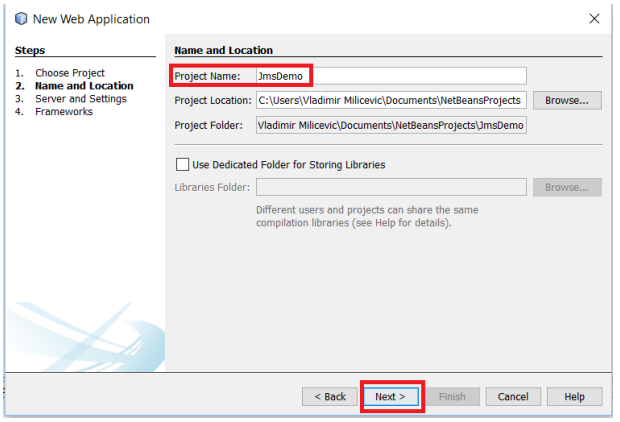
Posebno, kao i u svim dosadašnjim lekcijama, za kvalitetniju analizu i diskusiju, kao i za lakše razumevanje problematike *JMS poruka*, biće uveden pažljivo izabran primer kao podrška celokupnom izlaganju.

Posebno je bitno istaći da će za potrebe primera biti korišćen *PTP domen poruka.* To praktično znači, kao što je istaknuto u prethodnom izlaganju, da je neophodno kreirati *JMS red za poruke* (JMS queue). Procedura po kojoj se, u slučaju *pub/sub domena poruka,* kreiraju *teme poruka* (JMS topics) je skoro identična.

Dakle, neophodno je krenuti sa novim Java EE primerom kao podrškom za dalju analizu i diskusiju. Primer će sadržati projekat pod nazivom *JmsDemo* iz kategorije *Java Web / Web Applications*. Početni korak kreiranja JMS projekta je prikazan Slikom 1. Nakon početnog prozora, klikom na dugme *Next.* odlazi se na prozor gde se definiše naziv projekta *JMS aplikacije* koja se kreira. Navedeno je prikazano Slikom 2.



Slika 2.1 Kreiranje JMS projekta [izvor: autor]

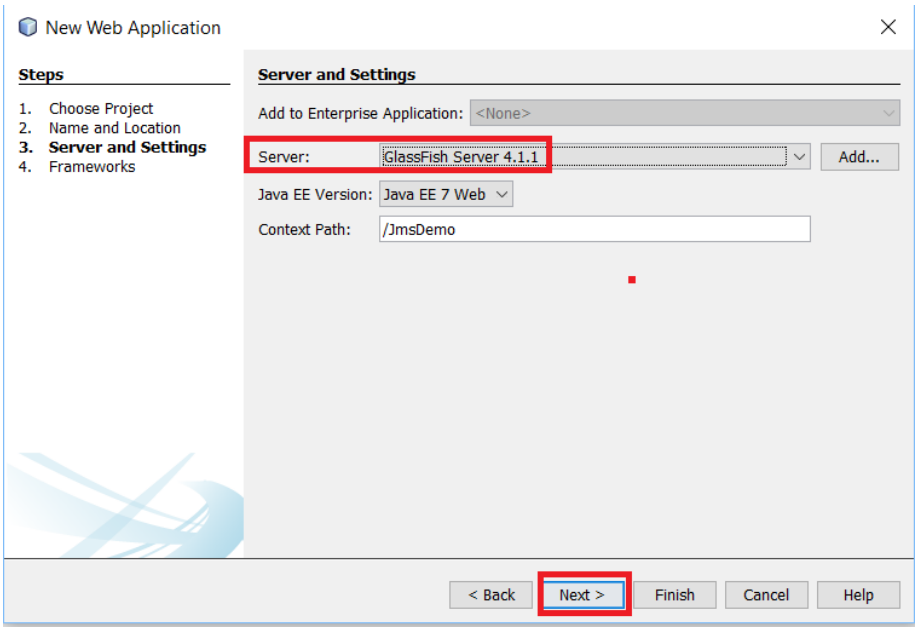


Slika 2.2 Dodela naziva JMS projektu [izvor: autor]

PODEŠAVANJA JMS PROJEKTA

Nophodno je, dalje, izvršiti sva podešavanja JMS projekta koji se kreira.

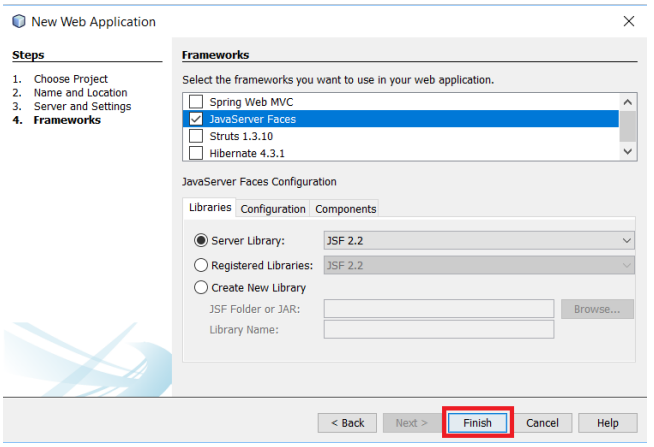
U nastavku je neophodno ispratiti čarobnjak razvojnog okruženja i obaviti dodatna podešavanja *JMS projekta* koji se kreira. Klikom na dugme *Next,* u prozoru sa Slike 2, otvara se prozor prikazan Slikom 3. Ovde se obavlja izbor aplikativnog servera za aplikaciju koja se kreira i to će u konkretnom slučaju biti aplikativni server *GlassFish 4.1.1*.



Slika 2.3 Izbor aplikativnog servera za JMS projekat [izvor: autor]

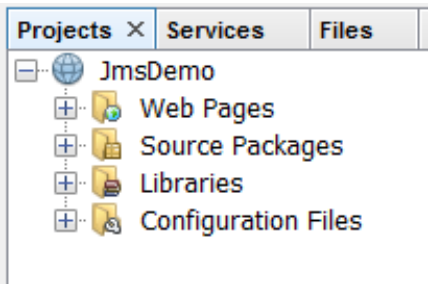
Klikom na dugme *Next,* otvara se prozor, prikazan Slikom 4, u kojem se obavljaju završna podešavanja ovog *JMS projekta*.

Kao poslednje podešavanje JMS projekta biće dodavanje *JavaServer Faces* okvira u projekat.



Slika 2.4 Izbor JSF okvira za projekat [izvor: autor]

Klikom na *Finish,* projekat je kreiran.

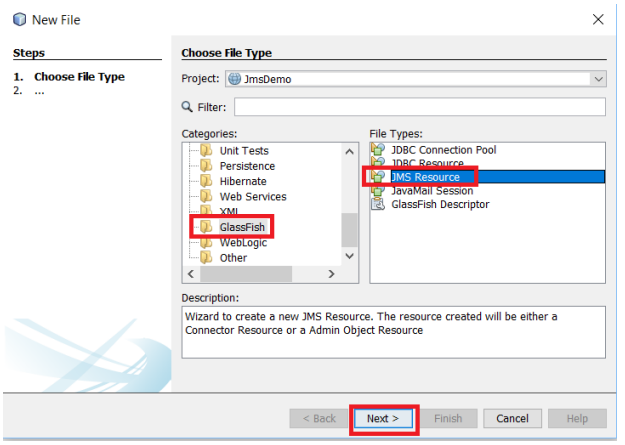


Slika 2.5 Inicijalna struktura JSF projekta - primera [izvor: autor]

DODAVANJE JMS DESTINACIJE ZA APLIKATIVNI SERVER

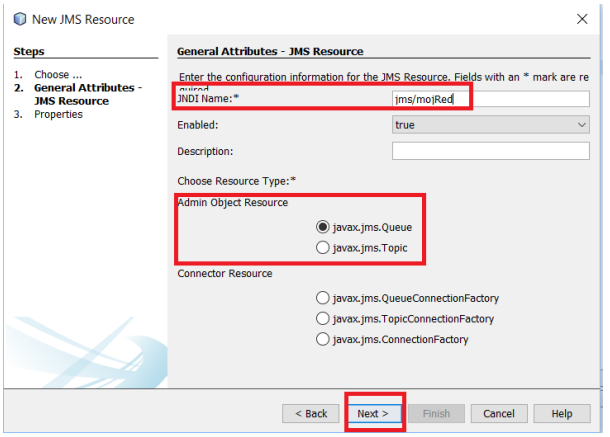
JMS destinacija, u konkretnom slučaju, biće red za poruke.

Pošto je *JMS projekat* kreiran, i inicijalno podešen, moguće je pristupiti dodavanju *JMS destinacije* u projekat. *JMS destinacija*, u konkretnom slučaju, biće *red za poruke* za aplikativni server. Da bi *red za poruke* bio kreiran, u aktuelnom projektu, neophodno je iz menija izabrati opciju *File | New File,* a nakon toga iz kategorije *GlassFish*, aplikativnog servera, izabrati tip datoteke *JMS Resource.* Prethodno izlaganje je ilustrovano sledećom slikom.



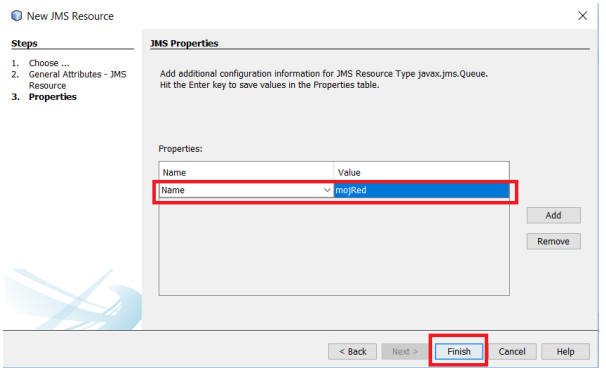
Slika 2.6 Početak kreiranja JMS resursa [izvor: autor]

U nastavku, nakon klika na dugme *Next,* otvara se prozor u kojem je neophodno uneti JNDI (*Java Naming and Directory Interface*) naziv reda za poruke koji se kreira. Naziv može biti *jms/mojRed*. u ovom prozoru se takođe bira i tip domena poruka, *javax.jms.Queue* u ovom slučaju. Prethodno navedeno je ilustrovano sledećom slikom.



Slika 2.7 JNDI naziv reda za poruke i tip domena poruka [izvor: autor]

Red za poruke zahteva osobinu *Name* koja se definiše u prozoru sa sledeće slike, koji je otvoren nakon klika na dugme *Next* prozora sa Slike 7.



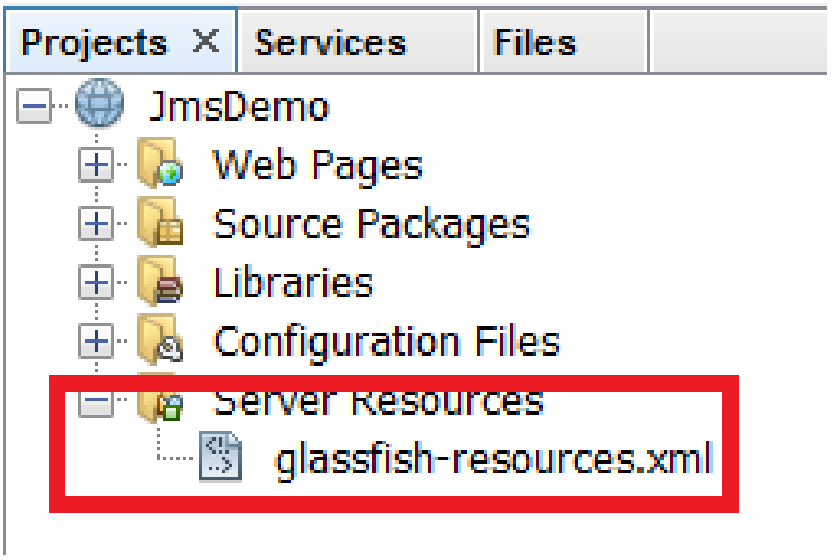
Slika 2.8 Definisanje osobine Name reda za poruke [izvor: autor]

DODATNA RAZMATRANJA

Nakon podešavanja kreiran je XML fajl resursa servera sa odgovarajućim kodom.

Klikom na dugme *Finish,* prozora prikazanog na Slici 8, završava se dodavanje reka za poruke u aplikativni server *GlassFish,* za kreirani JMS projekat. Pre toga je važno bilo napomenuti da je kao vrednost osobine *Name* uzet naziv kreiranog reda za poruke bez prefiksa *jms/*.

Nakon obavljenih zadataka, koji su bili diktirani čarobnjakom razvojnog okruženja, u ovom slučaju *NetBeans IDE,* kreiran je fajl pod nazivom *sun-resources.xml* koji se u projektu nazali u okviru čvora *Server Resources.* Navedeno je prikazano sledećom slikom.



Slika 2.9 Kreirani fajl pod nazivom sun-resources.xml [izvor: autor]

Kada se, u razvojnom okruženju *NetBeans IDE,* klikne na kreiranu datoteku, u editoru je moguće primetiti i XML kod koji je automatski generisan kada je završeno podešavanje koje se odnosilo na dodavanje reda za poruke *mojRed* u aplikativni server *GlassFish*. Generisani kod je priložen sledećim listingom.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <admin-object-resource enabled="true" jndi-name="jms/mojRed" object-type="user"
res-adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Name" value="mojRed"/>
  </admin-object-resource>
  </resources>
```

Kada se angažuje kreirani projekat, kroz *GlassFish* aplikativni server, vrši se učitavanje ovog fajla i kreiranje resursa definisanog u njemu.

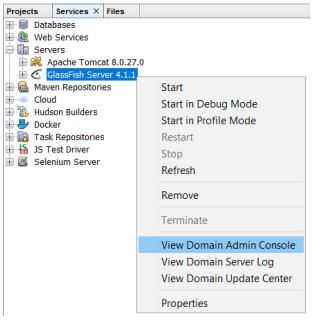
U nastavku, moguće je postupiti na način dobre prakse kada je u pitanju kreiranje i implementacija Java EE distribuiranih rešenja. To podrazumeva proveru da li su *JMS resursi* kreirani na ispravan način. Upravo će se sledeće izlaganje detaljno pozabaviti ovom problematikom.

PROVERA KREIRANIH JMS RESURSA

Dobra praksa je obavljanje provere nakon kreiranih JMS resursa.

Dobra praksa je obavljanje provere nakon kreiranih JMS resursa. Za potvrdu da je *red za poruke* uspešno kreiran neophodno je angažovati *GlassFish veb konzolu* (GlassFish web console) na sledeći način:

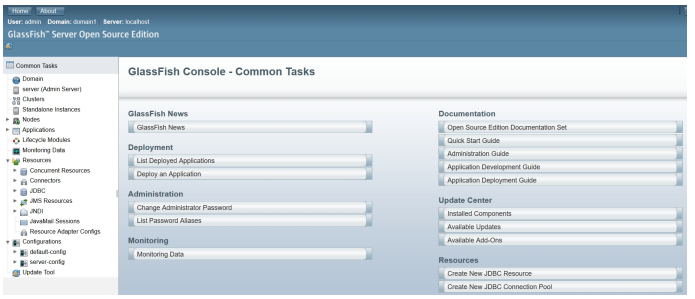
- U razvojnom okruženju *NetBeans IDE*, za kreirani JMS projekat, neophodno je kliknuti na tab *Services*;
- Dalje, neophodno je izlistati sadržaj čvora *Servers* (klikom na "+");
- U sledećem koraku je neophodno desnim klikom miša izabrati opciju *GlassFish* 4.1.1;
- Konačno, iz ponuđenog menija opcija bira se *View Domain Admin Console* (videti sledeću sliku).



Slika 2.10 Izbor View Domain Admin Console opcije [izvor: autor]

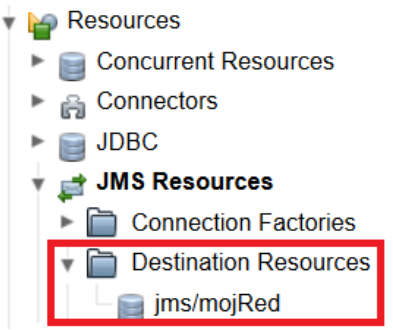
Da bi ova opcija bila dostupna, neophodno je pokrenuti GlassFish server prethodno.

Veoma brzo u podrazumevanom veb pregledaču će biti učitana *GlassFish konzola,* veb bazirana administrativna konzola je prikazana Slikom 11.



Slika 2.11 GlassFish konzola [izvor: autor]

Desnim klikom na kreirani projekat i izborom opcije *Deploy,* projekat se povezuje sa aplikativnim serverom i kreirani resursi su dostupni u serveru. U *GlassFish konzoli* je to moguće uočiti proširivanjem čvora *JMS Resources,* a zatim i njegovog podčvora *Destination Resources.* Ukoliko tamo postoji kreirani red *jms/mojRed* (viseti sledeću sliku), posao kreiranja JMS resursa za aplikaciju je uspešno obavljen i to je provereno.



Slika 2.12 Kreirani red za poruke u aplikativnom serveru GlassFish [izvor: autor]

Poglavlje 3

Implementacija JMS proizvođača

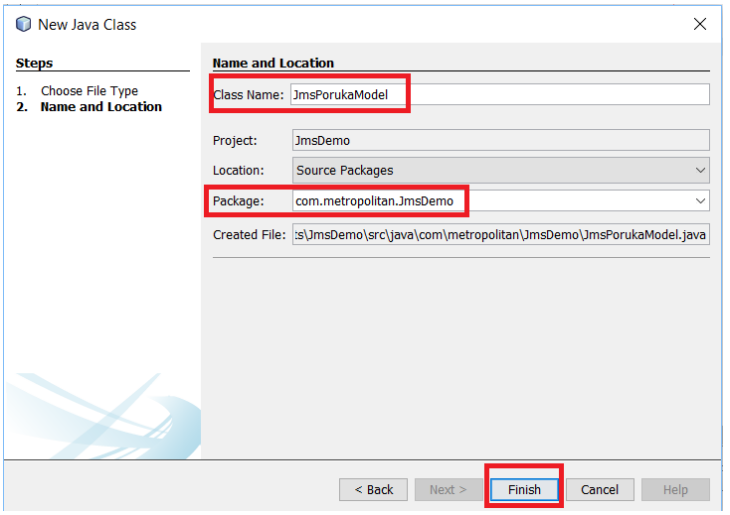
MODEL JMS PROIZVOĐAČA

U prvom koraku će biti kreiranja Java klasa modela JMS proizvođača.

Pošto je kreiran red za poruke ispunjeni su svi uslovi za kreiranje aktera procesa razmene poruka. U tu svrhu kreirani projekat će, po principu korak - po - korak, dobijati nove datoteke koje će sa sobom donositi nove funkcionalnosti u aplikaciju. Aplikacija, koja služi za analizu i diskusiju, biće u daljem radu razvijana kao *JSF* aplikacija. Jedno od *CDI upravljanih zrna* aplikacije će funkcionisati kao *proizvođač JMS poruka.* Ovo zrno će imati zadatak da kreirane *JMS poruke* šalje u *kreirani red za poruke,* upravo onaj resurs koji je podešen na način izložen u prethodnom izlaganju.

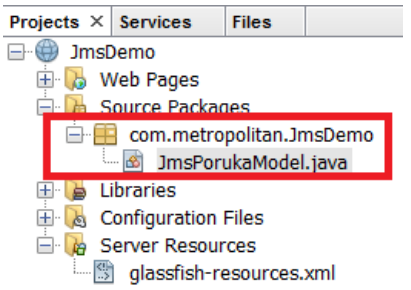
Na osnovu prethodnog izlaganja, moguće je sada početi sa zacrtanim proširenjem kreiranog projekta *JmsDemo.* Za početak će biti kreirana Java klasa pod nazivom *JmsPorukaModel.java.* Ova klasa će imati zadatak da čuva tekst poruke koja će biti *prosleđena u red* za poruke.

Klasa će biti kreirana, na dobro poznat način, kao standardna Java klasa kreiranog projekta, izborom opcije *File | New File,* u razvojnom okruženju *NetBeans IDE*, nakon čega će biti otvoren prozor pod nazivom *New Java Class* u kojem je neophodno dodeliti naziv klasi (*JmsPorukaModel*) i naziv paketu u kojem će se čuvati kreirana klasa (ovom slučaju *com.metropolitan.JmsDemo*). Navedeno je ilustrovano sledećom slikom.



Slika 3.1 Kreiranje klase model JMS proizvođača [izvor: autor]

Klikom na *Finish* kreirana je tražena Java klasa sa inicijalnim kodom.



Slika 3.2 Datoteka klase modela JMS proizvođača u projektu [izvor: autor]

KODIRANJE KLASE MODELA JMS PROIZVOĐAČA

Kreiranu datoteku modela JMS proizvođača neophodno je konkretizovati odgovarajućim kodom.

U prethodnom izlaganju je kreirana Java datoteka pod nazivom *JmsPorukaModel.java* koja bi trebalo da ima zadatak čuvanje JMS poruke koja će, kada na to bude došao red, biti prosleđena u kreirani *red za poruke*. Ova klasa, u ovom trenutku, poseduje inicijalni kod, a to znači da je prazna i da još ne predstavlja *CDI zrno.*

Da bi kreiranja klasa postala *CDI zrno*, prvo što je neophodno učiniti jeste njeno obeležavanje anotacijom ***@Named.*** Takođe, odmah nakon što je ovaj zadatak obavljen, klasa će biti obeležena novom anotacijom. Anotacija ***@RequestScoped*** će klasi dodeliti pripadnost oblasti zahteva (o ovome je detaljno bilo govora u prethodnim lekcijama).

Nakon obeležavanja odgovarajućim anotacijama, klasa je sada postala *CDI zrno* koje se odnosi na oblast zahteva. Ovo zrno je sada neophodno kompletirati dodajući mu odgovarajuću logiku koju mora da implementira. Prvo će biti kreirana ***String*** varijabla pod nazivom *msgText.* Varijabla je privatna i nakon njenog inicijalizovanja u klasu će biti neophodno dodati i odgovarajuće setter i getter metode.

Takođe, da bi klasa funkcionisala na pravi način, odnosno da bi pomenute anotacije bile upotrebljive, neophodno je uvesti i odgovarajuće *import* instrukcije u datoteku klase *JmsPorukaModel.java.*

Kada je sve obavljeno na gore pomenuti način, klasa *JmsPorukaModel.java* ima oblik priložen sledećim listingom.

```
package com.metropolitan.JmsDemo;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JmsPorukaModel {
```

```
    private String msgText;

    public String getMsgText() {
        return msgText;
    }

    public void setMsgText(String msgText) {
        this.msgText = msgText;
    }


}
```

Pošto je uspešno kodirana klasa *modela JMS proizvođača,* moguće je krenuti dalje u razvoj aplikacije. To znači da je za kreirani model neophodno razviti klasu kontrolera i upravo će na tome biti akcenat u narednom izlaganju.

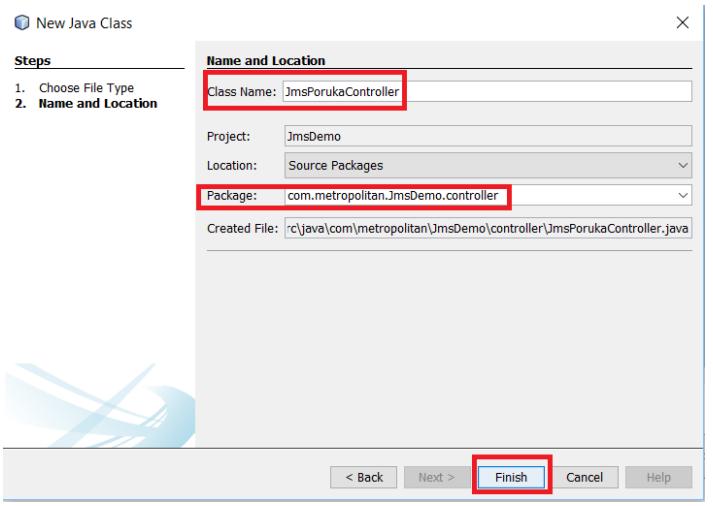
KONTROLER JMS PROIZVOĐAČA

U nastavku će pažnja biti usmerana na kreiranje klase kontrolera JMS proizvođača.

Kao što je istaknuto u prethodnom izlaganju, klasa *modela JMS proizvođača* je uspešno kodirana i moguće je krenuti dalje u razvoj aplikacije. To znači da je za kreirani model neophodno razviti klasu kontrolera i upravo će na tome biti akcenat u narednom izlaganju.

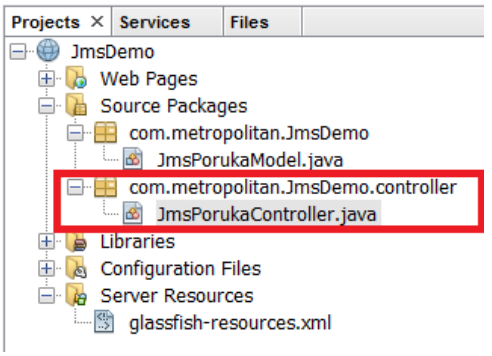
Pošto će u ovom delu lekcije sva pažnja biti usmerena ka *kontroleru JMS proizvođača,* neophodno je prvo istaći šta će biti njegov konkretan zadatak. Kontroler će, zapravo, vršiti slanje *JMS poruka* u *red za poruke* koji je kreiran na način objašnjen i prikazan u nekom od prethodnih izlaganja.

Sada je moguće pristupiti kreiranju datoteke *kontrolera JMS proizvođača.* Klasa će biti kreirana, na dobro poznat način, kao standardna Java klasa kreiranog projekta, izborom opcije *File | New File*, u razvojnom okruženju *NetBeans IDE*, nakon čega će biti otvoren prozor pod nazivom *New Java Class* u kojem je neophodno dodeliti naziv klasi *kontrolera JMS proizvođača* (*JmsPorukaController*) i naziv paketu u kojem će se čuvati kreirana klasa (ovom slučaju *com.metropolitan.JmsDemo.controller*). Navedeno je ilustrovano sledećom slikom.



Slika 3.3 Kreiranje datoteke klase kontrolera JMS proizvođača [izvor: autor]

Klikom na *Finish* kreirana je tražena Java klasa sa inicijalnim kodom.



Slika 3.4 Datoteka klase kontrolera JMS proizvođača u projektu [izvor: autor]

GENERISANJE KODA ZA SLANJE JMS PORUKA

Kreiranu datoteku kontrolera JMS proizvođača neophodno je konkretizovati odgovarajućim kodom.

Kao što je istaknuto u prethodnom izlaganju, u ovom delu lekcije sva pažnja biti usmerena ka *kontroleru JMS proizvođača* koji će, zapravo, vršiti slanje JMS poruka u *red za poruke* koji je kreiran na način objašnjen i prikazan u nekom od prethodnih izlaganja. Da bi kontroler postao to što se od njega očekuje, kreirana klasa *JmsPorukaController.java* mora da bude proširena konkretnom logikom. Za početak klasa će morati da bude obeležena anotacijama ***@Named*** i ***@RequestScoped.*** Ovim anotacijama klasa je obeležena kao *CDI zrno* koje se odnosi na *oblast zahteva*. Da bi anotacije mogle da budu korišćene, neophodno je dodati i odgovarajuće ***import*** instrukcije u kod klase *kontrolera JMS proizvođača*.

Nakon obavljenog obeležavanja klase navedenim anotacijama, klasa *kontrolera JMS proizvođača* dobija sledeći privremeni oblik:

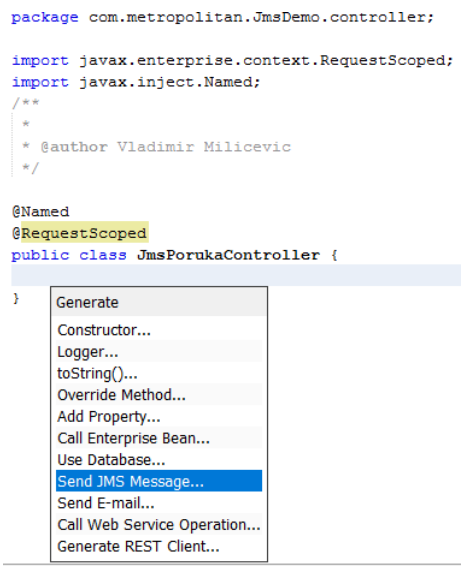
```
package com.metropolitan.JmsDemo.controller;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
/**
 *
 * @author Vladimir Milicevic
 */

@Named
@RequestScoped
public class JmsPorukaController {

}
```

U nastavku je neophodno konkretizovati ovu klasu dodavanjem logike koja se odnosi na zadatak slanja *JMS poruka*. Ponovo je moguće istaći veliki značaj savremenih razvojnih okruženja, a *NetBeans IDE* je jedno od njih, u razvoju savremenih Java aplikacija. U ovom slučaju, razvojno okruženje *NetBeans IDE* može biti od velike pomoći koja se ogleda kroz automatizovanje procesa kodiranja koda za *slanje JMS poruka*. Klikom na kombinaciju tastera *Alt + Insert* otvara se prozor pod nazivom *Generate* iz kojeg je neophodno izabrati opciju *Send JMS Message.* Navedena aktivnost može biti ilustrovana sledećom slikom.



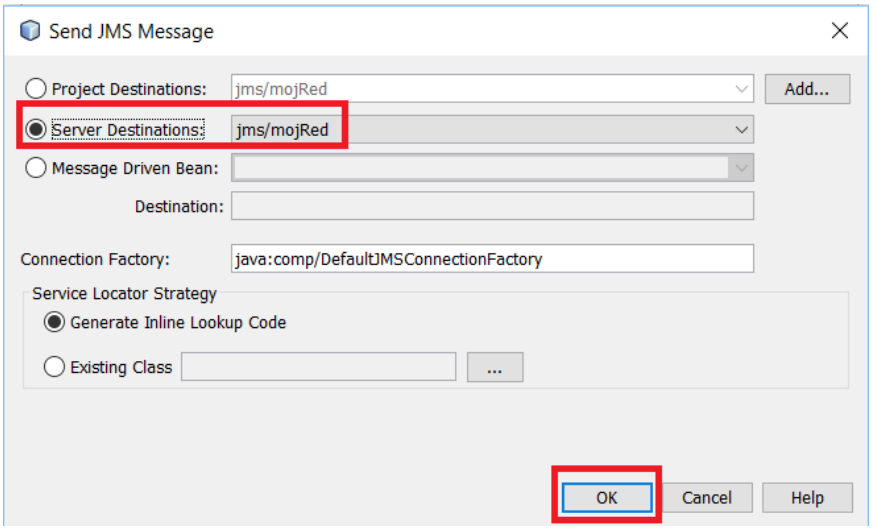
Slika 3.5 Generisanje koda za slanje JMS poruka [izvor: autor]

GENERISANJE KODA ZA SLANJE JMS PORUKA - DODATNA RAZMATRANJA

Razvojno okruženje je od velike pomoći i omogućava automatsko generisanje koda.

Kao što je moguće primetiti na priloženoj Slici 6, za generisanje koda klase *kontrolera JMS proizvođača,* zaduženog za slanje *JMS poruka*, neophodno izabrati opciju *Send JMS Message*.

Nakon izbora ove opcije otvara se prozor pod nazivom *Send JMS Message* koji je ilustrovan sledećom slikom.



Slika 3.6 Prozor Send JMS Message [izvor: autor]

Ono što je moguće primetiti sa slike jeste da je neophodno čekirati odgovarajuće *radio dugme* pod nazivom *Server Destinations* i iz odgovarajućeg padajućeg menija izabrati red za poruke koji je kreiran u prethodno prikazanom izlaganju. *Red za poruke* je, u konkretnom slučaju, odrešen nazivom *jms/mojRed*.

Kada je obavljen zadatak, koji je upravo opisan, neophodno je jednostavno kliknuti na dugme *OK,* prozora *Send JMS Message,* i u klasu *kontrolera JMS proizvođača* automatski će biti prosleđen kod za slanje *JMS poruka.* Navedena klasa sada ima oblik koji odgovara sledećem listingu.

```
package com.metropolitan.JmsDemo.controller;

import javax.annotation.Resource;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.jms.JMSConnectionFactory;
import javax.jms.JMSContext;
import javax.jms.Queue;
/**
 *
 * @author Vladimir Milicevic
 */

@Named
@RequestScoped
public class JmsPorukaController {

    @Resource(mappedName = "jms/mojRed")
    private Queue mojRed;

    @Inject
```

```
    @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory")
    private JMSContext context;

    private void sendJMSMessageToMojRed(String messageData) {
        context.createProducer().send(mojRed, messageData);
    }

}
```

ANALIZA GENERISANOG KODA ZA SLANJE JMS PORUKA

Sledi analiza automatski generisanog koda za slanje JMS poruka klase kontrolera JMS proizvođača.

U prethodnom izlaganju je prikazan listing koda klase *kontrolera JMS proizvođača* u koji je pomoću razvojnog okruženja *NetBeans IDE*, automatski ugrađen kod za upravljanje *JMS porukama*. U narednom izlaganju, sledi analiza automatski generisanog koda za slanje *JMS poruka* klase *kontrolera JMS proizvođača*.

Prvo što je moguće primetiti jeste da je generisana privatna varijabla pod nazivom *mojRed,* tipa *javax.jms.Queue.* Ova varijabla je obeležena anotacijom *@Resource(mappedName = "jms/mojRed")* koja ukazuje na povezanost varijable sa *redom za poruke jms/mojRed* kreiranim na način koji je obrađen i prezentovan u nekom od prethodnih izlaganja.

Takođe, moguće je primetiti da je u kodu generisana i privatna varijabla pod nazivom *context*, tipa *JMSContext.* Ova varijabla je obeležena anotacijom ***@Inject,*** a to rezultuje, u aplikativnom serveru *GlassFish,* umetanjem instance tipa *JMSContext* u ovu varijablu, tokom vremena izvršavanja (runtime). Varijabla *context* je takođe obeležena anotacijom ***@JMSConnectionFactory*** koja povezuje ovu varijablu sa produkcijom JMS konekcije (JMS connection factory).

> U prethodnim verzijama Java EE platforme bilo je neophodno kreirati produkciju JMS konekcijekao dodatak za JMS destinaciju. Java EE 7 uvodi podrazumevanu produkciju JMS konekcije koju je moguće koristiti u JMS kodu. Razvojno okruženje NetBeans u potpunosti koristi navedene prednosti najnovije Java EE platforme.

Konačno, u programski kod klase *kontrolera JMS proizvođača* ugrađena je automatski i metoda čiji je kod izolovan sledećim listingom:

```
private void sendJMSMessageToMojRed(String messageData) {
        context.createProducer().send(mojRed, messageData);
    }
```

Ova metoda zapravo i obavlja zadatak slanja *JMS poruke* u *red za poruke*. Njen naziv direktno zavisi od naziva kreiranog reda za slanje poruka. Pošto je naziv reda za poruke

mojRed, prilikom generisanja posmatrane metode, automatski joj je dodeljen naziv *sendJMSMessageToMojRed.*

Generisana metoda koristi pojednostavljen *JMS 2.0 API* koji je dodat u platformu Java EE 7. U telu metode, *context* objekat poziva metodu ***createProducer()*** koja vraća instancu klase *javax.jms.JMSProducer.* Nakon toga, vraćeni objekat tipa *JMSProducer* poziva metodu ***send().*** Prvi parametar ove metode predstavlja *JMS destinaciju* na koju će poruka biti poslata, u konkretnom slučaju to je *kreirani red za poruke mojRed*. Drugi parametar metode predstavlja ***String*** objekta koji je zapravo poruka koju bi trebalo proslediti.

Postoji nekoliko tipova JMS poruka koje je moguće poslati u JMS red za poruke. Najčešći tip poruke je *javax.jms.TextMessage.* U prethodnim verzijama *JMS API*, bilo je neophodno eksplicitno koristiti ovaj interfejs za slanje poruka koje sadrže jednostavan ***String***. Novi *JMS 2.0 API* kreira instancu klase koja implementira ovaj interfejs "iza scene" kada šalje ***String*** kao poruku i značajno olakšava posao programerima.

DODATNO KODIRANJE KODA KONTROLERA

Automatski generisan kod kontrolera je moguće dopuniti porukom koju je neophodno proslediti.

Kreiranu klasu *kontrolera JMS proizvođača* je neophodno dopuniti sa nekoliko sitnih modifikacija koje se pre svega odnose na poruku koju bi trebalo poslati. Sledećim listingom priložen je dopunjeni kod klase *JmsPorukaController.*

```
package com.metropolitan.JmsDemo.controller;

import com.metropolitan.JmsDemo.JmsPorukaModel;
import javax.annotation.Resource;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.jms.JMSConnectionFactory;
import javax.jms.JMSContext;
import javax.jms.Queue;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JmsPorukaController {

    @Inject
    private JmsPorukaModel jmsPorukaModel;

    @Resource(mappedName = "jms/mojRed")
    private Queue mojRed;
```

```
    @Inject
    @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory")
    private JMSContext context;

    public String sendMsg() {
        sendJMSMessageToMojRed(jmsPorukaModel.getMsgText());
        return "potvrda";
    }

    private void sendJMSMessageToMojRed(String messageData) {
        context.createProducer().send(mojRed, messageData);
    }

}
```

Iz priloženog i dopunjenog listinga klase *kontrolera JMS proizvođača* moguće je primetiti da sve što je bilo neophodno da se uradi jeste umetanje instance prethodno kreirane klase *modela JMS proizvođača JmsPorukaModel,* kao i dodavanje jednostavne metode *sendMsg()* u okviru koje će biti izvršeno pozivanje metode *sendJMSMessageToMojRed(),* prosleđujući tekstualnu poruku kao parametar.

STRANICA INDEX.XHTML

U nastavku je neophodno baviti se datotekama JSF stranica projekta.

Da bi korisnik aplikacije mogao da interaguje sa aplikacijom koja se upravo kreira, u nastavku je neophodno baviti se datotekama JSF stranica projekta.

Prvo će biti modifikovana početna stranica *index,xhtml*. Na stranicu je neophodno dodati formu povezujući *msgText* varijablu klase *JmsPorukaModel* sa poljem za unos teksta i komandnim dugmetom *Posalji* koje poziva metodu *sendMsg().* Kod modifikovane JSF stranice *index.xhtml* je priložen sledećim listingom.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Slanje JMS poruke</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="msgText" value="Unesite tekst poruke:"/>
                <h:inputText id="msgText"
                             value="#{jmsPorukaModel.msgText}"/>
                <h:panelGroup/>
                <h:commandButton value="Posalji"
                                 action="#{jmsPorukaController.sendMsg()}"/>
```

```
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

Iz priloženog koda je moguće primetiti da je upotrebljen *<h:inputText>* tag za povezivanje ulaza korisnika sa *msgText* varijablom klase *JmsPorukaModel* pomoću varijable ***value,*** Unificiranog jezika izraza (Unified Expression Language), čija je vrednost *#{jmsMessageModel.msgText}.*

Ako se u kodu obrati pažnja na ***action*** atribut komandnog dugmeta *Posalji,* moguće je primetiti da njegov tag *<h:commandButton>* prosleđuje kontrolu metodi *sendMsg()* klase *JmsPorukaController* kada korisnik klikne na dugme.

Ponovo je potrebno istaći da *JmsMessageController.sendMsg()* preuzima vrednost *JmsMessageModel.msgText* i prosleđuje je u *red za poruke*. Takođe, *JmsMessageController.sendMsg()* preusmerava korisnika na jednostavnu stranicu za potvrdu unosa o kojoj će biti reči u sledećem izlaganju.

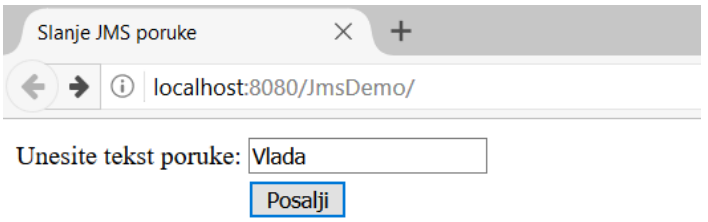
STRANICA POTVRDA.XHTML

Projektu nedostaje stranica za potvrdu slanja JMS poruke.

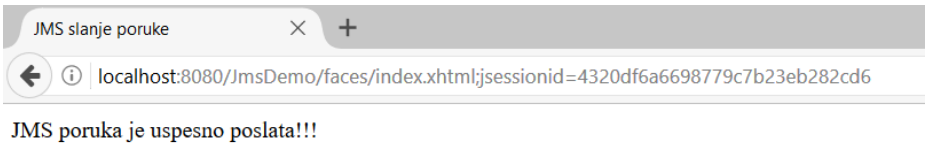
U prethodnom izlaganju je istaknuto da *JmsMessageController.sendMsg()* preusmerava korisnika na jednostavnu stranicu za potvrdu unosa. Stranica se naziva *potvrda.xhtml* i njen kod je priložen sledećim listingom.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>JMS slanje poruke</title>
    </h:head>
    <h:body>
        JMS poruka je uspesno poslata!!!
    </h:body>
</html>
```

Kreiranja JSF stranica je veoma jednostavna i ona jednostavno, u veb pregledaču, prikazuje informaciju da je JMS poruka uspešno poslata. Slikama 7 i 8 su prikazane stranice *index.xhtml* i *potvrda.xhtml* respektivno.



Slika 3.7 Stranice index.xhtml [izvor: autor]



Slika 3.8 Stranica potvrda.xhtml [izvor: autor]

Za sada su objašnjeni i demonstrirani mehanizmi produkcije poruka i njihovog prosleđivanja u *redove za poruke*. U narednom izlaganju će biti neophodno baviti se analizom i diskusijom u vezi sa razvojem koda koji će imati sposobnost preuzimanja poruka koje su postavljene u *red za poruke*.

VIDEO MATERIJAL

JMS 01 Post to a Queue with Glassfish + Netbeans

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 4

Korišćenje JMS poruka pomoću zrna vođenih poruka

ZRNA VOĐENA PORUKAMA

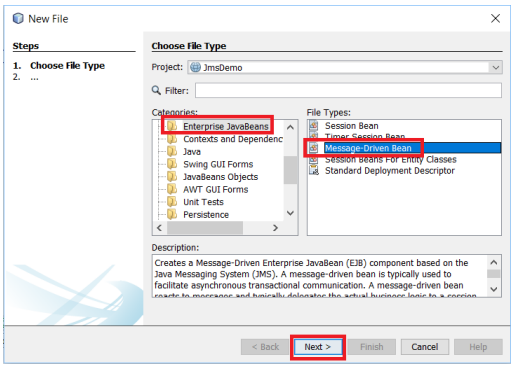
Najpogodniji način za implementiranje JMS potrošača poruka jeste razvoj zrna vođenih porukama.

Prethodno izlaganje se baziralo na *proizvođačima poruka* i postavljanjem poruka na *red za poruke*. U narednom izlaganju će biti neophodno baviti se analizom i diskusijom u vezi sa razvojem koda koji će imati sposobnost preuzimanja poruka koje su postavljene u red za poruke.

Najpogodniji način za implementiranje JMS potrošača poruka jeste razvoj i primena zrna vođenih porukama (message-driven beans). Zrna vođena porukama su posebna vrsta EJB (Enterprise JavaBean) zrna čija je namena osluškivanje *JMS poruka* postavljenih u *red za poruke* ili *temu*.

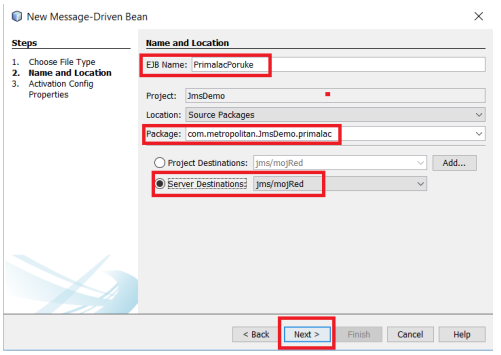
> U realnim sistemima, JMS proizvođači i JMS potrošači se razvijaju u odvojenim projektima budući da oni, zapravo, i predstavljaju odvojene sisteme. Zbog jednostavnosti i lakšeg kreiranja primera, koji ilustruje ovu problematiku, ovi koncepti će biti razvijeni u okviru istog, aktuelnog, projekta.

Kao što je postavljeno kao praksa u ovim materijalima, svako izlaganje, pa i ovo, praćeno je podrškom adekvatnog primera. U navedenom svetlu, vrši se proširenje primera *JmsDemo* u kojeg bi trebalo da budu uvedene funkcionalnosti karakteristične za *JMS potrošače poruka*. Za početak, biće pokazano kako se u, aktuelnom projektu, kreira *zrno vođeno porukama*. U razvojnom okruženju, *NetBeans IDE,* za projekat *JmsDemo*, bira se opcija *File | New File,* a zatim se, u konkretnom prozoru, iz kategorije *Enterprise JavaBeans* bira kreiranje datoteke tipa *Message-Driven Bean.* Navedeno izlaganje je ilustrovano sledećom slikom.



Slika 4.1 Početak kreiranja datoteke zrna vođenog porukama [izvor: autor]

U nastavku je neophodno, nakon klika na dugme *Next* i otvaranja novog prozora, dati naziv klasi *zrna vođenog porukama*, definisati paket kojem će kreirana klasa pripadati i čekirati opciju *Server Destinations* gde je izabran kreirani red za poruke *jms/MojRed* . Navedeno je prikazano Slikom 2.

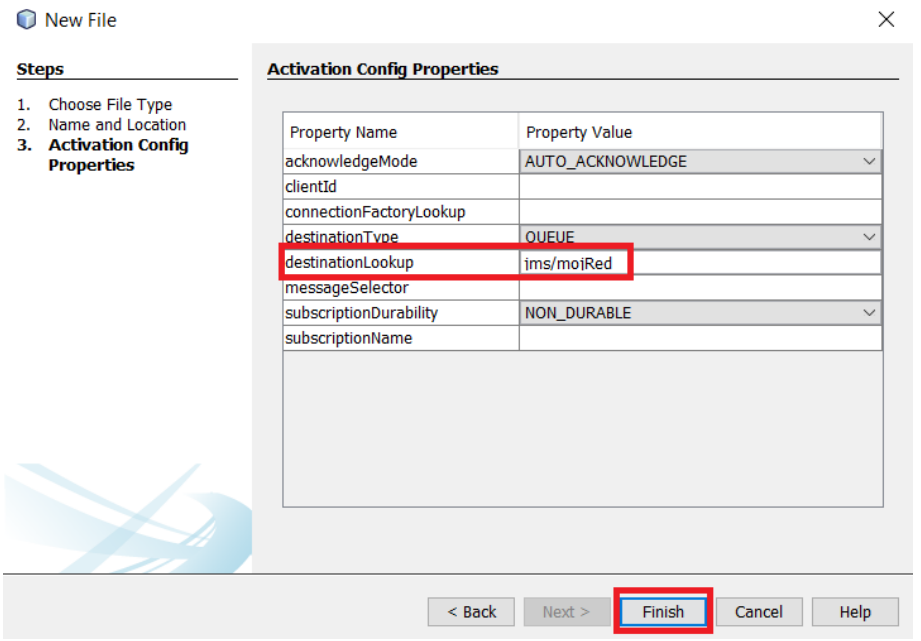


Slika 4.2 Osnovna podešavanja zrna vođenog porukama [izvor: autor]

PODEŠAVANJE ZRNA VOĐENOG PORUKAMA

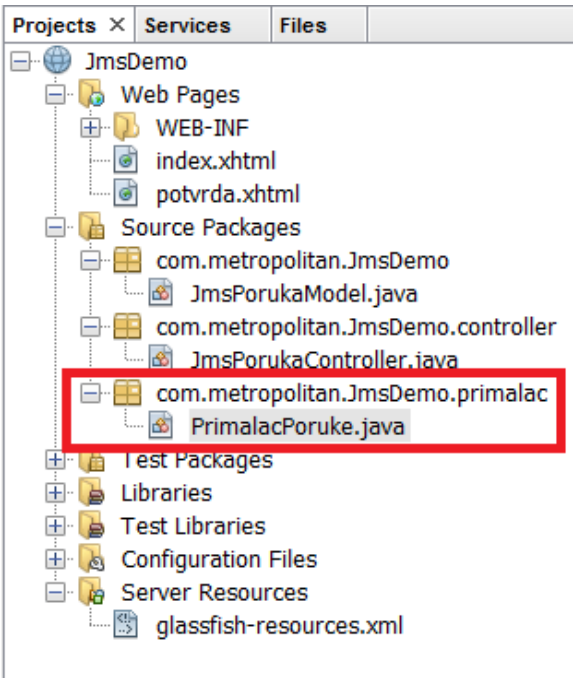
Neophodno je izabrati nekoliko osobina za dodatno podešavanje zrna vođenog porukama.

Ako se pogleda prozor, priložen prethodnom slikom, nakon obavljenih osnovnih podešavanja vrši se klik mišem na dugme *Next.* Tada se otvara nov prozor, pod nazivom *New File / Activation Config Properties,* u okviru kojeg je moguće izabrati nekoliko osobina značajnih za dodatno podešavanje *zrna vođenog porukama* koje se kreira. Navedeni prozor je ilustrovan sledećom slikom.



Slika 4.3 Dodatno podešavanje zrna vođenog porukama [izvor: autor]

Iz priložene slike je moguće primetiti da je kao destinacija sa porukama, videti osobinu *destinationLookup,* izabran kreirani red sa porukama *jms/mojRed,* a da je kao odgovarajući tip, videti osobinu *destinationType,* istaknuto *QUEUE,* odnosno red. Klikom na *Finish,* završava se kreiranje ove datoteke i ona je zauzela svoje mesto u strukturi projekta (sledeća slika).



Slika 4.4 Datoteka zrna vođenog porukama u strukturi projekta [izvor: autor]

OSOBINE ZRNA VOĐENOG PORUKAMA

Posebno je potrebno pozabaviti se osobinama zrna vođenog porukama.

Posebno je potrebno pozabaviti se osobinama koje se u prethodnom prozoru definišu i pridružuju zrnu vođenim porukama:

- ***acknowledgeMode*** - može imati dve vrednosti ***AUTO_ACKNOWLEDGE*** ili ***DUPS_OK_ACKNOWLEDGE***. Prva vrednost znači da server potvrđuje poruku odmah nakon prijema, dok druga vrednost ukazuje da server može da potvrdi poruku u bilo kojem vremenu nakon njenog prijema;
- ***clientId*** - vrednost ima slobodnu formu. Osobina predstavlja *ID* trajnih korisnika (pretplatnika). Osobina je validna samo u slučaju korišćenja *pub/sub* domena poruka i koristi se za *teme*, a nikako za *redove za poruke*;
- ***connectionFactoryLookup*** - vrednost ima slobodnu formu. Osobina se odnosi na *JNDI* naziv za *JMS produkciju konekcije* (JMS connection factory) i podrazumevani naziv predstavlja naziv podrazumevane *JMS produkcije konekcije*;
- ***destinationType*** - može imati dve vrednosti ***TOPIC*** ili ***QUEUE.*** Osobina predstavlja tip destinacije koja je aktuelna u zavisnosti od korišćenog domena JMS poruka: ***TOPIC*** za *pub/sub* domen poruka ili ***QUEUE*** za *PTP* domen poruka;
- ***destinationLookup*** - vrednost ima slobodnu formu. Osobina se odnosi na *JNDI* naziv destinacije JMS poruka - *reda za poruke* ili *teme;*

- ***messageSelector*** - vrednost ima slobodnu formu. Osobina ukazuje da *zrno vođeno porukama* može biti selektivno birano u smislu poruka koje mora da obrađuje;
- ***subscriptionDurability*** - može imati dve vrednosti ***NON_DURABLE*** ili ***DURABLE.*** Osobina se koristi samo u slučaju tema *pub/sub* domena poruka i ukazuje da li će korišćenje (pretplata) opstati nakon restartovanja ili rušenja servera (***DURABLE)*** ili ne (***NON_DURABLE***);
- ***subscriptionName*** - vrednost ima slobodnu formu. Podešava naziv pretplate za trajne (***DURABLE***) pretplate.

Nakon prikaza i davanja opisa zrna vođenog porukama, moguće je posvetiti posebnu pažnju generisanom kodu zrna *PrimalacPoruke.* O tome više u narednom izlaganju.

GENERISANI KOD ZRNA VOĐENOG PORUKAMA

Posebno je značajno posvetiti posebnu pažnju generisanom kodu zrna PrimalacPoruke.

Klikom na dugme *Finish* prozora za podešavanje *zrna vođenog porukama* (videti Sliku 3) i davanja definicije zrnu vođenim porukama, moguće je posvetiti posebnu pažnju generisanom kodu zrna *PrimalacPoruke*. O tome će detaljno biti govora u narednom izlaganju. Sledećim listingom je prikazan generisani kod zrna *PrimalacPoruke* kao rezultat izvršenih podešavanja vođenih čarobnjakom *NetBeans IDE* razvojnog okruženja:

```
package com.metropolitan.JmsDemo.primalac;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

/**
 *
 * @author Vladimir Milicevic
 */
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"jms/mojRed")
    ,
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue")
})
public class PrimalacPoruke implements MessageListener {

    public PrimalacPoruke() {
    }

    @Override
    public void onMessage(Message message) {
    }

}
```

Prvo što je moguće primetiti da je kreirana klasa obeležena anotacijom ***@MessageDriven*** koja ukazuje da se radi o klasi *zrna vođenog porukama.* Atribut ove anotacije ***activationConfig*** prihvata niz anotacija ***@ActivationConfigProperty,*** od kojih svaka specificira naziv konkretne JMS osobine i odgovarajuću vrednost. Obe navedene anotacije su automatski generisane na osnovu obavljenih podešavanja demonstriranih u prethodnom izlaganju i priloženih Slikom 3.

Dalje, moguće je primetiti da kreirana klasa implementira interfejs pod nazivom ***javax.jms.MessageListener.*** Ovo predstavlja zahtev za zrna vođena porukama (message-driven beans). Implementacijom ovog interfejsa, klasa *zrna vođenog porukama* dobija jednu metodu, ***onMessage(),*** koju mora da redefiniše. Metoda uzima, kao jedini parametar, instancu klase ***javax.jms.Message i*** vraća ***void.*** Ova metoda se poziva automatski kada je poruka primljena na *JMS destinaciji* koju kreirano *zrno vođeno porukama* osluškuje.

Metodu je neophodno redefinisati konkretnim funkcionalnostima, a o tome će biti više reči u izlaganju koje neposredno sledi.

REDEFINISANJE METODE ONMESSAGE()

Metodu onMessage() je neophodno snabdeti konkretnim funkcionalnostima.

Kao što je istaknuto u prethodnom izlaganju, kreirana klasa zrna vođenog porukama dobila je jednu metodu od interfejsa ***MessageListener*** i ovu metodu, koja je za sada prazna, neophodno je dopuniti odgovarajućom programskom logikom. Takođe bi trebalo istaći da se ova metoda poziva automatski kada je poruka primljena na *JMS destinaciji* koju *kreirano zrno vođeno porukama* osluškuje. Sledećim listingom je priložen kod kojim je neophodno proširiti definiciju metode ***onMessage()*** kreirane klase *zrna vođenog porukama PrimalacPoruke*.

```
@Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println("Primljena poruka: "
                    + textMessage.getText());
        } catch (JMSException ex) {
            Logger.getLogger(PrimalacPoruke.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }
```

Svi tipovi *JMS poruka* proširuju interfejs ***javax.jms.Message.*** Sa ciljem obrade poruka, neophodno je angažovanje specifičnog podinterfejsa interfejsa ***Message.*** U konkretnom slučaju radi se o prijemu instance interfejsa ***javax.jms.TextMessage.***

U ovom jednostavnom primeru vrši se slanje sadržaja poruke u *log* aplikativnog servera pozivom *System.out.println()* i prosleđivanjem vrednosti *textMessage.getText()* kao parametra.

Metoda ***getText()*** interfejsa ***javax.jms.TextMessage*** vraća ***String*** koji sadrži tekst poruke.

> U realnim aplikacijama ovakav primer bi se bavio realnijim softverskim problemima kao što je popunjavanje tabela baze podataka na osnovu sadržaja poruka ili preusmeravanje poruke na drugu JMS destinaciju, opet na osnovu sadržaja poruke.

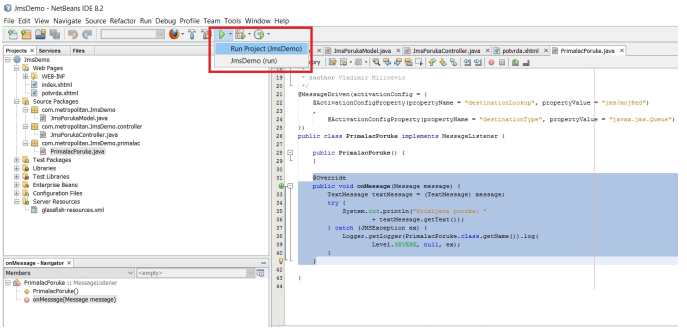
Posebno je važno istaći da metoda ***getText()*** može da izbaci izuzetak tipa ***JMSException*** pa je otuda bilo neophodno i metodi ***onMessage()*** implementirati ***try - catch*** blok za upravljanje izuzecima.

TESTIRANJE APLIKACIJE SA ZRNIMA VOĐENIM PORUKAMA

Na kraju je neophodno proveriti funkcionalnost kreirane aplikacije .

Pre implementacije klase *zrna vođenog porukama*, izvršeno je testiranje privremene verzije aplikacije. Tadašnja funkcionalnost je bila fokusirana na prikazivanje informacije o uspešnom slanju poruke na stranici *potvrda.xhtml* i u suštini nije imala ozbiljnije procesiranje. Sada, u poslednjoj verziji, aplikacija ima funkcionalnost koja dozvoljava njenim modulima i da primaju poruke koje su postavljene na kreirani red *za poruke*. Upravo u ovom delu lekcije je cilj da se proveri da li je celokupan posao, koji je u prethodnim delovima lekcije izlagan i analiziran,

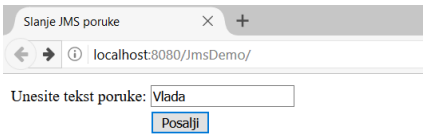
uspešno obavljen. Testiranje je moguće započeti od razvojnog okruženja *NetBEans IDE* gde će biti izvršeno prevođenje i pokretanje kreirane aplikacije. Ovaj zadatak može da se obavi klikom na opciju *Run project* na način prikazan sledećom slikom.



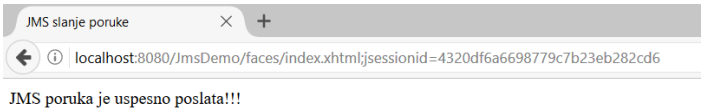
Slika 4.5 Pokretanje urađene aplikacije [izvor: autor]

Ubrzo, u podrazumevanom veb pregledaču, otvara se prozor u kojem je moguće, u odgovarajućem polju za unos teksta uneti sadržaj poruke (Slika 6).

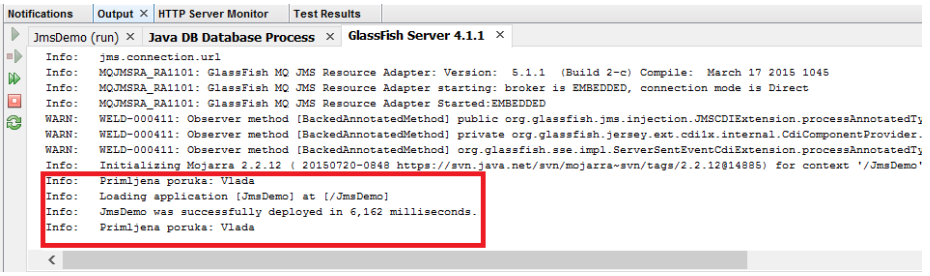
Klikom na dugme *Potvrdi* poruka se prosleđuje i poruka o uspešnom slanju poruke se vidi na stranici prikazanoj na Slici 7. Istovremeno, poruka se prosleđuje na *red za poruke* koji je osluškivan *kreiranim zrnom upravljanim porukama*. Kada se poruka nađe u redu za poruke, preuzima se *zrnom upravljanim porukama* i prikazuje kao *Log* u prozoru aplikativnog servera (Slika 8).



Slika 4.6 Stranica za slanje poruke [izvor: autor]



Slika 4.7 Stranica za potvrdu slanja poruke [izvor: autor]



Slika 4.8 Log informacija u prozoru aplikativnog servera [izvor: autor]

VIDEO MATERIJALI 1

Slede izabrani video materijali iz oblasti zrna vođenih porukama.

EJB Message Driven Bean JMS Theory Part 1

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EJB Message Driven Bean Glassfish Creating Queue and Destination Part 2

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EJB Message Driven Bean Glassfish Creating MessageListener Bean Part 3

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO MATERIJALI 2

Slede još izabranih video materijala iz oblasti zrna vođenih porukama.

EJB Message Driven Bean Glassfish Creating Servlet Part 4

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EJB Message Driven Bean Glassfish Creating Servlet Part 5

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EJB Message Driven Bean Glassfish Running Part 6

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 5

Pokazni primer - Java Messaging Service

POSTAVKA (45 MIN)

Kreiranje JMS aplikacije praktičnim radom na računaru.

Zadatak:

1. U razvojnom okruženju NetBeans IDE kreirajte projekat Java EE veb aplikacije pod nazivom PP10;
2. Projekat uključuje JSF okvir;
3. Kreirati dve JSF stranice: index.xhtml i potvrda.xhtml;
4. Prva stranica preuzima string, u polju za unos teksta i taj string kao JMS poruku, klikom na dugme koje je implementirano na stranici, šalje ga u red za poruke. Druga stranica služi da prikaže poruku o uspešnom slanju unete JMS poruke;
5. Kreirati nov red za poruke za GlassFish aplikativni server;
6. Kreirati model i kontroler proizvođača JMS poruka.
7. Kreirati zrno vođeno porukama za preuzimanje poruke sa reda za poruke;

JSF STRANICE

Kreiraju se JSF stranice koje prikazuju sadržaj vidljiv korisniku aplikacije.

Sledi listing početne stranice index.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Slanje JMS poruke</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="msgText" value="Unesite tekst poruke:"/>
                <h:inputText id="msgText"
                             value="#{jmsPorukaModel.msgText}"/>
```

```
                <h:panelGroup/>
                <h:commandButton value="Posalji"
                                 action="#{jmsPorukaController.sendMsg()}"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

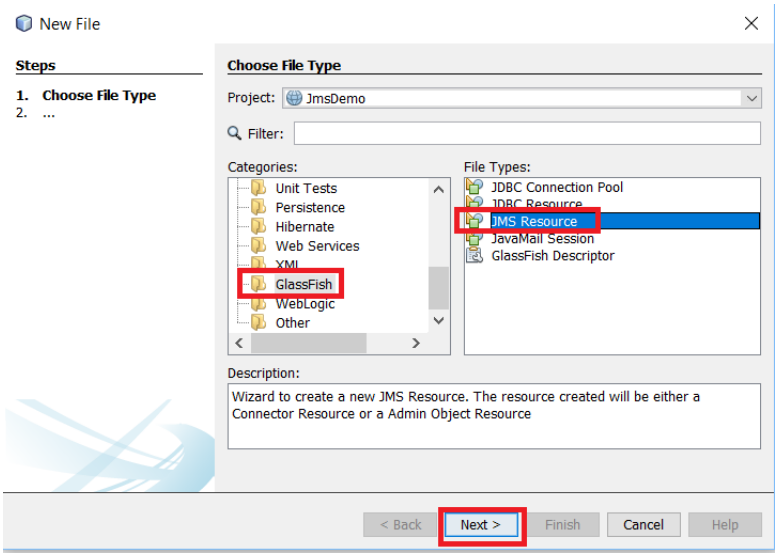
Sledi listing početne stranice potvrda.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>JMS slanje poruke</title>
    </h:head>
    <h:body>
        JMS poruka je uspesno poslata!!!
    </h:body>
</html>
```

DODAVANJE REDA ZA PORUKE

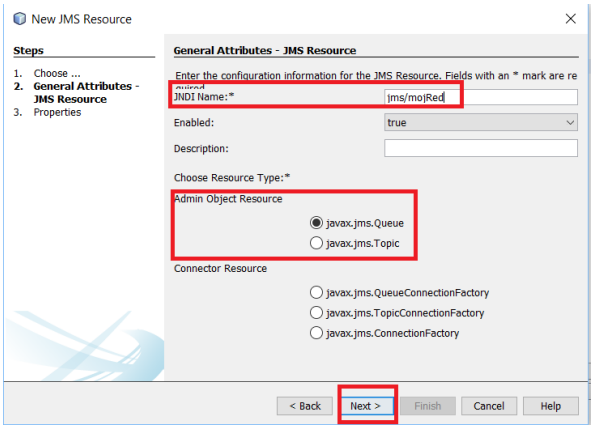
U projektu Vezba10 neophodno je dodati red za poruke u aplikativni server GlassFish.

U razvojnom okruženju NetBeans IDE, za projekat PP10, kreirati novu datoteku kategorije GlassFish, aplikativnog servera, tipa JMS Resource (sledeća slika).



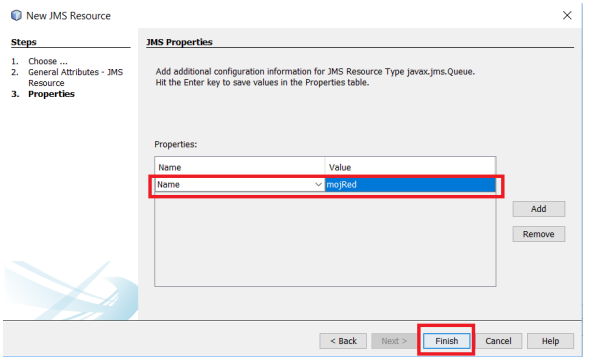
Slika 5.1 Kreiranje JMS resurs - reda za poruke [izvor: autor]

Zatim, u sledećem prozoru za tip resursa izabrati QUEUE, dodeliti mu JNDI naziv jms/mojRed (sledeća slika)



Slika 5.2 Definisanje tipa JMS reursa i njegovog JNDI naziva [izvor: autor]

Na kraju, u poslednjem prozoru je neophodno definisati osobinu Name kreiranog reda za poruke kao na sledećoj slici.



Slika 5.3 Kraj kreiranja reda za poruke projekta PP10 [izvor: autor]

MODEL I KONTROLER JMS PROIZVOĐAČA PORUKA

U narednom izlaganju neophodno je kreirati proizvođača JMS poruka u formi dva CDI zrna.

U formi na stranici index.xhtml, polje za unos teksta je u direktnoj vezi sa modelom JMS proizvođača poruka, a akcija koja je inicirana klikom na dugme kontrole je u direktnoj vezi sa kontrolerom.

Sledi listing klase modela JMS proizvođača poruka:

```
package com.metropolitan.JmsDemo;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
/**
 *
 * @author Vladimir Milicevic
```

```
 */
@Named
@RequestScoped
public class JmsPorukaModel {
    private String msgText;

    public String getMsgText() {
        return msgText;
    }

    public void setMsgText(String msgText) {
        this.msgText = msgText;
    }


}
```

Sledi listing klase kontrolera JMS proizvođača poruka:

```
package com.metropolitan.JmsDemo.controller;

import com.metropolitan.JmsDemo.JmsPorukaModel;
import javax.annotation.Resource;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.jms.JMSConnectionFactory;
import javax.jms.JMSContext;
import javax.jms.Queue;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JmsPorukaController {

    @Inject
    private JmsPorukaModel jmsPorukaModel;

    @Resource(mappedName = "jms/mojRed")
    private Queue mojRed;

    @Inject
    @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory")
    private JMSContext context;

    public String sendMsg() {
        sendJMSMessageToMojRed(jmsPorukaModel.getMsgText());
        return "potvrda";
    }
```

```
    private void sendJMSMessageToMojRed(String messageData) {
        context.createProducer().send(mojRed, messageData);
    }

}
```

ZRNO VOĐENO PORUKAMA

Neophodno je kreirati zrno vođeno porukama koje preuzima poruku sa reda za poruke.

Na predavanjima je detaljno pokazano kako se navigacijom kroz čarobnjak razvojnog okruženja NetBeans IDE veoma lako generiše većina koda zrna vođenog porukama. Ovde će biti ispoštovana kompletna procedura sa akcentom na sledeće:

- U razvojnom okruženju, NetBeans IDE bira se opcija File | New File, a zatim se, u konkretnom prozoru, iz kategorije Enterprise JavaBeans bira kreiranje datoteke tipa Message-Driven Bean.
- U nastavku je neophodno, nakon klika na dugme Next i otvaranja novog prozora, dati naziv klasi zrna vođenog porukama, definisati paket kojem će kreirana klasa pripadati i čekirati opciju Server Destinations gde je izabran kreirani red za poruke jms/MojRed;
- U sledećem prozoru za osobinu destinationLookup izabrati kreirani red sa porukama jms/mojRed;
- Izvršiti redefinisanje i dopuno koda klase na način priložen sledećim listingom.

```
package com.metropolitan.JmsDemo.primalac;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 *
 * @author Vladimir Milicevic
 */
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"jms/mojRed")
    ,
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue")
})
public class PrimalacPoruke implements MessageListener {
```

```
    public PrimalacPoruke() {
    }

    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println("Primljena poruka: "
                    + textMessage.getText());
        } catch (JMSException ex) {
            Logger.getLogger(PrimalacPoruke.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

}
```

Poglavlje 6

Individualne vežbe 10

INDIVIDUALNA VEŽBA (135 MIN)

Pokušajte sami.

Pokušajte sami sledeće:

1. Kreirajte novu JEE aplikaciju
2. JSF stranice implementirajte primenom neke od poznatih JSF biblioteka komponenata;
3. Osmislite na drugi način zadavanje JMS poruke (na primer predefinisane su ali se biraju čekiranjem odgovarajućeg CheckBox - a ili RadioButton - a);
4. Koristite GlassFish veb konzolu za proveru kreirane JMS destinacije

NAPOMENA:

1. Korisniku je ponuđeno nekoliko predefinisanih poruka.
2. On ih bira putem kontrola CheckBox ili RadioButton.
3. U prvom slučaju, sve izabrane poruke je neophodno spojiti u jedan string (poruku) koji će biti sačuvan na redu za poruke i dostupan ostalim korisnicima za preuzimanje.
4. U drugom slučaju, moguće je izabrati samo jednu poruku koja će biti sačuvana u kreiranom redu za poruke.

Dokumentujte urađeni primer i obratite se predmetnom asistentu.

Poglavlje 7

Domaći zadatak 10

ZADATAK 1 (120 MIN)

Uradite domaći zadatak.

Uradite domaći zadatak po sledećim zahtevima:

1. na formi prve stranice (index.xhtml) postoji pitanje sa tri ponuđena odgovora;
2. odgovori su organizovani kao RadioButton grupa;
3. klikom na dugme Posalji odgovor se kao poruka šalje na red za poruke;
4. u zavisnosti od odgovora na posebnoj stranici se daje potvrda da li odgovor tačan ili ne;
5. JSF stranice aplikacije realizovati primenom neke od poznatih JSF biblioteka komponenata;
6. realizovati zrno vođeno porukama za osluškivanje i preuzimanje poruka sa reda za poruke kao u zadatku sa vežbi;
7. koristite GlassFish veb konzolu za proveru kreirane JMS destinacije.

Nakon urađenog obaveznog zadatka, studenti dobijaju i različite zadatke od predmetnog asistenta.

Poglavlje 8

Zaključak

ZAKLJUČAK

Lekcija je detalno obradila Java Message Service (JMS) API za upravljanje porukama u Java EE.

Lekcija je detaljno obradila ***Java Message Service (JMS) API*** za upravljanje porukama u ***Java EE.*** Nakon uvodnih razmatranja, u lekciji, posebno je istaknuto da *Java Message Service (JMS)* predstavlja standardni *Java EE API* (Application Programming Interface) za upravljanje porukama koji omogućava, u formi labave povezanosti komponenata, asinhronu komunikaciju između Java EE komponenata.

Dalje je akcenat stavljen na implementaciju kroz savremena Java razvojna okruženja. Ovde je, posebno, istaknuto da savremena Java razvojna okruženja, a među njima je i *NetBeans IDE*, omogućavaju odličnu podršku koja je od velike pomoći za kreiranje Java EE aplikacija koje se baziraju na punoj prednosti primene *JMS API,* generišući većinu specifičnog JMS koda pri čemu se fokus programera ne rasipa i stavlja na razvoj poslovne logike konkretne aplikacije.

Lekcija je kreirana na takav način da su JMS teme veoma pažljivo birane te je moguće istaći da za potpuno razumevanje principa i koncepata *JMS API,* koji će biti korišćeni prilikom razvoja Java EE aplikacija, u ovoj lekciji su izabrane teme pažljivo analizirane i diskutovane. Ono na čemu je posebno insistirano tokom pripremanja lekcije su primeri. Zato je posebna pažnja bila stavljena na podršku izlaganju u formi pažljivo biranih i detaljno razrađenih i objašnjenih pokaznih primera.

Zbog svega navedenog, lekcija se posebno bavila sledećim važnim JMS temama:

- Uvod u JMS - u ovom delu lekcije je detaljno predstavljen JMS API uključujući i dva domena JMS poruka;
- Kreiranje JMS resursa - u ovom delu lekcije je detaljno obrađen koncept JMS destinacije. Odavde je započet projekat kao podrška izlaganju u lekciji i u okviru projekta je bilo pokazano kako se dodaje JMS destinacija u aplikativni server. Na kraju, ovog dela lekcije, prikazan je način provere kreiranih JMS resursa u GlassFish veb konzoli;
- Implementacija JMS proizvođača - u ovom delu lekcije poseban akcenat je stavljen na kreiranje i implementaciju klasa modela i kontrolera JMS proizvođača. U ovom delu lekcije su kreirane i JSF stranice neophodne za testiranje aplikacije koja je potom kreirana;
- Korišćenje JSM poruka pomoću zrna vođenih porukama - ovaj deo lekcije se bavio kreiranjem i implementacijom zrna vođenih porukama, kao sredstva za preuzimanje poruka sa prethodno kreirane JMS destinacije.

Na kraju je važno istaći da savladavanjem ove lekcije student je osposobljen da u potpunosti

razume i koristi Java Messaging System i zrna vođena porukama u Java EE aplikacijama. Takođe, student je ovladao dodatnim naprednim alatom za razvoj distribuiranih veb sistema.

LITERATURA

Za pripremu lekcije korišćena je najnovija literatura.

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. HeffelFinger. 2015. Java EE7 Develpomnet With NetBeans 8, PACK Publishing
3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh JUneau. 2015. Java EE7 Recipes, Apress