



IT355 - WEB SISTEMI 2

JMS i poruke u Springu

Lekcija 11

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 11

JMS I PORUKE U SPRINGU

- ✓ JMS i poruke u Springu
- ✓ Poglavlje 1: Slanje i prijem JMS poruka u Springu
- ✓ Poglavlje 2: Konverzija JMS poruka
- ✓ Poglavlje 3: Upravljanje JMS transakcijama
- ✓ Poglavlje 4: Kreiranje POJO-a baziranih na porukama u Springu
- ✓ Poglavlje 5: Kreiranje konekcije
- ✓ Poglavlje 6: Pokazni primer - Spring 4 / 5 + JMS + ActiveMQ
- ✓ Poglavlje 7: Pokazna vežba 11
- ✓ Poglavlje 8: Individualna vežba 11
- ✓ Poglavlje 9: Domaći zadatak 11
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Lekcija se bavi izučavanjem Spring podrške za JMS (Java Messaging Service).

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

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 server.
- **Implementacija JMS proizvođača** - resurs koji kreira poruku koja se čuva na kreiranoj destinaciji;
- **Korišćenje JMS 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.

Spring pojednostavljuje primenu *JMS*-a, pristupom koji se oslanja na primenu šablona. Spring dozvoljava zrnima deklarisanim u *Spring IoC* kontejneru da osluškiju JMS poruke i da reaguju na njih.

Po savladavanju ove lekcije, studenti će biti osposobljeni da kreiraju i pristupaju srednjem sloju aplikacije baziranom na porukama primenom *Springa* i *JMS*. Moći će aktivno da koriste *Spring JMS* podršku za pojednostavljeno slanje, prijem i osluškivanje *JMS* poruka.

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

▼ Poglavlje 1

Slanje i prijem JMS poruka u Springu

PROBLEM KOMUNIKACIJE JEE KOMPONENATA

Na JAVA EE platformi, aplikacije često moraju da komuniciraju preko JMS.

Kada se sagleda razvoj na JAVA EE platformi, aplikacije često moraju da komuniciraju preko JMS. Za slanje i prijem poruka, neophodno je izvesti sledeće zadatke:

- Kreiranje JMS produkcije **konekcije** (**connection factory**);
- Kreiranje JMS destinacije (**destination**);
- Otvaranje JMS konekcije pomoću **produkcije konekcije**;
- Slanje ili prijem JMS poruke pomoću proizvođača (**producer**) ili korisnika poruke (**consumer**);
- Upravljanje izuzecima tipa JMSException;
- Zatvaranje JMS sesije i JMS konekcije.

Na osnovu navedenih koraka moguće je izvesti zaključak da je neophodno primeniti dosta kodiranja za slanje / prijem jednostavne JMS poruke. Takođe, većinu ovih zadataka je neophodno ponavljati svaki put kada se radi sa JMS.

Spring nudi veoma elegantno i jednostavno rešenje koje se oslanja na primenu šablona za pojednostavljivanje JMS koda. Sa JMS šablonima (Spring klasa **JmsTemplate**) moguće je slanje i prijem poruka realizovati pomoću mnogo manje koda. Šabloni upravljaju zadacima i omogućavaju, takođe, konvertovanje hijerarhije izuzetaka **JMS API** - ja **JMSException** u Spring hijerarhiju izuzetaka **org.springframework.jms.JmsException**.

▼ 1.1 Rad sa porukama bez JmsTemplate Spring podrške

SIMULACIJA KOMUNIKACIJE IZMEĐU DVE APLIKACIJE

Simulacija razmene poruka između poštanskih službi.

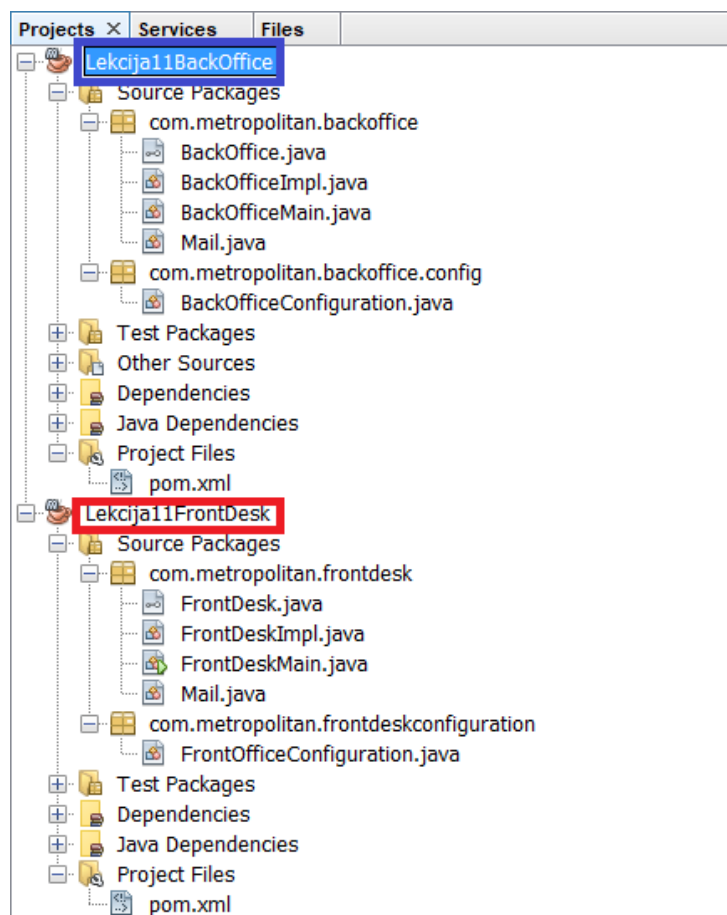
Kao i u svim dosadašnjim lekcijama, rad na problematici će se vrteti oko odgovarajućeg pratećeg primera. Pretpostavka je da se razvija aplikacija poštanske kancelarije sa dva

podсистema: *front-desk* (šalter) i *back-office* (prateća služba). Kada šalter primi poštu od nekog pošiljaoca, prosleđuje je pratećoj službi na kategorizaciju i dostavu.

Za početak, biće kreirana dva projekta. Kreirane aplikacije će simulirati rad šaltera i prateće službe respektivno. Sledećom slikom je prikazana konačna struktura kreiranih projekata u razvojnom okruženju.

Za rad sa **JMS** porukama, neophodno je dodati i sledeće zavisnosti u datoteke *pom.xml* kreiranih projekata.

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.15.1</version>
</dependency>
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>7.0</version>
    <type>jar</type>
</dependency>
```



Slika 1.1.1 Kreirani projekti za simulaciju rada poštanskih službi [izvor: autor]

MODEL I SERVISI

Kreiranje klase pošiljke i metoda za slanje i prijem informacija o pošiljci.

Kao što je i nagovešteno, u prethodnom izlaganju, radi se na razvoju sistema poštanske kancelarije koji uključuje dva podsistema:

- šalter (front office);
- prateća služba (back office).

Scenario rada ove kancelarije može biti zamišljen tako da kada šalter primi pošiljku prosleđuje informaciju, u vidu *JMS* poruke, pratećoj službi za beleženje (kategorizaciju) primljene pošiljke i njeno konačno prosleđivanje. Informacije koje se razmenjuju navedenom porukom sadržane su u klasi *Mail.java* koja je zajednička za oba kreirana projekta.

```
/**
 *
 * @author Vlada
 */
public class Mail {

    private String mailId;
    private String country;
    private double weight;

    // konstruktori, seteri i geteri

}
```

U nastavku je neophodno obezbediti metode za slanje i prijem poruka. Interfejsi, kojima su definisane navedene metode, pripadaju različitim projektima: šalter i prateća služba, respektivno. Sledi listing interfejsa *FrontDesk*.

```
package com.metropolitan.frontdesk;

/**
 *
 * @author Vlada
 */
public interface FrontDesk {

    public void sendMail(Mail mail);

}
```

Sledi listing interfejsa *BackOffice*.

```
package com.metropolitan.backoffice;

/**
```

```
*
* @author Vlada
*/
public interface BackOffice {

    public Mail receiveMail();
}
```

Pre slanja ili prijema JMS poruka, neophodno je obezbediti brokera JMS poruka, kao što je Apache ActiveMQ (<http://activemq.apache.org/>).

SLANJE PORUKE BEZ JMSTEMPLATE SPRING PODRŠKE

Prvo će biti demonstrirano slanje JMS poruke bez JmsTemplate podrške.

Prvo će biti demonstrirano slanje *JMS* poruke bez *Spring* podrške. Prvo će biti ugrađene odgovarajuće zavisnosti u projekat. Kreira se implementaciona klasa *FrontDeskImpl*, za interfejs, *FrontDesk*, koja šalje JMS poruke koristeći direktno standardni Java EEJMS API.

```
package com.metropolitan.frontdesk;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.MessageProducer;
import javax.jms.Session;

import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl implements FrontDesk {

    @Override
    public void sendMail(Mail mail) {
        ConnectionFactory cf
            = new ActiveMQConnectionFactory("tcp://localhost:61616");
        Destination destination = new ActiveMQQueue("mail.queue");

        Connection conn = null;
        try {
            conn = cf.createConnection();
            Session session
                = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer producer = session.createProducer(destination);
```

```

        MapMessage message = session.createMapMessage();
        message.setString("mailId", mail.getMailId());
        message.setString("country", mail.getCountry());
        message.setDouble("weight", mail.getWeight());
        producer.send(message);
        System.out.println("Pošiljka je poslata!!!");
        session.close();
    } catch (JMSException e) {
        throw new RuntimeException(e);
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (JMSException e) {
            }
        }
    }
}
}
}

```

U prethodnoj `sendMail()` metodi, prvo je kreiran *JMS*-specifični **ConnectionFactory** i objekti **Destination** pomoću klasa obezbeđenih sa **ActiveMQ**. URL brokera poruka je podrazumevani za **ActiveMQ** ako se pokreće na **localhost** -u. U *JMS* postoje dva tipa destinacija: **queue** i **topic**. Prvi je namenjen za tačka - u - tačku (**point - to - point**) model komunikacija, a drugi tip se koristi za pretplatnički (**subscribe**) komunikacioni model. Budući da se po prvom modelu vrši slanje *JMS* poruka od šaltera ka pratećoj službi, biće korišćen **red za poruke**, tj **queue**.

Dalje, neophodno je kreirati konekciju, sesiju i proizvođača poruke pre nego što bude omogućeno slanje poruke. Postoji nekoliko tipova poruka uključenih u *JMS API*, uključujući sledeće: **TextMessage**, **MapMessage**,

BytesMessage, **ObjectMessage** i **StreamMessage**. **MapMessage** podrazumevaju sadržaj poruke organizovan po principu **key/value**. Svi ovi tipovi su, zapravo interfejsi sa zajedničkom superklasom **Message**. U međuvremenu, neophodno je rukovati izuzetkom **JMSException** koji će biti pomoću *JMS API*. Na kraju, neophodno je zatvoriti sesiju i konekciju i osloboditi sistemске resurse. Svaki put kada je *JMS konekcija* zatvorena, sve otvorene sesije biće automatski zatvorene. Dakle, neophodno je obezbediti da *JMS konekcija* bude pravilno zatvorena u bloku **finally**.

PRIJEM PORUKE BEZ JMSTEMPLATE SPING PODRŠKE

Klasa BackOfficeImpl prima JMS poruku direktno putem JMS API.

Na drugoj strani, klasa **BackOfficeImpl**, implementaciona za interfejs **BackOffice**, prima *JMS* poruku direktno putem *JMS API*:

```

package com.metropolitan.backoffice;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;

```



```
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.MessageConsumer;
import javax.jms.Session;

import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl implements BackOffice {

    @Override
    public Mail receiveMail() {
        ConnectionFactory cf =
            new ActiveMQConnectionFactory("tcp://localhost:61616");
        Destination destination = new ActiveMQQueue("mail.queue");

        Connection conn = null;
        try {
            conn = cf.createConnection();
            Session session =
                conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageConsumer consumer = session.createConsumer(destination);

            conn.start();
            MapMessage message = (MapMessage) consumer.receive();
            Mail mail = new Mail();
            mail.setMailId(message.getString("mailId"));
            mail.setCountry(message.getString("country"));
            mail.setWeight(message.getDouble("weight"));

            session.close();
            return mail;
        } catch (JMSException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (JMSException e) {
                }
            }
        }
    }
}
```

Moguće je primetiti da je korišćena metoda konekcije [start\(\)](#) koja nije korišćena pre u [FrontDeskImpl](#) primeru. Kada se koristi **Connection** za prijem poruka, moguće je dodati osluškivače na konekciju, koji se pozivaju na prijemu poruke, ili

blokirati sinhrono, čekajući poruke da stignu. Kontejner ne može da zna koji će pristup biti upotrebljen i ne uključuje se u rad sa porukama do eksplicitnog poziva metode `start()`. Ukoliko se dodaju osluškivači ili bilo koji tip konfiguracije, to je neophodno učiniti pre poziva metode `start()`.

JAVA KONFIGURACIJE PODSISTEMA ŠALTER I PRATEĆA SLUŽBA

Neophodno je instancirati zrna koja će biti korišćena u daljem radu.

Konačno, neophodno je za svaki od projekata kreirati konfiguracione klase. Ovim klasama je neophodno instancirati zrna koja odgovaraju kreiranim implementacionim klasama.

Sledi listing klase *FrontOfficeConfiguration.java*.

```
package com.metropolitan.frontdeskconfiguration;

import com.metropolitan.frontdesk.FrontDeskImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 *
 * @author Vlada
 */
@Configuration
public class FrontOfficeConfiguration {

    @Bean
    public FrontDeskImpl frontDesk() {
        return new FrontDeskImpl();
    }
}
```

Konačno, sledi listing i druge konfiguracione klase pod nazivom *BackOfficeConfiguration.java*.

```
package com.metropolitan.backoffice.config;

import com.metropolitan.backoffice.BackOfficeImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 *
 * @author Vlada
 */
@Configuration
public class BackOfficeConfiguration {

    @Bean
    public BackOfficeImpl backOffice() {
        return new BackOfficeImpl();
    }
}
```

```
}  
}
```

GLAVNE KLASJE PROJEKATA

Neophodno je kreirati i klase za slanje i prijem JMS poruka.

Konačno neophodno je kreirati i dve klase, po jednu u svakom projekata, snabdevenih poznatim `main()` metodama za testiranje urađenih zadataka za slanje u prijem JMS poruka.

Sledi listing klase *FrontDeskMain.java*.

```
package com.metropolitan.frontdesk;  
  
import com.metropolitan.frontdeskconfiguration.FrontOfficeConfiguration;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
/**  
 *  
 * @author Vlada  
 */  
public class FrontDeskMain {  
  
    public static void main(String[] args) {  
  
        ApplicationContext context = new  
AnnotationConfigApplicationContext(FrontOfficeConfiguration.class);  
  
        FrontDesk frontDesk = context.getBean(FrontDesk.class);  
        frontDesk.sendMail(new Mail("1234", "RS", 1.5));  
  
    }  
}
```

Konačno i drugi projekat može biti zaokružen klasom *BackOfficeMain.java*.

```
package com.metropolitan.backoffice;  
  
import com.metropolitan.backoffice.config.BackOfficeConfiguration;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
/**  
 *  
 * @author Vlada  
 */  
public class BackOfficeMain {  
  
    public static void main(String[] args) {
```

```

ApplicationContext context = new
AnnotationConfigApplicationContext(BackOfficeConfiguration.class);

BackOffice backOffice = context.getBean(BackOffice.class);
Mail mail = backOffice.receiveMail();
System.out.println("Pošiljka #" + mail.getMailId() + " je primljena");
}
}

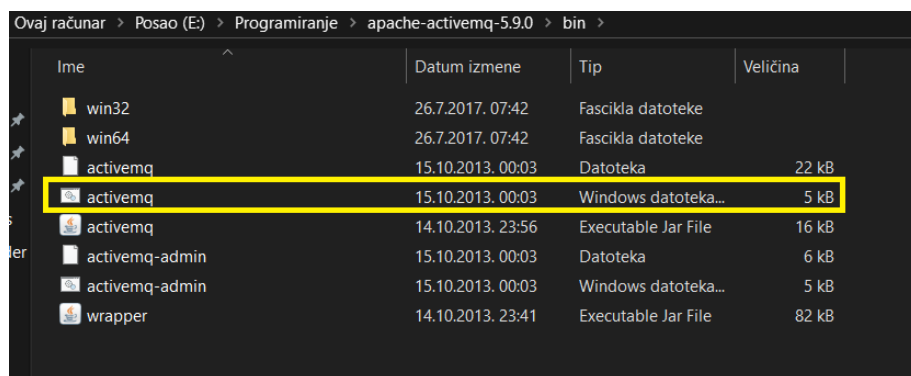
```

BROKER JMS PORUKA

Neophodno je pokrenuti brokera poruka pre slanja i prijema poruka.

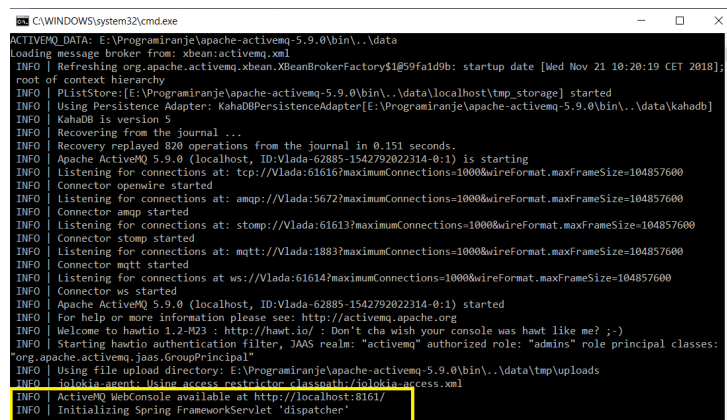
Da bi primer mogao bez problema da funkcioniše, neophodno je obezbediti brokera JMS poruka, odnosno pokrenuti *Apache ActiveMQ server*.

Lokacija sa koje se pokreće server na kojem se čuva poruka na redu, a potom preuzima sa reda i čita, prikazana je sledećom slikom.



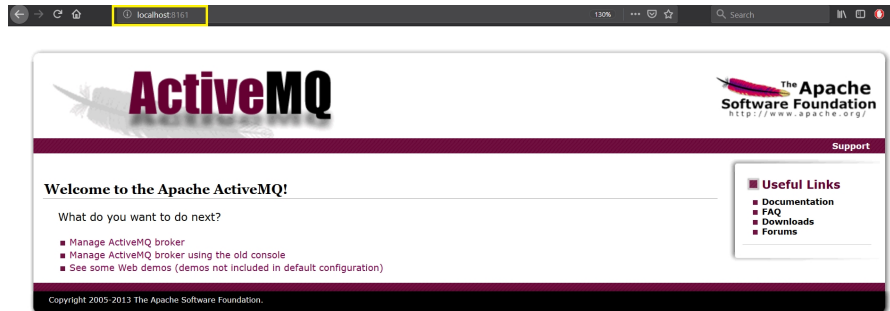
Slika 1.1.2 Lokacija na disku sa koje se pokreće ActiveMQ server [izvor: autor]

Ubrzo, server se pokreće, a to je moguće pratiti u *Command Prompt* prozoru, kao na sledećoj slici.



Slika 1.1.3 Pokrenut ActiveMQ server [izvor: autor]

Na poslednjoj slici, žutom bojom je istaknut link: <http://localhost:8161/> na kojem je moguće vršiti monitoring rada sa JMS porukama na *ActiveMQ* konzoli.



Slika 1.1.4 ActiveMQ konzolia [izvor: autor]

DEMONSTRACIJA

Demonstracija slanja i prijema JMS poruka bez Spring podrške.

Tek sada su kreirani podsistemi spremni da šalju i primaju *JMS* poruke. Neophodno je pokrenuti brokera poruka pre slanja i prijema poruka, sledećim glavnim klasama. Prvo se pokreće *FrontDeskMain*, a zatim *BackOfficeMain* u drugom prozoru ili konzoli.

Rezultat izvršavanja klase *FrontDeskMain* je prikazan sledećom slikom.

```
Lekcija11FrontDesk (clean) x Build (Lekcija11FrontDesk) x Run (Lekcija11FrontDesk) x
--- exec-maven-plugin:1.2.1:exec (default-cli) @ Lekcija11FrontDesk
Pošiljka je poslata!!!
-----
BUILD SUCCESS
-----
Total time: 3.199s
Finished at: Wed Nov 21 08:39:33 CET 2018
Final Memory: 5M/107M
-----
```

Slika 1.1.5 Uspešno slanje poruke o pošiljci [izvor: autor]

Rezultat izvršavanja klase *BackOfficeMain* je prikazan sledećom slikom.

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ Lekcija11BackOffice
Pošiljka #1234 je primljena
-----
BUILD SUCCESS
-----
Total time: 2.324s
Finished at: Wed Nov 21 08:40:34 CET 2018
Final Memory: 5M/135M
-----
```

Slika 1.1.6 Uspešan prijem poruke o pošiljci [izvor: autor]

▼ 1.2 Rad sa porukama sa JmsTemplate Spring podrškom

SLANJE PORUKA POMOĆU SPRING JMS ŠABLONA

Spring JMS šablon značajno pojednostavljuje JMS kod.

Spring JMS šablon značajno pojednostavljuje JMS kod, jednostavnim pozivom metode `send()` i obezbeđivanjem destinacije poruke, kao i objekta MessageCreator koji kreira *JMS poruku* koja će biti poslata. MessageCreator objekat je obično implementiran kao unutrašnja anonimna klasa:

```
package com.metropolitan.frontdesk;

import javax.jms.Destination;
import javax.jms.MapMessage;
import org.springframework.jms.core.JmsTemplate;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl implements FrontDesk {

    private JmsTemplate jmsTemplate;
    private Destination destination;

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public void setDestination(Destination destination) {
        this.destination = destination;
    }

    @Override
    public void sendMail(final Mail mail) {
        jmsTemplate.send(destination, session -> {
            MapMessage message = session.createMapMessage();
            message.setString("mailId", mail.getMailId());
            message.setString("country", mail.getCountry());
            message.setDouble("weight", mail.getWeight());
            return message;
        });
        System.out.println("Pošiljka je poslata!!!");
    }
}
```

Moguće je primetiti da unutrašnja klasa može da pristupa samo argumentima ili varijablama ograđujuće metode koji su deklarirani kao **final**. **MessageCreator** interfejs deklarirše samo jednu metodu **createMessage()** koju je neophodno implementirati. U ovoj metodi, kreira se i vraća JMS *poruka* sa obezbeđenom *JMS sesijom*. *JMS šablon* pomaže prihvatanju i oslobađanju *JMS sesije i konekcije* i šalje *JMS poruku* kreiranu objektom **MessageCreator**. Takođe, metoda vrši konvertovanje izuzetaka tipa **JMS API JMSException** u izuzetke tipa **Spring JMS runtime exception**, čija je osnovna klasa **org.springframework.jms.JmsException**.

U priloženom listingu pomenuta unutrašnja klasa je realizovana kao lambda izraz. Sledećom slikom je prikazan ekvivalentan kod o kojem je bilo diskusije u prethodnom izlaganju.

```
public void sendMail(final Mail mail) {
    jmsTemplate.send(destination, new MessageCreator() {
        public Message createMessage(Session session) throws JMSException {
            MapMessage message = session.createMapMessage();
            message.setString("mailId", mail.getMailId());
            message.setString("country", mail.getCountry());
            message.setDouble("weight", mail.getWeight());
            return message;
        }
    });
}
```

Slika 1.2.1 Anonimna klasa MessageCreator [izvor: autor]

KONFIGURISANJE SLANJA PORUKA

Neophodno je modifikovati kreiranu konfiguracionu klasu za projekat šaltera.

U konfiguracionom zrnu podsistema šaltera, biće konfigurisan *JMS šablon* koji ukazuje na produkciju *JMS konekcije* za otvaranje konekcije. Nakon toga, vrši se umetanje ovog šablona, kao i destinacije poruke, u zrno šaltera.

```
package com.metropolitan.frontdeskconfiguration;

import com.metropolitan.frontdesk.FrontDeskImpl;
import javax.jms.ConnectionFactory;
import javax.jms.Queue;

import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.core.JmsTemplate;

/**
 *
 * @author Vlada
 */
@Configuration
public class FrontOfficeConfiguration {
```

```

@Bean
public ConnectionFactory connectionFactory() {
    return new ActiveMQConnectionFactory("tcp://localhost:61616");
}

@Bean
public Queue mailDestination() {
    return new ActiveMQQueue("mail.queue");
}

@Bean
public JmsTemplate jmsTemplate() {
    JmsTemplate jmsTemplate = new JmsTemplate();
    jmsTemplate.setConnectionFactory(connectionFactory());
    return jmsTemplate;
}

@Bean
public FrontDeskImpl frontDesk() {
    FrontDeskImpl frontDesk = new FrontDeskImpl();
    frontDesk.setJmsTemplate(jmsTemplate());
    frontDesk.setDestination(mailDestination());
    return frontDesk;
}
}

```

PRIJEM PORUKA POMOĆU SPRING JMS ŠABLONA

Za prijem JMS poruke, JMS šablonom, poziva se metoda `receive()`.

Za prijem *JMS poruke*, *JMS šablonom*, poziva se metoda `receive()` uz obezbeđivanje destinacije poruke. Ova metoda vraća *JMS poruku*, `javax.jms.Message`, koja predstavlja osnovni tip *JMS poruke* (a to je interfejs), pa ju je neophodno prevesti u odgovarajući tip pre daljeg rada. U ovom slučaju to je tip `MapMessage`.

```

package com.metropolitan.backoffice;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MapMessage;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.support.JmsUtils;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl implements BackOffice {

```



```
private JmsTemplate jmsTemplate;
private Destination destination;

public void setJmsTemplate(JmsTemplate jmsTemplate) {
    this.jmsTemplate = jmsTemplate;
}

public void setDestination(Destination destination) {
    this.destination = destination;
}

@Override
public Mail receiveMail() {
    MapMessage message = (MapMessage) jmsTemplate.receive(destination);
    try {
        if (message == null) {
            return null;
        }
        Mail mail = new Mail();
        mail.setMailId(message.getString("mailId"));
        mail.setCountry(message.getString("country"));
        mail.setWeight(message.getDouble("weight"));
        return mail;
    } catch (JMSException e) {
        throw JmsUtils.convertJmsAccessException(e);
    }
}
}
```

Međutim, prilikom izvlačenja informacija iz primljenog **MapMessage** objekta, i dalje je neophodno rukovati **JMS API JMSException** izuzecima. Ovde je neophodno izvršiti poziv [JmsUtils.convertJmsAccessException\(\)](#) za konverziju **JMS API JMSException** u Spring **JmsException**.

KONFIGURISANJE PRIJEMA PORUKA

Neophodno je modifikovati kreiranu konfiguracionu klasu za projekat prateće službe.

U konfiguracionom zrnu podsistema prateće službe, biće deklarisan **JMS šablon** i umetnut zajedno sa destinacijom poruke u zrno prateće službe.

```
package com.metropolitan.backoffice.config;

import com.metropolitan.backoffice.BackOfficeImpl;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.core.JmsTemplate;
import javax.jms.ConnectionFactory;
import javax.jms.Queue;

/**
 *
 * @author Vlada
 */
@Configuration
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public Queue destination() {
        return new ActiveMQQueue("mail.queue");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setReceiveTimeout(10000);
        return jmsTemplate;
    }

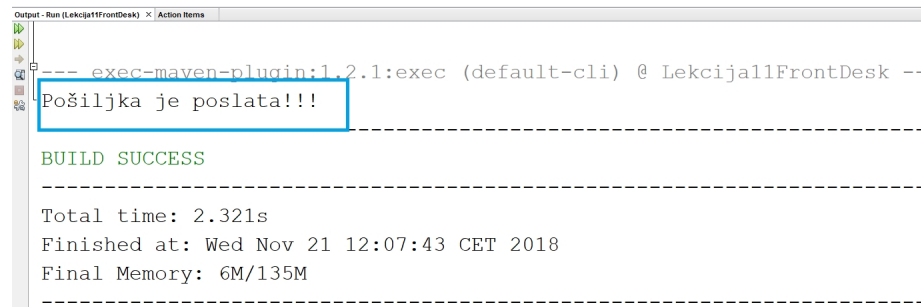
    @Bean
    public BackOfficeImpl backOffice() {
        BackOfficeImpl backOffice = new BackOfficeImpl();
        backOffice.setDestination(destination());
        backOffice.setJmsTemplate(jmsTemplate());
        return backOffice;
    }
}
```

Primećuje se *receiveTimeout* (u milisekundama) osobina *JMS šablona* kojom je određeno vreme čekanja *JMS poruke* na odgovarajućoj destinaciji. Po osnovnim podešavanjima to je *zauvek* i pozivajuća nit je blokirana u međuvremenu. Da poruka ne bi dugo čekala, neophodno je specificirati *receiveTimeout* za ovaj šablon. Ukoliko na destinaciji, u ovom periodu, nema dostupnih poruka *receive()* će vratiti *null*.

PRIJEM I SLANJE PORUKE POMOĆU SPRING JMS ŠABLONA - DEMO

Demonstracija slanja i prijema JMS poruka pomoću Spring JMS šablona.

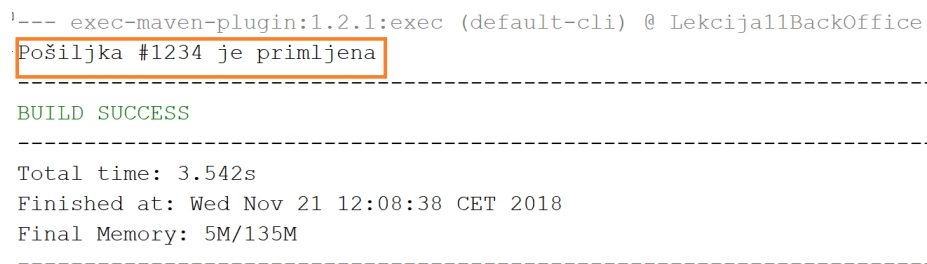
Za učinjene izmene na programu, neophodno je testirati urađeni primer. Primer je trenutno zadužen za upravljanje *JMS* porukama, pomoću *Spring JMS šablona*. Prvo će biti pokrenuta modifikovana klasa *FrontDeskMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.



```
Output - Run (Lekcija1FrontDesk) x Action Items
--- exec-maven-plugin:1.2.1:exec (default-cli) @ Lekcija1FrontDesk ---
Pošiljka je poslata!!!
BUILD SUCCESS
Total time: 2.321s
Finished at: Wed Nov 21 12:07:43 CET 2018
Final Memory: 6M/135M
```

Slika 1.2.2 Slanje JMS poruke pomoću Spring JMS šablona [izvor: autor]

Konačno, pokreće se i modifikovana klasa *BackOfficeMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.



```
'--- exec-maven-plugin:1.2.1:exec (default-cli) @ Lekcija1BackOffice
Pošiljka #1234 je primljena
BUILD SUCCESS
Total time: 3.542s
Finished at: Wed Nov 21 12:08:38 CET 2018
Final Memory: 5M/135M
```

Slika 1.2.3 Prijem JMS poruke pomoću Spring JMS šablona [izvor: autor]

Urađeni primer možete preuzeti iz aktivnosti Shared Resources na kraju objekta učenja.

▼ 1.3 Rad sa porukama sa podrazumevanim destinacijama

KONFIGURISANJE PODRAZUMEVANE JMS DESTINACIJE

Više ne postoji potreba za umetanjem destinacije u zrna pošiljaoca i prijemnika poruke.

Umesto specifikiranja destinacije poruke za svaki poziv metoda *send()* i *receive()* JMS šablona, moguće je specifikirati *podrazumevanu destinaciju za JMS šablon*. U tom slučaju, ne postoji potreba za njenim umetanjem u zrna pošiljaoca i prijemnika poruke.

U tom svetlu bi trebalo redefinisati posmatrane konfiguracione klase podsistema šaltera i prateće službe. Nova definicija klase *FrontOfficeImpl.java* glasi:

```
@Configuration
public class FrontOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public Queue mailDestination() {
        return new ActiveMQQueue("mail.queue");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setDefaultDestination(mailDestination());
        return jmsTemplate;
    }

    @Bean
    public FrontDeskImpl frontDesk() {
        FrontDeskImpl frontDesk = new FrontDeskImpl();
        frontDesk.setJmsTemplate(jmsTemplate());
        return frontDesk;
    }
}
```

Klasa *BackOfficeImpl.java* sa podešenom podrazumevanom destinacijom za JMS poruke ima sledeći oblik:

```
@Configuration
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public Queue mailDestination() {
        return new ActiveMQQueue("mail.queue");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setDefaultDestination(mailDestination());
        jmsTemplate.setReceiveTimeout(10000);
        return jmsTemplate;
    }
}
```

```
@Bean
public BackOfficeImpl backOffice() {
    BackOfficeImpl backOffice = new BackOfficeImpl();
    backOffice.setJmsTemplate(jmsTemplate());
    return backOffice;
}
}
```

SLANJE I PRIJEM PORUKA SA PODRAZUMEVANIH DESTINACIJA.

Moguće je specificirati podrazumevanu destinaciju za JMS šablon.

Sa podrazumevanom destinacijom, definisanom u JMS šablonu, moguće je sada obrisati seter metodu za destinaciju poruke iz klasa pošiljaoca i prijemnika poruke. Sada, sa pozivom metoda `send()` i `receive()`, nije više potrebno specificirati odredište poruke.

Sledi listing izmenjene klase `FrontDeskImpl.java`.

```
public class FrontDeskImpl implements FrontDesk {

    private JmsTemplate jmsTemplate;

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    @Override
    public void sendMail(final Mail mail) {
        jmsTemplate.send(session -> {
            MapMessage message = session.createMapMessage();
            message.setString("mailId", mail.getMailId());
            message.setString("country", mail.getCountry());
            message.setDouble("weight", mail.getWeight());
            return message;
        });
        System.out.println("Pošiljka je poslata!!!");
    }
}
```

Neophodno je modifikovati i klasu koja redefiniše metodu za čitanje JMS poruka sa podrazumevane destinacije.

```
public class BackOfficeImpl implements BackOffice {

    private JmsTemplate jmsTemplate;

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }
}
```

```
}

@Override
public Mail receiveMail() {
    MapMessage message = (MapMessage) jmsTemplate.receive();
    try {
        if (message == null) {
            return null;
        }
        Mail mail = new Mail();
        mail.setMailId(message.getString("mailId"));
        mail.setCountry(message.getString("country"));
        mail.setWeight(message.getDouble("weight"));
        return mail;
    } catch (JMSEException e) {
        throw JmsUtils.convertJmsAccessException(e);
    }
}
}
```

Umesto specificiranja instance interfejsa *Destination* za JMS šablon, moguće je specificirati naziv odredišta kojeg će *JMS šablon* rešiti umesto nas, pa je moguće obrisati deklaraciju objekta *Destination* iz oba konfiguraciona fajla zrna. Ovo je omogućeno upotrebom osobine *defaultDestinationName*.

PRIJEM I SLANJE PORUKE SA PODRAZUMEVANIH DESTINACIJA - DEMO

Demonstracija slanja i prijema JMS poruka sa podrazumevanih destinacija.

Primer je ponovo redefinisan i sada upravlja prijemom i slanjem poruka sa podrazumevanih destinacija. Iz navedenog razloga, neophodno je testirati urađeni primer. Primer je, takođe, zadužen za upravljanje *JMS* porukama, sa *Spring* podrškom. Prvo će biti pokrenuta klasa *FrontDeskMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

Pošiljka je poslata!!!

BUILD SUCCESS

Total time: 4.617s

Finished at: Wed Nov 21 14:56:24 CET 2018

Slika 1.3.1 Slanje JMS poruke na podrazumevanu destinaciju [izvor: autor]

Konačno, pokreće se i klasa *BackOfficeMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

Pošiljka #1234 je primljena

BUILD SUCCESS

Slika 1.3.2 Čitanje JMS poruke sa podrazumevane destinacije [izvor: autor]

Urađen primer možete preuzeti iz aktivnosti Shared Resources na kraju objekta učenja.

▼ 1.4 Primena klase JmsGatewaySupport

OPCIJE PRIMENE KLASJE JMSGATEWAYSUPPORT

Postoje dve opcije po kojima klase mogu da naslede JmsGatewaySupport klasu.

JMS klase pošiljaoca i prijemnika poruke mogu da prošire klasu JmsGatewaySupport za prijem JMS šablona. Postoje dve opcije po kojima klase mogu da naslede JmsGatewaySupport klasu:

- Umetanje produkcije JMS konekcije u JmsGatewaySupport za automatsko kreiranje JMS šablona. U ovom slučaju nije moguće podešavati detalje JMS šablona;
- Umetanje korisnički kreiranog JMS šablona za JmsGatewaySupport.

Ukoliko se želi veća kontrola nad šablonima, drugi pristup je pogodniji. Moguće je obrisati privatno polje jmsTemplate i seter metodu iz obe klase, pošiljaoca i prijemnika. Kada se želi pristupiti JMS šablonu, potrebno je obaviti samo poziv metode getJmsTemplate().

Sledi izmenjeni kod klase FrontDeskImpl.java.

NASLEĐIVANJE KLASJE JMSGATEWAYSUPPORT

Kreirane implementacione klase nasleđuju JmsGatewaySupport.

JMS klase pošiljaoca i prijemnika poruke mogu da prošire klasu JmsGatewaySupport za prijem JMS šablona, kao što je istaknuto u prethodnoj sekciji.

Sledi izmenjeni kod klase FrontDeskImpl.java.

```
package com.metropolitan.frontdesk;  
  
import javax.jms.MapMessage;
```

```
import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl extends JmsGatewaySupport implements FrontDesk {

    @Override
    public void sendMail(final Mail mail) {
        getJmsTemplate().send(session -> {
            MapMessage message = session.createMapMessage();
            message.setString("mailId", mail.getMailId());
            message.setString("country", mail.getCountry());
            message.setDouble("weight", mail.getWeight());
            return message;
        });
        System.out.println ("Pošiljka je poslata!!!");
    }
}
```

Sledi izmenjeni kod klase *BackOfficeImpl.java*.

```
package com.metropolitan.backoffice;

import javax.jms.JMSException;
import javax.jms.MapMessage;
import org.springframework.jms.core.support.JmsGatewaySupport;
import org.springframework.jms.support.JmsUtils;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl extends JmsGatewaySupport implements BackOffice {

    @Override
    public Mail receiveMail() {
        MapMessage message = (MapMessage) getJmsTemplate().receive();
        try {
            if (message == null) {
                return null;
            }
            Mail mail = new Mail();
            mail.setMailId(message.getString("mailId"));
            mail.setCountry(message.getString("country"));
            mail.setWeight(message.getDouble("weight"));
            return mail;
        } catch (JMSException e) {
            throw JmsUtils.convertJmsAccessException(e);
        }
    }
}
```


PRIJEM I SLANJE PORUKE PRIMENOM JMSGATEWAYSUPPORT - DEMO

Demonstracija slanja i prijema JMS poruka primenom JmsGatewaySupport.

Konačno, neophodno je testirati urađeni primer. Primer je trenutno zadužen za upravljanje *JMS* porukama, sa *Spring* podrškom, primenom klase **JmsGatewaySupport**. Prvo će biti pokrenuta klasa *FrontDeskMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
--- exec-maven-plugin:1.2.1:exec
Pošiljka je poslata!!!
-----
BUILD SUCCESS
```

Slika 1.4.1 Slanje poruke primenom JmsGatewaySupport [izvor: autor]

Konačno, pokreće se i klasa *BackOfficeMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
1--- exec-maven-plugin:1.2.1:exec
Pošiljka #1234 je primljena
-----
BUILD SUCCESS
-----
```

Slika 1.4.2 Prijem poruke primenom JmsGatewaySupport [izvor: autor]

Urađeni primer možete preuzeti iz aktivnosti Shared Resources na kraju objekta učenja.

▼ Poglavlje 2

Konverzija JMS poruka

PROBLEM KONVERZIJE JMS PORUKA

Spring obezbeđuje implementaciju `SimpleMessageConvertor`.

Aplikacija prima poruke iz reda za poruke ali bi, takođe, trebalo da transformiše ove poruke iz **JMS specifičnog tipa** u specifičnu *poslovnu klasu*.

Spring obezbeđuje implementaciju **`SimpleMessageConvertor`** za rukovanje prevođenja *JMS poruke* u poslovni objekat i obratno.

Po osnovnim podešavanjima *JMS šablon* koristi **`SimpleMessageConvertor`** za konverziju **`TextMessage`** u **`String`** i obratno, **`BytesMessage`** u **`byte`** niz i obratno, **`MapMessage`** u **`Map`** i obratno, **`ObjectMessage`** u **`Serializable object`** i obratno. Za kreirane klase šaltera i prateće službe pošte, moguće je poslati i primiti mapu primenom metoda **`convertAndSend()`** i **`receiveAndConvert()`** i mapa će biti konvertovana u / iz **`MapMessage`**.

SLANJE I PRIJEM MAPE POMOĆU JMS ŠABLONA

*Moguće je poslati i primiti mapu primenom metoda **`convertAndSend()`** i **`receiveAndConvert()`**.*

Kao što je istaknuto u prethodnoj sekciji. Za kreirane klase šaltera i prateće službe pošte, moguće je poslati i primiti mapu primenom metoda **`convertAndSend()`** i **`receiveAndConvert()`** i mapa će biti konvertovana u / iz **`MapMessage`**.

Klasu odgovornu za slanje JMS poruka na navedeni način moguće modifikovati kao u slučaju sledećeg listinga. Slanje i konverzija mape u **`MapMessage`** obavljena je u liniji koda 19.

```
package com.metropolitan.frontdesk;

import java.util.HashMap;
import java.util.Map;
import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl extends JmsGatewaySupport implements FrontDesk {
```

```
@Override
public void sendMail(final Mail mail) {
    Map<String, Object> map = new HashMap<>();
    map.put("mailId", mail.getMailId());
    map.put("country", mail.getCountry());
    map.put("weight", mail.getWeight());
    getJmsTemplate().convertAndSend(map);
    System.out.println ("Pošiljka je poslata!!!");
}
}
```

Nakon modifikovanja klase *FrontDeskImpl*, sledi modifikacija klase *BackOfficeImpl.java*. Sledi listing sa modifikovanim kodom ove klase. Prijem i vraćanje mape iz *MapMessage* objekta je urađen u liniji 15 sledećeg listinga.

```
package com.metropolitan.backoffice;

import java.util.Map;
import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl extends JmsGatewaySupport implements BackOffice {

    @Override
    public Mail receiveMail() {
        @SuppressWarnings("unchecked")
        Map<String, Object> map = (Map<String, Object>)
getJmsTemplate().receiveAndConvert();
        Mail mail = new Mail();
        mail.setMailId((String) map.get("mailId"));
        mail.setCountry((String) map.get("country"));
        mail.setWeight((Double) map.get("weight"));
        return mail;
    }
}
```

KREIRANJE KONVERTERA JMS PORUKA

*Konverter poruka se kreira implementacijom interfejsa **MessageConverter**.*

U nastavku je zadatak kreiranje konvertera poruka. Da bi ovaj zadatak bio uspešno obavljen, neophodno je implementirati interfejs **MessageConverter** za konverziju objekata pošiljki.

Sledi listing kreirane klase *MailMessageConevrter.java*. Kreirana klasa je redefinisala dve metode: *fromMessage()* za dobijanje informacija iz poruke, u konkretnom slučaju iz objekta

poruke vraća *Mail* objekta, i *toMessage()*, koja radi suprotno, iz objekta kreira poruku. Ova klasa pripada podsistemu prateće službe i prikazana je sledećim listingom.

Ukoliko su odvojene i nezavisne aplikacije, za slanje i prijem poruka, ovu klasu je neophodno dodati i konfigurstai u oba projekta.

```
package com.metropolitan.frontdesk;

import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.Session;
import org.springframework.jms.support.converter.MessageConversionException;
import org.springframework.jms.support.converter.MessageConverter;

/**
 *
 * @author Vlada
 */
public class MailMessageConverter implements MessageConverter {

    @Override
    public Object fromMessage(Message message) throws JMSException,
        MessageConversionException {
        MapMessage mapMessage = (MapMessage) message;
        Mail mail = new Mail();
        mail.setMailId(mapMessage.getString("mailId"));
        mail.setCountry(mapMessage.getString("country"));
        mail.setWeight(mapMessage.getDouble("weight"));
        return mail;
    }

    @Override
    public Message toMessage(Object object, Session session) throws JMSException,
        MessageConversionException {
        Mail mail = (Mail) object;
        MapMessage message = session.createMapMessage();
        message.setString("mailId", mail.getMailId());
        message.setString("country", mail.getCountry());
        message.setDouble("weight", mail.getWeight());
        System.out.println ("Pošiljka je poslata!!!");
        return message;
    }
}
```

KONFIGURISANJE KREIRANOG JMS KONVERTERA

Konverter poruka je neophodno deklarirati u oba konfiguraciona fajla i umetnuti u JMS šablon.

Za primenu kreiranog konvertera poruka, neophodno ga je deklarirati u oba konfiguraciona fajla i umetnuti u *JMS šablon*. Prvim listingom je redefinisana konfiguraciona klasa za podsistem šaltera, a drugim za podsistem prateće službe.

```
@Configuration
public class FrontOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setDefaultDestinationName("mail.queue");
        jmsTemplate.setMessageConverter(mailMessageConverter());
        return jmsTemplate;
    }

    @Bean
    public MailMessageConverter mailMessageConverter() {
        return new MailMessageConverter();
    }

    @Bean
    public FrontDeskImpl frontDesk() {
        FrontDeskImpl frontDesk = new FrontDeskImpl();
        frontDesk.setJmsTemplate(jmsTemplate());
        return frontDesk;
    }
}
```

```
@Configuration
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
```

```

        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setDefaultDestinationName("mail.queue");
        jmsTemplate.setMessageConverter(mailMessageConverter());
        jmsTemplate.setReceiveTimeout(10000);
        return jmsTemplate;
    }

    @Bean
    public MailMessageConverter mailMessageConverter() {
        return new MailMessageConverter();
    }

    @Bean
    public BackOfficeImpl backOffice() {
        BackOfficeImpl backOffice = new BackOfficeImpl();
        backOffice.setJmsTemplate(jmsTemplate());
        return backOffice;
    }
}

```

IMPLEMENTACIJA KREIRANOG KONVERTERA PORUKA

Za primenu konvertera poruka, neophodno ga je deklarirati u konfiguraciji i umetnuti u JMS šablon.

Kada se eksplicitno podesi konverter poruka za JMS šablon, on će preklopiti podrazumevani **SimpleMessageConverter**. Sada je moguće pozvati metode `convertAndSend()` i `receiveAndConvert()` za prijem i slanje objekata poštanskih pošiljki.

Sledi listing inovirane klase *FrontDeskImpl.java*.

```

package com.metropolitan.frontdesk;

import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl extends JmsGatewaySupport implements FrontDesk {

    @Override
    public void sendMail(final Mail mail) {
        getJmsTemplate().convertAndSend(mail);
    }
}

```

Nakon ažurirane klase *FrontDeskImpl.java* sledi i listing ažurirane klase *BackOfficeImpl.java*.

```
package com.metropolitan.backoffice;

import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl extends JmsGatewaySupport implements BackOffice {

    @Override
    public Mail receiveMail() {
        return (Mail) getJmsTemplate().receiveAndConvert();
    }
}
```

KONVERZIJA JMS PORUKA - DEMO

Demonstracija primera sa podrškom konverziji JMS poruka

Sledi demonstracija inoviranog primera. Za početak, neka je modifikovana klasa *BackOfficeMain.java* na sledeći način:

```
package com.metropolitan.backoffice;

import com.metropolitan.backoffice.config.BackOfficeConfiguration;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

/**
 *
 * @author Vlada
 */
public class BackOfficeMain {

    public static void main(String[] args) {

        ApplicationContext context = new
AnnotationConfigApplicationContext(BackOfficeConfiguration.class);

        BackOffice backOffice = context.getBean(BackOffice.class);
        Mail mail = backOffice.receiveMail();
        System.out.println("JMS poruka je primljena!!!");
        System.out.println("ID pošiljke :" + mail.getMailId());
        System.out.println("Država u koju se šalje :" + mail.getCountry());
        System.out.println("Masa pošiljke :" + mail.getWeight() );

    }
}
```

Prvo će biti pokrenuta klasa *FrontDeskMain.java* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
1 --- exec-maven-plugin:1.2.1:exec
Pošiljka je poslata!!!
-----
BUILD SUCCESS
-----
Total time: 2.592s
```

Slika 2.1 Slanje JMS poruka primenom konvertera [izvor: autor]

Konačno, pokreće se i klasa *BackOfficeMain.java* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
--- exec-maven-plugin:1.2.1:exec (
JMS poruka je primljena!!!
ID pošiljke :1234
Država u koju se šalje :RS
Masa pošiljke :1.5
-----
BUILD SUCCESS
```

Slika 2.2 Prijem JMS poruka primenom konvertera [izvor: autor]

ZADATAK ZA SAMOSTALNI RAD 1

Pokušajte sami

1. Preuzmite primer koji je priložen kao aktivnost SharedResources uz prethodni objekat učenja;
2. Implementirajte izmene priložene u ovom objektu učenja;
3. Testirajte obe Main klase.

Ukoliko ne možete trenutno da se izborite sa ovim zadatkom, preuzmite i analizirajte urađeni primer na kraju ovog objekta učenja. Primer je priložen kao aktivnost SharedResources.

▼ Poglavlje 3

Upravljanje JMS transakcijama

JMS TRANSAKCIJE

Pokrivanje što većeg broja implementacija `TransactionManager` i povezivanje ponašanja u zrna.

Rad sa JMS porukama može biti podignut na još viši nivo - upotreba JMS transakcija tako da su slanje i prijem poruka transakcioni.

Moguće je koristiti istu strategiju kao i bile gde drugo u Springu: pokrivanje što većeg broja implementacija TransactionManager i povezivanje ponašanja u zrna. Kada se proizvodi ili koristi veći broj JMS poruka u jednoj metodi, u slučaju dešavanja greške, JMS poruke kreirane ili iskorišćene na odredištu mogu biti ostavljene u nekonzistentnom stanju. Neophodno je okružiti metodu transakcijom da bi se izbegao ovaj problem.

PRIMENA TRANSAKCIJA NA SLANJE I PRIJEM JMS PORUKA

Upravljanje JMS porukama je konzistentno sa drugim strategijama pristupa podacima.

U Springu, upravljanje JMS porukama je konzistentno sa drugim strategijama pristupa podacima. Na primer, moguće je obeležiti anotacijom @Transactional metodu koja zahteva upravljanje transakcijama.

Primenjeno na klasu podsistema šaltera, implementacija transakcija na slanje JMS poruke može biti obavljena na način prikazan sledećim listingom.

```
package com.metropolitan.frontdesk;

import org.springframework.transaction.annotation.Transactional;
import org.springframework.jms.core.support.JmsGatewaySupport;

/**
 *
 * @author Vlada
 */
public class FrontDeskImpl extends JmsGatewaySupport implements FrontDesk {

    @Transactional
```

```
@Override
public void sendMail(final Mail mail) {
    getJmsTemplate().convertAndSend(mail);
}
}
```

Primenjeno na klasu podsistema prateće službe, implementacija transakcija na prijem *JMS* poruke može biti obavljena na način prikazan sledećim listingom.

```
package com.metropolitan.backoffice;

import org.springframework.jms.core.support.JmsGatewaySupport;
import org.springframework.transaction.annotation.Transactional;

/**
 *
 * @author Vlada
 */
public class BackOfficeImpl extends JmsGatewaySupport implements BackOffice {

    @Transactional
    @Override
    public Mail receiveMail() {
        return (Mail) getJmsTemplate().receiveAndConvert();
    }
}
```

JAVA KONFIGURACIJE ZA PRIMENU JMS TRANSAKCIJA

Neophodno je ažurirati i konfiguracione klase za podršku transakcijama.

Obeležene su odgovarajuće metode, klasa *FrontDeskImpl.java* i *BackOfficeImpl.java*, anotacijom **@Transactional**.

Sledeće što je potrebno učiniti jeste da je, u obe konfiguracione klase, neophodno dodati anotaciju **@EnableTransactionManagement** za deklarisanje menadžera transakcija. Odgovarajući menadžer transakcija za lokalne *JMS transakcije* je **JmsTransactionManager**, koji zahteva referencu na *produkciju JMS konekcije*.

Sledi listing inovirane konfiguracione klase *FrontOfficeConfiguration.java*.

```
@Configuration
@EnableTransactionManagement
public class FrontOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }
}
```

```

@Bean
public JmsTemplate jmsTemplate() {
    JmsTemplate jmsTemplate = new JmsTemplate();
    jmsTemplate.setConnectionFactory(connectionFactory());
    jmsTemplate.setDefaultDestinationName("mail.queue");
    jmsTemplate.setMessageConverter(mailMessageConverter());
    return jmsTemplate;
}

@Bean
public MailMessageConverter mailMessageConverter() {
    return new MailMessageConverter();
}

@Bean
public PlatformTransactionManager transactionManager() {
    return new JmsTransactionManager(connectionFactory());
}

@Bean
public FrontDeskImpl frontDesk() {
    FrontDeskImpl frontDesk = new FrontDeskImpl();
    frontDesk.setJmsTemplate(jmsTemplate());
    return frontDesk;
}
}

```

Sledi listing inovirane konfiguracije klase *BackOfficeConfiguration.java*.

```

@Configuration
@EnableTransactionManagement
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jmsTemplate = new JmsTemplate();
        jmsTemplate.setConnectionFactory(connectionFactory());
        jmsTemplate.setDefaultDestinationName("mail.queue");
        jmsTemplate.setMessageConverter(mailMessageConverter());
        jmsTemplate.setReceiveTimeout(10000);
        return jmsTemplate;
    }

    @Bean
    public MailMessageConverter mailMessageConverter() {
        return new MailMessageConverter();
    }
}

```

```
@Bean
public PlatformTransactionManager transactionManager() {
    return new JmsTransactionManager(connectionFactory());
}

@Bean
public BackOfficeImpl backOffice() {
    BackOfficeImpl backOffice = new BackOfficeImpl();
    backOffice.setJmsTemplate(jmsTemplate());
    return backOffice;
}
}
```

PRIJEM I SLANJE PORUKE PRIMENOM JMS TRANSAKCIJA - DEMO

*Demonstracija slanja i prijema JMS poruka primenom
JmsGatewaySupport.*

Primer sada ima mogućnost upravljanja slanjem i prijemom *JMS poruka*, koristeći *JMS transakcije*. Prvo će biti pokrenuta klasa *FrontDeskMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
1 --- exec-maven-plugin:1.2.1:exec
Pošiljka je poslata!!!
-----
BUILD SUCCESS
-----
Total time: 2.592s
```

Slika 3.1 Slanje JMS poruka primenom transakcija [izvor: autor]

Konačno, pokreće se i klasa *BackOfficeMain* i rezultat izvršavanja je moguće uočiti na sledećoj slici.

```
--- exec-maven-plugin:1.2.1:exec (<
JMS poruka je primljena!!!
ID pošiljke :1234
Država u koju se šalje :RS
Masa pošiljke :1.5
-----
BUILD SUCCESS
```

Slika 3.2 Prijem JMS poruka primenom transakcija [izvor: autor]

ZADATAK ZA SAMOSTALNI RAD 2

Pokušajte sami

1. Preuzmite primer koji je priložen kao aktivnost *SharedResources* uz prethodni objekat učenja;
2. Implementirajte izmene priložene u ovom objektu učenja;
3. Testirajte obe Main klase.

Ukoliko ne možete trenutno da se izborite sa ovim zadatkom, preuzmite i analizirajte urađeni primer na kraju ovog objekta učenja. Primer je priložen kao aktivnost Shared Resources.

▼ Poglavlje 4

Kreiranje POJO-a baziranih na porukama u Springu

DEFINISANJE PROBLEMA OSLUŠKIVANJA PORUKA

Spring dozvoljava zrnima deklarisanim u IoC kontejneru da osluškuju JMS poruke.

Kada se poziva metoda `receive()` da bi korisnik `JMS_poruke` primio poruku, pozivajuća nit je blokirana sve dok je poruka ne postane dostupna. Nit ne može ništa drugo da radi, osim da čeka. Ovaj tip prijema poruka je poznat kao ***sinhronizovani prijem***, zato što aplikacija mora da sačeka poruku da stigne pre nego što završi konkretan zadatak.

Počevši od verzije `EJB 2.0` (`Enterprise JavaBeans`) koristi se nova komponenta pod nazivom `MDB` (`message-driven bean`) za asinhroni *prijem JMS poruka*. `EJB kontejner` može da osluškuje `JMS` poruke na odredištu i da pokreće `MDB` zrna da reaguju na ova zrna i aplikacija više ne mora da čeka na poruke. `MDB` mora da implementira oba interfejsa `javax.ejb.MessageDrivenBean` i `javax.jms.MessageListener` i da redefiniše sve metode životnog ciklusa (`ejbCreate` i `ejbRemove`). Nove mogućnosti su dobijene sa `EJB 3.0`, `MDB` može biti `POJO` koji implementira interfejs `MessageListener` i obeležen je anotacijom `@MessageDriven`.

Iako `MDB` može da osluškuje `JMS` poruke, on mora da bude angažovan u `EJB kontejneru` da bi funkcionisao. Alternativno, moguće je dodeliti iste sposobnosti `POJO`-u, da osluškuje `JMS` poruke, bez `EJB` kontejnera.

Spring dozvoljava zrnima deklarisanim u IoC kontejneru da osluškuju *JMS poruke*, na isti način kao što to rade `MDB` zrna. Razlog za to je što Spring dodeljuje sposobnosti osluškivanja zrna `POJO` - ima koja se zovu `MDP` (`message-driven POJO`).

KREIRANJE OSLUŠKIVAČA JMS PORUKA

Asinhroni prijem JMS poruka za izbegavanje blokiranja niti koja prima JMS poruke.

Prethodni primer biće proširen u svetlu aktuelne analize. Pretpostavka je da se želi dodati elektronska tabla za kreiranu prateću službu pošte za prikazivanje informacija u vezi sa pošiljkama, u realnom vremenu kako one stižu sa šaltera. Šalter šalje *JMS poruku* zajedno sa pošiljkom, podsistem prateće službe može da osluškuje ove poruke i da ih prikazuje

na elektronskoj tabli. Za bolje performanse sistema, moguće je koristiti pristup **asinhronog prijema JMS poruka** za izbegavanje blokiranja niti koja prima *JMS poruke*.

Prvo je neophodno kreirati osluškivač *JMS* poruka, na primer *MailListener* osluškuje poruke koje poseduju informacije o pošiljkama:

```
package com.metropolitan.backoffice;

import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.MessageListener;

import org.springframework.jms.support.JmsUtils;

/**
 *
 * @author Vlada
 */
public class MailListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        MapMessage mapMessage = (MapMessage) message;
        try {
            Mail mail = new Mail();
            mail.setMailId(mapMessage.getString("mailId"));
            mail.setCountry(mapMessage.getString("country"));
            mail.setWeight(mapMessage.getDouble("weight"));
            displayMail(mail);
        } catch (JMSException e) {
            throw JmsUtils.convertJmsAccessException(e);
        }
    }

    private void displayMail(Mail mail) {
        System.out.println("JMS poruka je primljena!!!");
        System.out.println("ID pošiljke : " + mail.getMailId());
        System.out.println("Država u koju se šalje : " + mail.getCountry());
        System.out.println("Masa pošiljke : " + mail.getWeight() );
    }
}
```

Kreirani osluškivač predstavlja alternativu pristupu koji je u prethodnom izlaganju prikazan, a koji se oslanjao na primenu klase *BackOfficeImpl.java* i *JMS šablona*. Osluškivač može biti kreiran tako da vrši ulogu potrošača JMS poruka koje obezbeđuje JMS broker. Na primer, kreirani osluškivač osluškuje JMS poruke koje sadrže informacije o predatoj pošiljci.

Veoma je važno da osluškivač implementira interfejs *javax.jms.MessageListener*. Kada se prihvata *JMS* poruka, metoda *onMessage()* (linija koda 17) poziva se, a njen argument je upravo poruka. Na ovaj način je moguće veoma jednostavno prikazati informacije o pošiljci na konzoli.

Takođe je neophodno napomenuti da je izvlačenje informacija poruke iz objekta tipa *MapMessage* praćeno rukovanjem *JMSException* izuzecima koji pripadaju standardnom JMS *API*. Moguće je obaviti poziv *JmsUtils.convertJmsAccessException()* (linija koda 26) za prevođenje izuzetaka navedenog tipa u *JmsException* koji pripadaju Spring okviru.

PODEŠAVANJE OSLUŠKIVAČA

Neophodno je izvršiti konfigurisanje osluškivača u konfiguracionim klasama.

Dalje, neophodno je konfigurisati osluškivače u konfiguracionom fajlu prateće službe.

```
package com.metropolitan.backoffice.config;

import com.metropolitan.backoffice.MailListener;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import javax.jms.ConnectionFactory;
import org.springframework.jms.listener.SimpleMessageListenerContainer;

/**
 *
 * @author Vlada
 */
@Configuration
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public MailListener mailListener() {
        return new MailListener();
    }

    @Bean
    public Object container() {
        SimpleMessageListenerContainer smlc = new SimpleMessageListenerContainer();
        smlc.setConnectionFactory(connectionFactory());
        smlc.setDestinationName("mail.queue");
        smlc.setMessageListener(mailListener());
        return smlc;
    }
}
```


Spring obezbeđuje nekoliko tipova osluškivača *JMS* poruka iz paketa *org.springframework.jms.listener* pomoću koji se koriste kontejneri *SimpleMessageListenerContainer* i *DefaultMessageListenerContainer*.

SimpleMessageListenerContainer je najjednostavniji i ne podržava rad sa transakcijama. Ukoliko programer želi da koristi transakcije za razmenu JMS poruka, moraće da koristi *DefaultMessageListenerContainer*.

OSLUŠKIVANJE JMS PORUKA - MAIN KLASA

Sada je moguće pokrenuti osluškivač poruka.

Sada je moguće pokrenuti osluškivač poruka u sledećoj glavnoj klasi, a to funkcioniše isključivo u Spring IoC kontejneru.

```
import com.metropolitan.backoffice.config.BackOfficeConfiguration;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

/**
 *
 * @author Vlada
 */
public class BackOfficeMain {

    public static void main(String[] args) {

        ApplicationContext context = new
        AnnotationConfigApplicationContext(BackOfficeConfiguration.class);

    }
}
```

Sada, kada se pokrene aplikacija prateće službe ona će vršiti osluškivanje brokera poruka, a to je u ovom slučaju *ActiveMQ*. Kada šalter pošalje poruku, i kada nju broker registruje, aplikacija prateće službe reaguje i prikazuje poruku, preuzetu sa brokera, na konzoli. Navedeno je prikazano sledećom slikom.

--- exec-maven-plugin:1.2.1:exec

```
JMS poruka je primljena!!!
ID pošiljke :1234
Država u koju se šalje :RS
Masa pošiljke :1.5
```

Slika 4.1 Poruka koju je osluškivač preuzeo od brokera [izvor: autor]

Moguće je još jednom istaći da osluškivač ovde upotrebljen kao i potrošač poruke.

ZADATAK ZA SAMOSTALNI RAD 3

Pokušajte sami

1. Preuzmite primer koji je priložen kao aktivnost SharedResources uz prethodni objekat učenja;
2. Implementirajte izmene priložene u ovom objektu učenja;
3. Testirajte obe Main klase.

Ukoliko ne možete trenutno da se izborite sa ovim zadatkom, preuzmite i analizirajte urađeni primer na kraju ovog objekta učenja. Primer je priložen kao aktivnost SharedResources - Lekcija11Primer7.

OSLUŠKIVANJE JMS PORUKA POMOĆU POJO-A

Proizvoljno zrno deklarirano u Spring IoC kontejneru, može da osluškuje poruke.

Kao što osluškivač poruka koji implementira interfejs *JMS MessageListener* može da osluškuje poruke, može i proizvoljno zrno deklarirano u *Spring IoC kontejneru*. To znači da su ova zrna razdvojena od *Spring IoC interfejsa*, baš kao i od interfejsa JMS *MessageListener*. Da bi metoda ovakvog zrna bila pokrenuta kada stigne poruka, mora da prihvati jedan od sledećih tipova:

- *String* - za objekat *TextMessage*;
- *Map* - za objekat *MapMessage*;
- *byte[]* - za objekat *BytesMessage*;
- *Serializable* - za objekat *ObjectMessage*.

Na primer, za osluškivanje *MapMessage*, neophodno je deklarirati metodu koja uzima mapu kao argument i obeležena je anotacijom *@JmsListener*. Ovaj osluškivač ne mora više da implementira *MessageListener* interfejs.

```
package com.metropolitan.backoffice;

import java.util.Map;
import org.springframework.jms.annotation.JmsListener;

/**
 *
```

```

* @author Vlada
*/
public class MailListener {

    @JmsListener(destination = "mail.queue")
    private void displayMail(Map map) {
        Mail mail = new Mail();
        mail.setMailId((String) map.get("mailId"));
        mail.setCountry((String) map.get("country"));
        mail.setWeight((Double) map.get("weight"));
        System.out.println("JMS poruka je primljena!!!");
        System.out.println("ID pošiljke : " + mail.getMailId());
        System.out.println("Država u koju se šalje : " + mail.getCountry());
        System.out.println("Masa pošiljke : " + mail.getWeight());
    }
}

```

Da bi anotacija `@JmsListener` mogla da bude otkrivena, neophodno je dodati anotaciju `@EnableJms` u konfiguracionu klasu, kao i izvršiti registrovanje zrna `JmsListenerContainerFactory`. Ovo zrno se detektuje pod podrazumevanim nazivom `jmsListenerContainerFactory`.

POJO objekat se registruje u kontejneru osluškivača preko `JmsListenerContainerFactory`. Ovaj interfejs kreira i podešava `MessageListenerContainer` i registruje obeležene metode kao metode koje se osluškuju.

PODEŠAVANJE POJO OSLUŠKIVANJA

Neophodno je dodati anotaciju `@EnableJms` u konfiguracionu klasu.

Kao što je istaknuto, da bi anotacija `@JmsListener` mogla da bude otkrivena, neophodno je dodati anotaciju `@EnableJms` u konfiguracionu klasu.

Za implementaciju interfejsa `JmsListenerContainerFactory` biće, zbog pojednostavljenja i lakšeg razumevanja problematike, upotrebljen podinterfejs `SimpleJmsListenerContainerFactory`. Ukoliko programer želi da koristi transakcije poslednji interfejs se lako menja primenom `DefaultMessageListenerContainer`.

Sledi modifikovani listing konfiguracione klase podsistema prateće službe.

```

package com.metropolitan.backoffice.config;

import com.metropolitan.backoffice.MailListener;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.config.SimpleJmsListenerContainerFactory;
import javax.jms.ConnectionFactory;
import org.springframework.jms.connection.CachingConnectionFactory;

```

```
/**
 *
 * @author Vlada
 */
@Configuration
@EnableJms
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public CachingConnectionFactory cachingConnectionFactory() {
        return new CachingConnectionFactory(connectionFactory());
    }

    @Bean
    public MailListener mailListener() {
        return new MailListener();
    }

    @Bean
    public SimpleJmsListenerContainerFactory jmsListenerContainerFactory() {
        SimpleJmsListenerContainerFactory listenerContainerFactory = new
SimpleJmsListenerContainerFactory();
        listenerContainerFactory.setConnectionFactory(cachingConnectionFactory());
        return listenerContainerFactory;
    }
}
```

KONVERZIJA JMS PORUKA

Moguće je kreirati konverter poruka za konverziju objekata iz JMS poruka.

Takođe, moguće je kreirati konverter poruka za konverziju objekata pošiljki iz *JMS poruka* koje sadrže informacije o pošiljkama.

```
package com.metropolitan.backoffice;
import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.Session;
import org.springframework.jms.support.converter.MessageConversionException;
import org.springframework.jms.support.converter.MessageConverter;
/**
 *
 * @author Vlada
```

```

*/
public class MailMessageConverter implements MessageConverter {

    @Override
    public Object fromMessage(Message message) throws JMSEException,
        MessageConversionException {
        MapMessage mapMessage = (MapMessage) message;
        Mail mail = new Mail();
        mail.setMailId(mapMessage.getString("mailId"));
        mail.setCountry(mapMessage.getString("country"));
        mail.setWeight(mapMessage.getDouble("weight"));
        return mail;
    }

    public Message toMessage(Object object, Session session) throws JMSEException,
        MessageConversionException {

        Mail mail = (Mail) object;
        MapMessage message = session.createMapMessage();
        message.setString("mailId", mail.getMailId());
        message.setString("country", mail.getCountry());
        message.setDouble("weight", mail.getWeight());
        return message;
    }
}

```

Konverter poruka bi trebalo da bude primenjen na adapter osluškivača za konverziju poruka u objekte pre poziva *POJO* metoda.

```

package com.metropolitan.backoffice.config;

import com.metropolitan.backoffice.MailListener;
import com.metropolitan.backoffice.MailMessageConverter;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.config.SimpleJmsListenerContainerFactory;
import javax.jms.ConnectionFactory;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableJms
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }
}

```

```

@Bean
public MailListener mailListener() {
    return new MailListener();
}

@Bean
public MailMessageConverter mailMessageConverter() {
    return new MailMessageConverter();
}

@Bean
public SimpleJmsListenerContainerFactory jmsListenerContainerFactory() {
    SimpleJmsListenerContainerFactory listenerContainerFactory = new
SimpleJmsListenerContainerFactory();
    listenerContainerFactory.setConnectionFactory(connectionFactory());
    listenerContainerFactory.setMessageConverter(mailMessageConverter());
    return listenerContainerFactory;
}
}

```

Sa ovim konverterom poruka, metoda osluškivač kreiranog *POJO*-a može prihvatiti objekat pošiljke kao argument poruke.

MODIFIKACIJA OSLUŠKIVAČA JMS PORUKA

Metoda osluškivač kreiranog POJO-a može prihvatiti objekat pošiljke kao argument poruke.

Kao što je istaknuto u prethodnoj sekciji, sa kreiranim konverterom poruka, metoda osluškivač kreiranog *POJO*-a može prihvatiti direktno objekat pošiljke kao argument poruke.

Sada je neophodno to i potvrditi modifikacijom prethodno kreiranog osluškivača, na sledeći način:

```

package com.metropolitan.backoffice;

import org.springframework.jms.annotation.JmsListener;

/**
 *
 * @author Vlada
 */
public class MailListener {

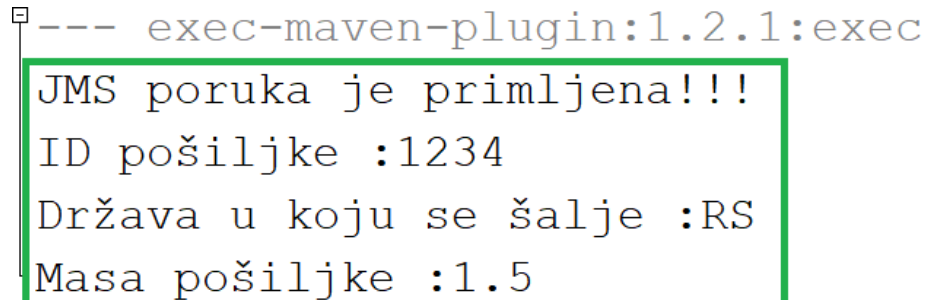
    @JmsListener(destination = "mail.queue")
    private void displayMail(Mail mail) {

        System.out.println("JMS poruka je primljena!!!");
        System.out.println("ID pošiljke :" + mail.getMailId());
    }
}

```

```
        System.out.println("Država u koju se šalje :" + mail.getCountry());  
        System.out.println("Masa pošiljke :" + mail.getWeight());  
    }  
}
```

Da je sve urađeno na pravi način pokazuje sledeća slika kojom je ilustrovan prijem JMS poruke na poslednje opisani način.



```
--- exec-maven-plugin:1.2.1:exec  
JMS poruka je primljena!!!  
ID pošiljke :1234  
Država u koju se šalje :RS  
Masa pošiljke :1.5
```

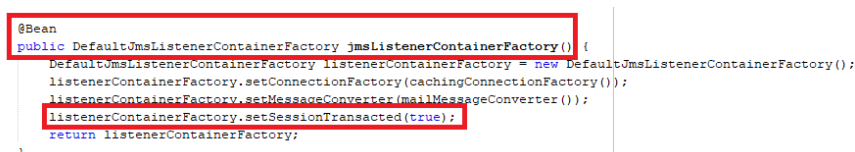
Slika 4.2 Prijem JMS poruke primenom konvertera i anotacija [izvor: autor]

UPRAVLJANJE JMS TRANSAKCIJAMA

Za upravljanje JMS transakcijama je neophodno koristiti `DefaultMessageListenerContainer`.

Kao što je već naglašeno, `SimpleMessageListenerContainer` ne podržava transakcije. U tu svrhu je neophodno koristiti `DefaultMessageListenerContainer`. Za lokalne JMS transakcije dovoljno je omogućiti njegovu osobinu `sessionTransacted`.

Izolovani kod kojim se registruje zrna tipa `DefaultMessageListenerContainer` u konfiguracionoj klasi podsistema prateće službe, prikazano je sledećom slikom.



```
@Bean  
public DefaultJmsListenerContainerFactory jmsListenerContainerFactory() {  
    DefaultJmsListenerContainerFactory listenerContainerFactory = new DefaultJmsListenerContainerFactory();  
    listenerContainerFactory.setConnectionFactory(cachingConnectionFactory());  
    listenerContainerFactory.setMessageConverter(mailMessageConverter());  
    listenerContainerFactory.setSessionTransacted(true);  
    return listenerContainerFactory;  
}
```

Slika 4.3 Registrovanje zrna DefaultMessageListenerContainer [izvor: autor]

Za širu sliku, prilaže se celokupan listing navedene konfiguracione klase.

```
package com.metropolitan.backoffice.config;  
  
import com.metropolitan.backoffice.MailListener;  
import com.metropolitan.backoffice.MailMessageConverter;  
import javax.jms.ConnectionFactory;  
  
import org.apache.activemq.ActiveMQConnectionFactory;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.jms.annotation.EnableJms;
```

```
import org.springframework.jms.config.DefaultJmsListenerContainerFactory;
import org.springframework.jms.connection.CachingConnectionFactory;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableJms
public class BackOfficeConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory("tcp://localhost:61616");
    }

    @Bean
    public CachingConnectionFactory cachingConnectionFactory() {
        return new CachingConnectionFactory(connectionFactory());
    }

    @Bean
    public MailListener mailListener() {
        return new MailListener();
    }

    @Bean
    public MailMessageConverter mailMessageConverter() {
        return new MailMessageConverter();
    }

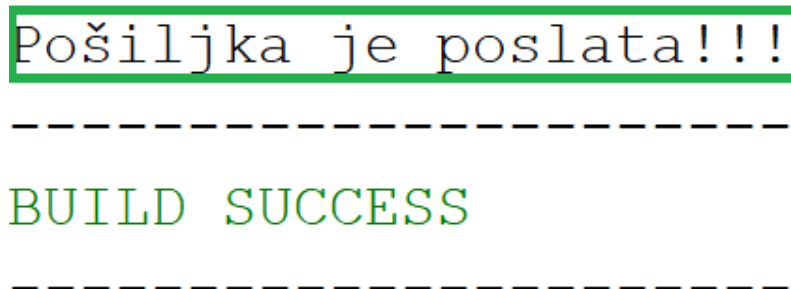
    @Bean
    public DefaultJmsListenerContainerFactory jmsListenerContainerFactory() {
        DefaultJmsListenerContainerFactory listenerContainerFactory = new
DefaultJmsListenerContainerFactory();
        listenerContainerFactory.setConnectionFactory(cachingConnectionFactory());
        listenerContainerFactory.setMessageConverter(mailMessageConverter());
        listenerContainerFactory.setSessionTransacted(true);
        return listenerContainerFactory;
    }
}
```

UPRAVLJANJE JMS TRANSAKCIJAMA - DEMO

Demonstracija primera nakon implementirane podrške transakcijama.

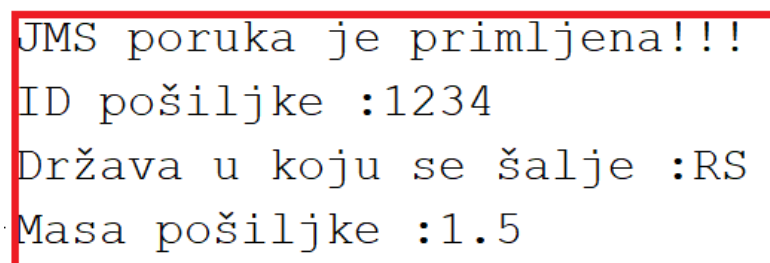
Konačno, primenu anotacija [@JmsListener](#) i [@EnableJms](#), kao i transakcija na način istaknut u prethodnom izlaganju i podržan prikazanim listinzima, neophodno je ilustrovati slikama izvršavanja aplikacije.

Prvo se pokreće, kao što je uvek i bio slučaj, aplikacija koja odgovara podsistemu šaltera pošte. Rezultat izvršavanja je prikazan sledećom slikom.



Slika 4.4 Slanje JMS poruke - demo [izvor: autor]

Konačno, krajnja verzija aplikacije podsistema prateće službe se pokreće i rezultati izvršavanja su prikazani sledećom slikom.



Slika 4.5 Prijem JMS poruke - demo [izvor: autor]

ZADATAK ZA SAMOSTALNI RAD 4

Pokušajte sami da implementirate nove izmene u JMS kodu.

1. Preuzmite primer koji je priložen kao aktivnost SharedResources uz prethodni objekat učenja;
2. Implementirajte izmene priložene u ovom objektu učenja;
3. Testirajte obe Main klase.

Ukoliko ne možete trenutno da se izborite sa ovim zadatkom, preuzmite i analizirajte urađeni primer na kraju ovog objekta učenja. Primer je priložen kao aktivnost SharedResources - Lekcija11Primer8.

▼ Poglavlje 5

Kreiranje konekcije

PROBLEM KONEKCIJE

ActiveMQ predstavlja alternativu konkretnoj produkciji konekcije.

U osnovi, neophodna je produkcija konekcije koja obezbeđuje povlačenje i keširanje prilikom publikovanja poruka pomoću *JmsTemplate*. Prvo mesto na kojem je moguće tražiti produkciju konekcije za povlačenje je aplikativni server, ako se koristi.

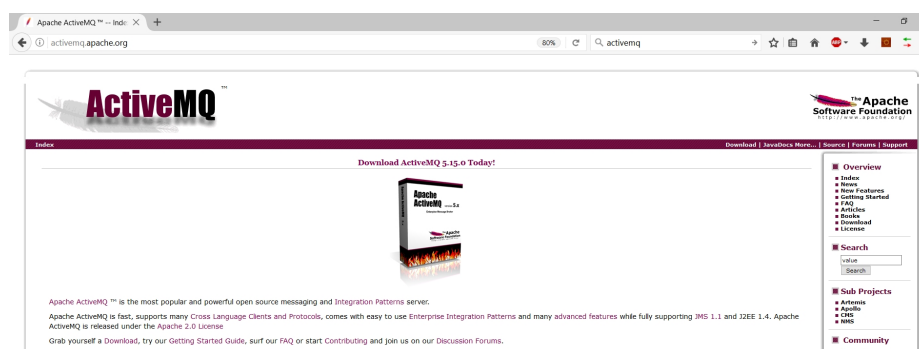
U konkretnom primeru, koristi se *ActiveMQ* koji predstavlja alternativu konkretnoj produkciji konekcije, koju je moguće koristiti umesto keširanja proizvođača poruka i sesija prilikom slanja poruka. Sledeća konfiguracija povlači produkciju konekcije u samostalnu konfiguraciju.

```
@Bean
public ConnectionFactory connectionFactory() {

    return new ActiveMQConnectionFactory("tcp://localhost:61616");

}
```

Više o *ApacheMQ* serveru, njegovom preuzimanju i instaliranju moguće je informisati se na linku <http://activemq.apache.org/>. Ukoliko se u veb pregledaču unese navedeni lik, otvoriće se stranica kao na sledećoj slici.



Slika 5.1 Početna stranica ApacheMQ sajta [izvor: autor]

▼ Poglavlje 6

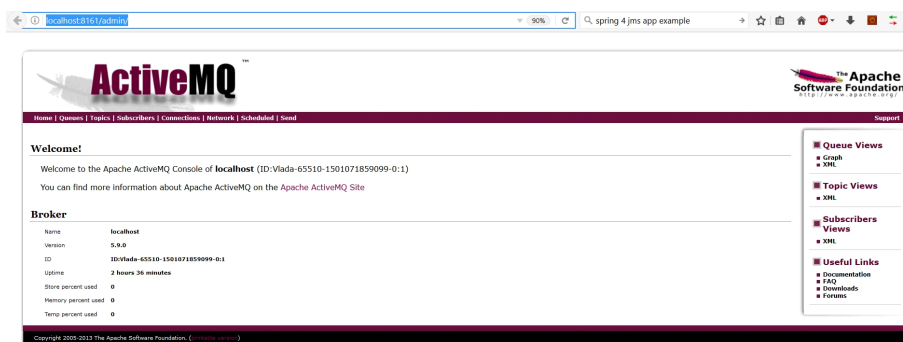
Pokazni primer - Spring 4 / 5 + JMS + ActiveMQ

SINERGIJA TEHNOLOGIJA I ALATA

Spring verzije 4 i 5 obezbeđuju prvoklasnu podršku za JMS.

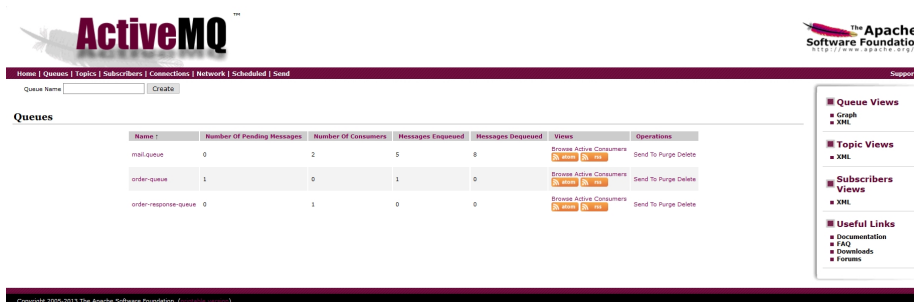
Spring 4 obezbeđuje prvoklasnu podršku za JMS (**Java Messaging System**). Posebno je značajno istaći da je veoma jednostavno i elegantno moguće kreirati *Spring MVC* aplikacije koje integrišu primenu najsavremenijih alata i tehnologija u sinergiji, poput: Spring4, JMS, ActiveMQ i pristupa baziranog na anotacijama.

Sve je jednostavno moguće pratiti na veb konzoli *ActiveMQ*, koja odgovara brokeru *JMS poruka*. Unosom linka u veb pregledač, otvara se pomenuta veb konzola kao na sledećoj slici.



Slika 6.1 ActiveMQ veb konzola [izvor: autor]

Na ovoj konzoli moguće je praćenje statusa slanja i prijema *JMS poruka*. Sledeća slika pokazuje upravo monitoring primera koji će biti kreiran kao podrška tekućem izlaganju.

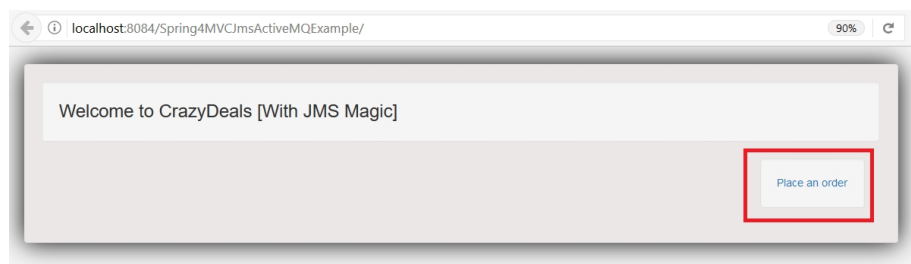


Slika 6.2 Monitoring prijema i slanja JMS poruka [izvor: autor]

IZGLED STRANICA POGLEDA

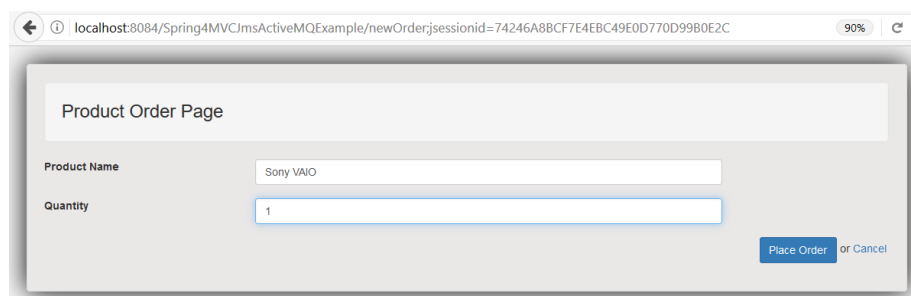
Sledi kratka navigacija kroz JMS aplikaciju

Prevođenjem i pokretanjem programa, na portu `http://localhost:8084`, otvara se početna stranica sa sledećim izgledom.



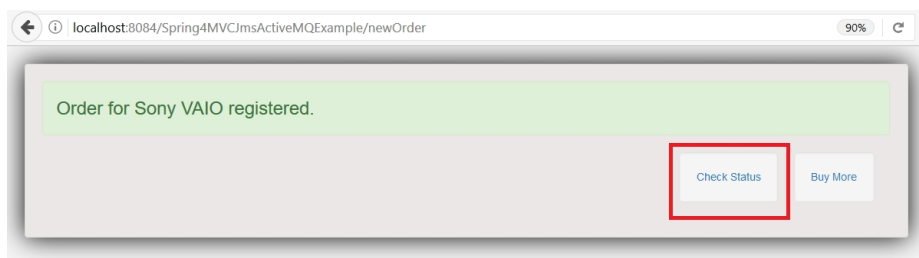
Slika 6.3 Početna stranica JMS aplikacije [izvor: autor]

Izborom opcije *Place an order* zahteva se kreiranje porudžbine kao na stranici prikazanoj sledećom slikom.



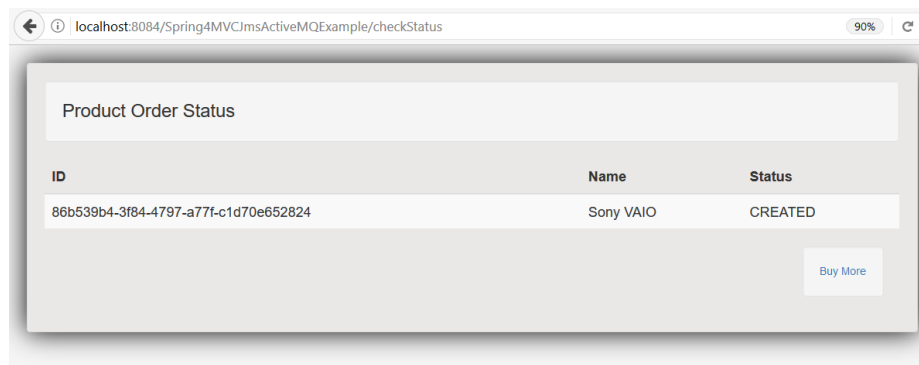
Slika 6.4 Kreiranje porudžbine [izvor: autor]

Sledeća stranica pokazuje listu izabranih proizvoda i daje izbor provere statusa.



Slika 6.5 Lista izabranih proizvoda [izvor: autor]

Klikom na dugme *Check Status* vrši se preusmeravanje na sledeću stranicu na kojoj mogu da se pogledaju svi naručeni proizvodi.



Slika 6.6 Status porudžbine [izvor: autor]

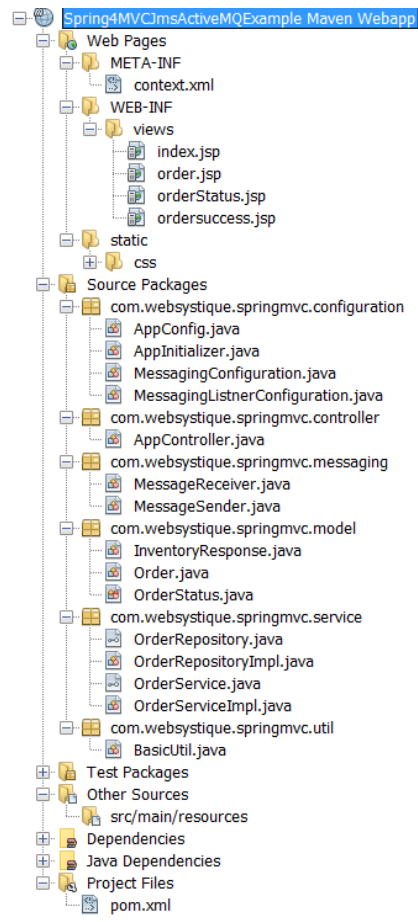
TEHNOLOGIJE I STRUKTURA PROJEKTA.

Pregled tehnologija i strukture keirane aplikacije.

Za kreiranje prezentovanog projekta biće korišćene sledeće tehnologije i alati:

1. *Spring 4.3.9.RELEASE;*
2. *Spring JMS 4.3.9.RELEASE;*
3. *ActiveMQ 5.9.0;*
4. *Maven 3*
5. *JDK 1.8*
6. *Tomcat 8*

Sledećom slikom je prikazan struktura projekta koji će biti demonstriran u ovom izlaganju.



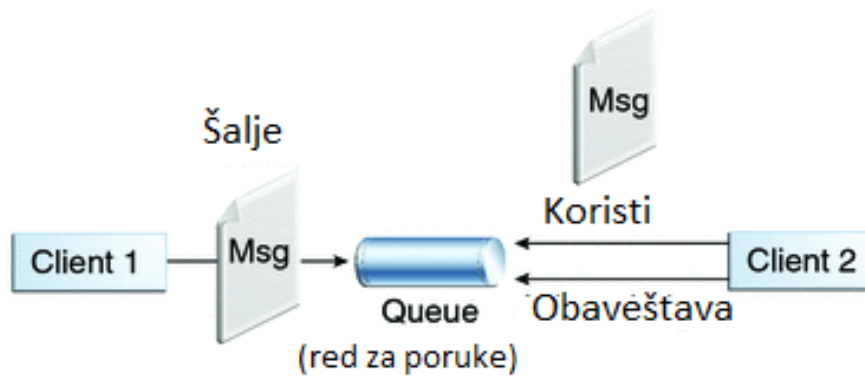
Slika 6.7 Struktura JMS projekta [izvor: autor]

KONFIGURACIONE DATOTEKE

Primer koristi Java konfiguracionu klasu.

I u ovom pokaznom primeru, akcenat će biti na novijem pristupu podešavanja konteksta aplikacije baziranom na primeni konfiguracionih anotacija.

Posebno je važno napomenuti da će poruke biti čuvane u redu za poruke (queue). Scenario sistemske razmene poruka, koji uključuje redove za poruke, može biti ilustrovan sledećom slikom.



Slika 6.8 JMS razmena poruka preko reda za poruke [izvor: autor]

Upravo će ovaj red, zajedno sa kontekstom aplikacije, i ostalim podešavanjima, biti definisan u Java konfiguracionim klasama. Prvo će biti prikazana klasa *AppConfig.java* koja je i osnovna konfiguraciona datoteka. Sledi njen listing.

```

package com.websystique.springmvc.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@ComponentScan(basePackages = "com.websystique.springmvc")
@Import({MessagingConfiguration.class, MessagingListnerConfiguration.class})
@EnableWebMvc
public class AppConfig extends WebMvcConfigurerAdapter{

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {

        InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        registry.viewResolver(viewResolver);
    }

    /*
     * Configure ResourceHandlers to serve static resources like CSS/ Javascript
    etc...
     */
}

```

```

    */
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/**").addResourceLocations("/static/");
    }
}

```

Sledi listing klase *MessageConfiguration* koja konfiguriše *red* za *poruke*, *ActiveMQConnectionFactory* i *JmsTemplate*.

```

package com.websystique.springmvc.configuration;

import java.util.Arrays;

import org.apache.activemq.spring.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.core.JmsTemplate;

@Configuration
public class MessagingConfiguration {

    private static final String DEFAULT_BROKER_URL = "tcp://localhost:61616";

    private static final String ORDER_QUEUE = "order-queue";

    @Bean
    public ActiveMQConnectionFactory connectionFactory(){
        ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory();
        connectionFactory.setBrokerURL(DEFAULT_BROKER_URL);

        connectionFactory.setTrustedPackages(Arrays.asList("com.websystique.springmvc"));
        return connectionFactory;
    }

    @Bean
    public JmsTemplate jmsTemplate(){
        JmsTemplate template = new JmsTemplate();
        template.setConnectionFactory(connectionFactory());
        template.setDefaultDestinationName(ORDER_QUEUE);
        return template;
    }
}

```

OSLUŠKIVANJE PORUKA

Neophodno je, u nastavku, podesiti osluškivanje poruka

Neophodno je, u nastavku, podesiti osluškivanje poruka. Moguće je koristiti standardni interfejs ***javax.jms.MessageListener***, međutim Spring daje dodatnu sposobnost podešavanja osluškivača primenom ***POJO*** klasa bez implementacije interfejsa. Da bi navedeno bilo moguće, neophodno je ispuniti sledeće:

1. Obeležiti konkretnu ***POJO*** metodu anotacijom ***@JmsListener***;
2. Podesiti ***kontejner za osluškivanje poruka*** da osluškuje destinaciju i kada poruka stigne na destinaciju, vraća poruku i prosleđuje zrna obeleženo anotacijom ***@JmsListener*** za tu destinaciju;
3. Koristiti anotaciju ***@EnableJms*** koja omogućava detekciju ***@JmsListener*** anotacija bilo kojeg Spring upravljanog zrna iz kontejnera.

Sledi kod konfiguracione klase za osluškivanje poruka.

```
package com.websystique.springmvc.configuration;

import javax.jms.ConnectionFactory;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.config.DefaultJmsListenerContainerFactory;

@Configuration
@EnableJms
public class MessagingListnerConfiguration {

    @Autowired
    ConnectionFactory connectionFactory;

    @Bean
    public DefaultJmsListenerContainerFactory jmsListenerContainerFactory() {
        DefaultJmsListenerContainerFactory factory = new
DefaultJmsListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        factory.setConcurrency("1-1");
        return factory;
    }
}
```

Nakon toga, prilaže se listing ***@JmsListener*** zrna.

```
package com.websystique.springmvc.messaging;

import javax.jms.JMSException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.annotation.JmsListener;
```

```
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageHeaders;
import org.springframework.stereotype.Component;

import com.websystique.springmvc.model.InventoryResponse;
import com.websystique.springmvc.service.OrderService;

@Component
public class MessageReceiver {
    static final Logger LOG = LoggerFactory.getLogger(MessageReceiver.class);

    private static final String ORDER_RESPONSE_QUEUE = "order-response-queue";

    @Autowired
    OrderService orderService;

    @JmsListener(destination = ORDER_RESPONSE_QUEUE)
    public void receiveMessage(final Message<InventoryResponse> message) throws
    JMSEException {
        LOG.info("+++++++");
        MessageHeaders headers = message.getHeaders();
        LOG.info("Application : headers received : {}", headers);

        InventoryResponse response = message.getPayload();
        LOG.info("Application : response received : {}", response);

        orderService.updateOrder(response);
        LOG.info("+++++++");
    }
}
```

SERVIS ZA SLANJE PORUKA

Sledi servis za slanje poruka.

Konačno, neophodno je konstruisati i servis za slanje poruka koji koristi **JmsTemplate**. Klasa se naziva **MessageSender.java** i priložena je sledećom listingom.

```
package com.websystique.springmvc.messaging;

import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.ObjectMessage;
import javax.jms.Session;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Component;
```

```
import com.websystique.springmvc.model.Order;

@Component
public class MessageSender {

    @Autowired
    JmsTemplate jmsTemplate;

    public void sendMessage(final Order order) {

        jmsTemplate.send(new MessageCreator(){
            @Override
            public Message createMessage(Session session) throws JMSException{
                ObjectMessage objectMessage =
session.createObjectMessage(order);
                return objectMessage;
            }
        });
    }
}
```

MAVEN DATOTEKA POM.XML

Za razvoj primera se koristi Maven alat za upravljanje zavisnostima.

Za razvoj primera se koristi *Maven* alat za upravljanje zavisnostima. Za razvoj Maven projekta, od ključnog značaja je precizno i kompletno definisana, dobro poznata, datoteka pom.xml.

Za ovaj pokazni primer sledi listing navedene datoteke sa svim korišćenim zavisnostima.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.websystique.springmvc</groupId>
    <artifactId>Spring4MVCJmsActiveMQExample</artifactId>
    <packaging>war</packaging>
    <version>1.0.0</version>
    <name>Spring4MVCJmsActiveMQExample Maven Webapp</name>

    <properties>
        <springframework.version>4.3.9.RELEASE</springframework.version>
    </properties>

    <dependencies>
        <dependency>
```

```

        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jms</artifactId>
        <version>${springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-spring</artifactId>
        <version>5.9.0</version>
    </dependency>
    <dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>1.1.0.Final</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.1.7</version>
    </dependency>
    <!-- Servlet+JSP+JSTL -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.1</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>

```

```

        <version>1.2</version>
    </dependency>

</dependencies>

<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <warSourceDirectory>src/main/webapp</warSourceDirectory>
                    <warName>Spring4MVCJmsActiveMQExample</warName>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
    <finalName>Spring4MVCJmsActiveMQExample</finalName>
</build>
</project>

```

KONTROLER I SERVIS ZA PORUDŽBINE

Sledi definicija servisa za poručivanje sa odgovarajućim kontrolerom.

Aplikacija je demonstrirala sistem za poručivanje proizvoda. Sledeći kontroler omogućava ovu funkcionalnost:

```

package com.websystique.springmvc.controller;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

```

```
import com.websystique.springmvc.model.Order;
import com.websystique.springmvc.service.OrderService;

@Controller
public class AppController {

    @Autowired
    OrderService orderService;

    @RequestMapping(value = { "/", "/home" }, method = RequestMethod.GET)
    public String prepareProduct(ModelMap model) {
        return "index";
    }

    @RequestMapping(value = { "/newOrder" }, method = RequestMethod.GET)
    public String prepareOrder(ModelMap model) {
        Order order = new Order();
        model.addAttribute("order", order);
        return "order";
    }

    @RequestMapping(value = { "/newOrder" }, method = RequestMethod.POST)
    public String sendOrder(@Valid Order order, BindingResult result,
        ModelMap model) {
        if (result.hasErrors()) {
            return "order";
        }
        orderService.sendOrder(order);
        model.addAttribute("success", "Order for " + order.getProductName() + "
registered.");
        return "ordersuccess";
    }

    @RequestMapping(value = { "/checkStatus" }, method = RequestMethod.GET)
    public String checkOrderStatus(ModelMap model) {
        model.addAttribute("orders", orderService.getAllOrders());
        return "orderStatus";
    }
}
```

Sledi kod interfejsa *OrderService.java* koji definiše metode za poručivanje.

```
package com.websystique.springmvc.service;

import java.util.Map;

import com.websystique.springmvc.model.InventoryResponse;
import com.websystique.springmvc.model.Order;

public interface OrderService {
    public void sendOrder(Order order);
}
```

```

    public void updateOrder(InventoryResponse response);

    public Map<String, Order> getAllOrders();
}

```

Sledi listing implementacione klase interfejsa *OrderService.java*.

```

package com.websystique.springmvc.service;

import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.websystique.springmvc.messaging.MessageSender;
import com.websystique.springmvc.model.InventoryResponse;
import com.websystique.springmvc.model.Order;
import com.websystique.springmvc.model.OrderStatus;
import com.websystique.springmvc.util.BasicUtil;

@Service("orderService")
public class OrderServiceImpl implements OrderService{

    static final Logger LOG = LoggerFactory.getLogger(OrderServiceImpl.class);

    @Autowired
    MessageSender messageSender;

    @Autowired
    OrderRepository orderRepository;

    @Override
    public void sendOrder(Order order) {
        LOG.info("++++++");
        order.setOrderId(BasicUtil.getUniqueId());
        order.setStatus(OrderStatus.CREATED);
        orderRepository.putOrder(order);
        LOG.info("Application : sending order request {}", order);
        messageSender.sendMessage(order);
        LOG.info("++++++");
    }

    @Override
    public void updateOrder(InventoryResponse response) {

        Order order = orderRepository.getOrder(response.getOrderId());
        if(response.getReturnCode()==200){
            order.setStatus(OrderStatus.CONFIRMED);
        }else if(response.getReturnCode()==300){
            order.setStatus(OrderStatus.FAILED);
        }else{

```

```
        order.setStatus(OrderStatus.PENDING);
    }
    orderRepository.putOrder(order);
}

public Map<String, Order> getAllOrders(){
    return orderRepository.getAllOrders();
}
}
```

INTERFEJS - REPOZITORIJUM

Definiše se repozitorijumski interfejs i odgovarajuća implementaciona klasa.

U nastavku izlaganja, razvoj primera teče kroz uvođenje repozitorijuma, još jednog interfejsa za upravljanje poručivanjem.

```
package com.websystique.springmvc.service;

import java.util.Map;

import com.websystique.springmvc.model.Order;

public interface OrderRepository {

    public void putOrder(Order order);

    public Order getOrder(String orderId);

    public Map<String, Order> getAllOrders();
}
```

Sledi implementaciona klasa kreiranog interfejsa **OrderRepository.java**.

```
package com.websystique.springmvc.service;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.springframework.stereotype.Service;

import com.websystique.springmvc.model.Order;

@Service("orderRepository")
public class OrderRepositoryImpl implements OrderRepository{

    private final Map<String, Order> orders = new ConcurrentHashMap<String, Order>();
}
```



```
@Override
public void putOrder(Order order) {
    orders.put(order.getId(), order);
}

@Override
public Order getOrder(String orderId) {
    return orders.get(orderId);
}

public Map<String, Order> getAllOrders(){
    return orders;
}
}
```

MODELSKE KLASKE

Sledi lista modelskih klasa.

Sledi lista modelskih klasa. Prva od njih će biti osnovna klasa *Order.java*. Sledi listing ove klase.

```
package com.websystique.springmvc.model;

import java.io.Serializable;

public class Order implements Serializable {

    private String orderId;

    private String productName;

    private int quantity;

    private OrderStatus status;

    public String getId() {
        return orderId;
    }

    public void setId(String orderId) {
        this.orderId = orderId;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
```

```
        this.productName = productName;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public OrderStatus getStatus() {
        return status;
    }

    public void setStatus(OrderStatus status) {
        this.status = status;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((orderId == null) ? 0 : orderId.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Order other = (Order) obj;
        if (orderId == null) {
            if (other.orderId != null)
                return false;
        } else if (!orderId.equals(other.orderId))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Order [orderId=" + orderId + ", productName=" + productName + ", quantity=" + quantity + ", status=" + status + "]\n";
    }
}
```

```
}
```

Sljede modelske klase za stanje porudžbine i odgovor za izabrane proizvode.

```
package com.websystique.springmvc.model;

public enum OrderStatus {

    CREATED("Created"),
    PENDING("Pending"),
    CONFIRMED("Confirmed"),
    FAILED("Failed");

    private String status;

    private OrderStatus(final String status){
        this.status = status;
    }

    public String getStatus(){
        return this.status;
    }

    @Override
    public String toString(){
        return this.status;
    }

    public String getName(){
        return this.name();
    }
}
```

```
package com.websystique.springmvc.model;

import java.io.Serializable;

public class InventoryResponse implements Serializable{

    private String orderId;
    private int returnCode;
    private String comment;

    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }
}
```

```

public int getReturnCode() {
    return returnCode;
}
public void setReturnCode(int returnCode) {
    this.returnCode = returnCode;
}
public String getComment() {
    return comment;
}
public void setComment(String comment) {
    this.comment = comment;
}
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((orderId == null) ? 0 : orderId.hashCode());
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    InventoryResponse other = (InventoryResponse) obj;
    if (orderId == null) {
        if (other.orderId != null)
            return false;
    } else if (!orderId.equals(other.orderId))
        return false;
    return true;
}
@Override
public String toString() {
    return "InventoryResponse [orderId=" + orderId + ", returnCode=" +
returnCode + ", comment=" + comment + "]";
}
}

```

BASICUTIL I APPINITIALIZER DATOTEKE

Izlaganje primera se završava ovim dvema datotekama.

Izlaganje primera se završava dvema datotekama *BasicUtil* i *AppInitializer*. Prva klasa daje metodu za slučajno generisanje ključeva *getUniqueld()*. Priložena je sledećim listingom.

```
package com.websystique.springmvc.util;

import java.util.UUID;

public class BasicUtil {

    public static String getUniqueId(){
        return UUID.randomUUID().toString();
    }
}
```

Druga klasa inicijalizuje kontekst aplikacije i nasleđuje klasu **AbstractAnnotationConfigDispatcherServletInitializer**. Priložena je sledećim listingom

```
package com.websystique.springmvc.configuration;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletIni
tializer;

public class AppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

Zbog složenosti primera, ovde su prikazane osnovne datoteke, napomene i izvršena je demonstracija funkcionalnosti aplikacije. Kompletan primer je priložen na kraju ovog objekta učenja kao aktivnost Shared Resources.

VIDEO MATERIJAL

Spring(Boot), JMS, ActiveMQ rezime uz video materijale.

Spring Boot ActiveMQ Example: 9 :16

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

Spring Boot JMS Tutorial - JmsTemplate JmsListener with ActiveMQ Example & JAXB:
21:17

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

▼ Poglavlje 7

Pokazna vežba 11

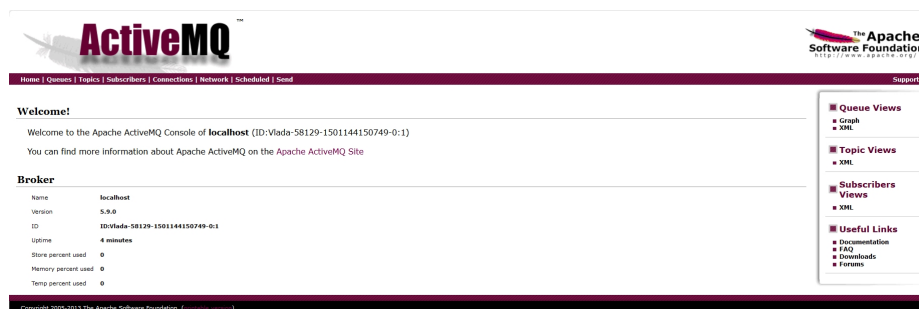
INSTALACIJA ACTIVEMQ SERVERA (TRAJANJE VEŽBE 45 MIN)

Kako bi se olakšalo praćenje poruka koje se šalju korišćenjem JMS okvira, za Spring je preporučljivo koristiti ActiveMQ broker kako bi se kreirala konekcija i vršilo praćenje poruka.

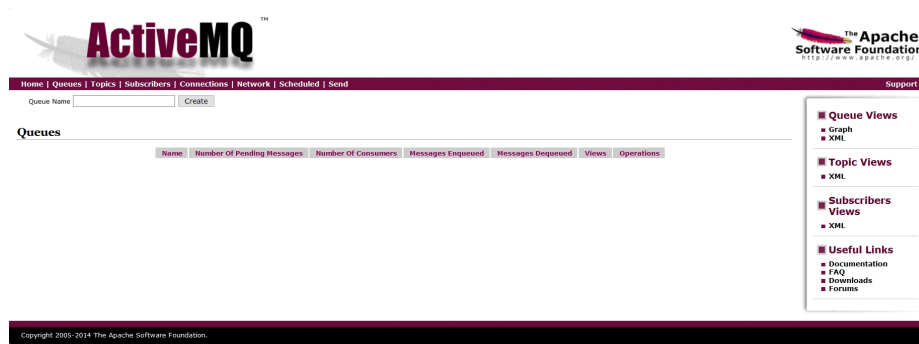
ActiveMQ predstavlja brokera poruka dostupnog na <tcp://localhost:61616> uz dostupnost konzole za praćenje JMS poruka na <http://localhost:8161/admin>. Korisničko ime za pristup konzoli i odgovarajuća lozinka dati su sa: *admin*, *admin*, respektivno.

Za instalaciju *ActiveMQ* servera je neophodno izvesti sledeće korake:

1. Preuzimanje *ActiveMQ* sa sledećeg linka <http://activemq.apache.org/activemq-5111-release.html>;
2. Raspakivanje preuzetog paketa u željeni folder;
3. U *ActiveMQ* folderu otvoriti podfolder */bin* i pokrenuti fajl *activemq.bat*;
4. Server je pokrenut;
5. Otvoriti veb pregledač i uneti link <http://localhost:8161/admin>;
6. Ulogovati se sa navedenim korisničkim imenom i lozinkom;
7. Sve je spremno za praćenje JMS poruka.



Slika 7.1 Početna strana ActiveMQ kozole [izvor: autor]



Slika 7.2 Stranica za praćenje redova za poruke [izvor: autor]

MAVEN JMS PROJEKAT

Cilj je da student na pokaznom primeru vidi kako se kreira i konfigurise aplikacija za rad sa JMS okvirom korišćenjem ActiveMQ brokera.

Kreirati novi Maven Web projekat prema uputstvima sa prethodnih vežbi i nazvati ga *IT355Veyba11*. Prvo treba da u *pom.xml* fajlu, koji se nalazi u *Project files* folderu pod *properties* tagom dodamo sledeće linije koda:

```
<spring-framework.version>5.1.0.RELEASE</spring-framework.version>
```

Nakon toga dodaje se lista zavisnosti, kao u sledećem listingu.

```
<dependencies>
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>7.0</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring-framework.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>${spring-framework.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.15.0</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
```



```

        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.6.2</version>
    </dependency>
</dependencies>

```

DOVRŠAVANJE POM.XML DEFINICIJE

Cilj je da studenti na pokaznom primeru vide kako se konfiguriše Spring MVC projekat putem anotacija

Kako bismo konfigurisali projekat putem anotacija prvo dodajemo pod *build* tagom u *pom.xml* fajlu sledeće linije koda:

```

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <compilerArguments>
                    <endorseddirs>${endorsed.dir}</endorseddirs>
                </compilerArguments>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.3</version>
            <configuration>
                <failOnMissingWebXml>>false</failOnMissingWebXml>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>2.6</version>
            <executions>
                <execution>
                    <phase>validate</phase>
                    <goals>
                        <goal>copy</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

```

        <configuration>
            <outputDirectory>${endorsed.dir}</outputDirectory>
            <silent>true</silent>
            <artifactItems>
                <artifactItem>
                    <groupId>javax</groupId>
                    <artifactId>javaee-endorsed-api</artifactId>
                    <version>7.0</version>
                    <type>jar</type>
                </artifactItem>
            </artifactItems>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>

</project>

```

KREIRANJE KONFIGURACIONIH KLASA

Sledi definisanje konfiguracionih klasa

Prvom datotekom će biti definisan kontekst aplikacije. Klasa nosi naziv *ApplicationInitializer.java* i priložena je sledećim listingom.

```

package com.metropolitan.it355vezball.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import
org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

/**
 *
 * @author Vladimir Milicevic
 */
public class ApplicationInitializer implements WebApplicationInitializer {

    public void onStartUp(ServletContext container) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new
AnnotationConfigWebApplicationContext();
        ctx.register(JmsConfig.class);
        ctx.setServletContext(container);
        ServletRegistration.Dynamic servlet = container.addServlet("dispatcher",
new DispatcherServlet(ctx));
    }
}

```

```
servlet.setLoadOnStartup(1);  
servlet.addMapping("/");  
}  
}
```

Druga klasa je obeležena anotacijama **@Configuration** i **@EnableJms**, nosi naziv *JmsConfig.java* i priložena je sledećim listingom.

```
package com.metropolitan.it355vezba11.config;  
  
import org.apache.activemq.ActiveMQConnectionFactory;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.jms.annotation.EnableJms;  
import org.springframework.jms.config.DefaultJmsListenerContainerFactory;  
import org.springframework.jms.core.JmsTemplate;  
  
/**  
 *  
 * @author Vladimir Milicevic  
 */  
@Configuration  
@EnableJms  
@ComponentScan(basePackages = "com.metropolitan.it355vezba11")  
public class JmsConfig {  
  
    String BROKER_URL = "tcp://localhost:61616";  
    String BROKER_USERNAME = "admin";  
    String BROKER_PASSWORD = "admin";  
  
    @Bean  
    public ActiveMQConnectionFactory connectionFactory() {  
        ActiveMQConnectionFactory connectionFactory = new  
ActiveMQConnectionFactory();  
        connectionFactory.setBrokerURL(BROKER_URL);  
        connectionFactory.setPassword(BROKER_USERNAME);  
        connectionFactory.setUserName(BROKER_PASSWORD);  
        return connectionFactory;  
    }  
  
    @Bean  
    public JmsTemplate jmsTemplate() {  
        JmsTemplate template = new JmsTemplate();  
        template.setConnectionFactory(connectionFactory());  
        return template;  
    }  
  
    @Bean  
    public DefaultJmsListenerContainerFactory jmsListenerContainerFactory() {  
        DefaultJmsListenerContainerFactory factory = new  
DefaultJmsListenerContainerFactory();  
        factory.setConnectionFactory(connectionFactory());  
    }  
}
```

```

        factory.setConcurrency("1-1");
        return factory;
    }
}

```

JMS OSLUŠKIVAČ

Kreira se klasa za osluškivanje JMS poruka

Anotacijom **@JmsListener** je obeležena metoda koja je ciljana od strane JMS osluškivača poruka za specificiranu destinaciju. U konkretnom slučaju destinacija nosi naziv *inbound.queue*. Klasa je odgovorna za osluškivanje i procesiranje poruke sa navedene destinacije. Klasa je priložena sledećim listingom.

```

package com.metropolitan.it355vezball.jms;

import org.springframework.stereotype.Component;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.TextMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.annotation.JmsListener;

/**
 *
 * @author Vladimir Milicevic
 */
@Component
public class Listener {

    @Autowired
    private Producer producer;

    @JmsListener(destination = "inbound.queue")
    public void receiveMessage(final Message jsonMessage) throws JMSException {
        String messageData = null;
        System.out.println("Received message " + jsonMessage);
        if (jsonMessage instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) jsonMessage;
            messageData = textMessage.getText();
        }
        producer.sendMessage("outbound.queue", messageData);
    }
}

```

JMS PROIZVOĐAČ

Kreira se klasa koja produkuje JMS poruke

U sledećem koraku je neophodno obezbediti producenta JMS poruka. Klasa će da parsira *Json* poruke i izvlači vrednost na osnovu ključa i šalje pozdravnu poruku na red. Sledi listing klase.

```
package com.metropolitan.it355vezball1.jms;

import com.google.gson.Gson;
import java.util.Map;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Component;

/**
 *
 * @author Vladimir Milicevic
 */
@Component
public class Producer {

    @Autowired
    JmsTemplate jmsTemplate;

    public void sendMessage(final String queueName, final String message) {
        Map map = new Gson().fromJson(message, Map.class);
        final String textMessage = "Hello" + map.get("name");
        System.out.println("Sending message " + textMessage + "to queue - " +
queueName);
        jmsTemplate.send(queueName, new MessageCreator() {

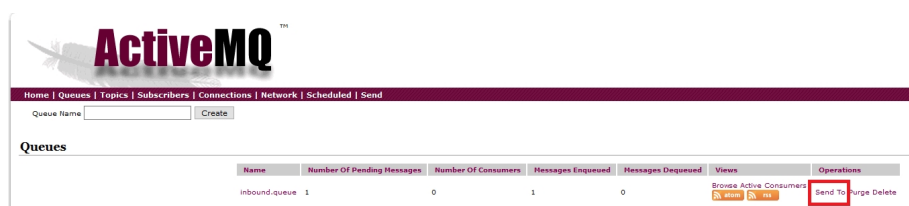
            public Message createMessage(Session session) throws JMSException {
                TextMessage message = session.createTextMessage();
                return message;
            }

        });
    }
}
```

POKRETANJE APLIKACIJE

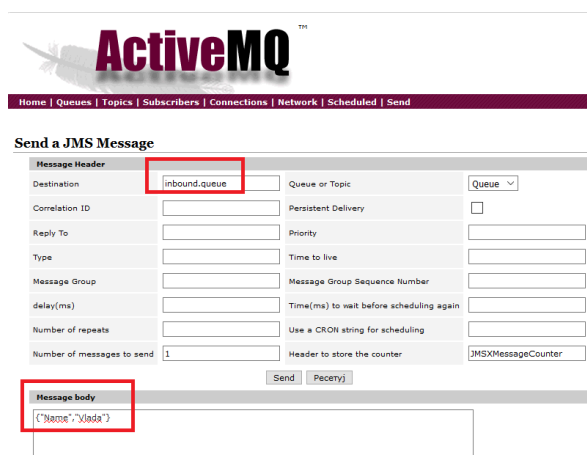
Aplikacija se pokreće i prate se JMS poruke

Aplikacija se prevodi, zajedno sa zavisnostima (**build with dependencies**) I red *inbound.queue* će automatski biti kreiran. Red je prikazan na slede'oj slici, u okviru veb konzole *ActiveMQ*.



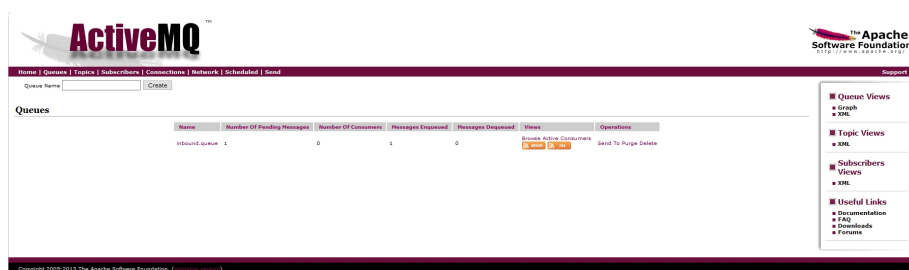
Slika 7.3 Kreirani red za poruke [izvor: autor]

Klikom na opciju *Send To* u veb konzoli je omogućeno slanje JMS poruke. Navedeno je prikazano sledećom slikom.



Slika 7.4 Kreiranje nove JMS poruke [izvor: autor]

Kreirana poruka se šalje na red za poruke i čeka da bude prosleđena JMS potrošaču. Navedeno je prikazano sledećom slikom.



Slika 7.5 Poslata poruka u redu za poruke [izvor: autor]

Kompletno urađen i testiran primer možete preuzeti iz aktivnosti Shared Resources odmah iza OU Vežbe 11.

▼ Poglavlje 8

Individualna vežba 11

INDIVIDUALNA VEŽBA (90 MIN)

Pokušajte sami

Samostalno podesite *ActiveMQ* server prateći sledeće korake:

1. Preuzimanje *ActiveMQ* sa sledećeg linka <http://activemq.apache.org/activemq-5111-release.html>;
2. Raspakivanje preuzetog paketa u željeni folder;
3. U *ActiveMQ* folderu otvoriti podfolder */bin* i pokrenuti fajl *activemq.bat*;
4. Server je pokrenut;
5. Otvoriti veb pregledač i uneti link <http://localhost:8161/admin>;
6. Ulogovati se sa navedenim korisničkim imenom i lozinkom;
7. Sve je spremno za praćenje JMS poruka.

Nakon podešavanja servera obavite sledeće zadatke:

1. Preuzmite primer koji je priložen uz Vežbe;
2. Pokušajte sami da kreirate JSP stranice za monitoring JMS poruka;
3. Pokrenite i testirajte inovirani projekat.

▼ Poglavlje 9

Domaći zadatak 11

DOMAĆI ZADATAK BROJ 11 (120 MIN)

Cilj ovog domaćeg zadatka je da student provežba naučeno u lekciji.

Na aktuelni projekat, koji razvijate kroz prethodne domaće zadatke implementirajte sledeće:

- kreirajte destinaciju za poruke;
- kreirajte osluškivače i mehanizme za slanje poruka;
- poruke su JSON stringovi objekata koji perzistiraju u bazi podataka;
- ukoliko imate poteškoća oko izrade domaćeg zadatka, konsultujte OU "Pokazna vežba";
- obezbedite monitoring rada sa porukama kroz posebno kreirane JMS stranice.

Domaći zadatak okačiti na GitHub kao Commit na repozitorijum koji ste napravili u prethodnom radu. Naslov commit-a treba da bude IT355-DZ11-Ime-Prezime-BrojIndeksa. Link ka GitHub repozitorijumu poslati u mejlu predmetnom asistentu sa naslovom IT355-DZ11

Ideju za rešenje domaćeg zadatka možete potražiti na linku: <https://grokonez.com/java-integration/spring-jms-activemq-send-java-object-messages-activemq-server-specially-bi-directional-relationship-java-objects>

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

▼ Poglavlje 10

Zaključak

ZAKLJUČAK

Obrađen je mehanizam primene JMS sistema poruka u Spring okviru.

Lekcija 11 je imala zadatak da detaljno obradi osnove procesa razmene poruka u Spring okviru. Upravo, prvi deo lekcije se bavio slanjem i prijemom JMS poruka u Springu sa akcentom na dva scenarija: prijem i slanje JMS poruka bez Spring podrške, a zatim prijem i slanje JMS poruka sa Spring podrškom. Osnovna ideja jeste, pored savladavanja koncepta JMS, bila je i da se pokaže u kolikoj meri se pojednostavljuje problematika prijema i slanja poruka, uz smanjenje količine potrebnog programskog koda Spring podrškom za manipulisanje JMS porukama. Posebno, u ovom delu lekcije je elaborirano da je za slanje i prijem poruka, neophodno je izvesti sledeće zadatke:

- Kreiranje JMS (produkcije konekcije) *connection factory*;
- Kreiranje JMS destinacije;
- Otvaranje JMS konekcije pomoću produkcije konekcije;
- Slanje ili prijem JMS poruke pomoću proizvođača ili korisnika poruke;
- Upravljanje izuzecima tipa *JMSException*;
- Zatvaranje JMS sesije i konekcije.

U nastavku, lekcija se bavila mehanizmima konverzije JMS poruka. Spring obezbeđuje implementaciju *SimpleMessageConvertor* za rukovanje prevođenja JMS poruke u poslovni objekat i obratno.

Poseban deo lekcije je obrađivao temu upravljanja JMS transakcijama. Kada se proizvodi ili koristi veći broj JMS poruka u jednoj metodi, u slučaju dešavanja greške, JMS poruke kreirane ili iskorišćene na odredištu mogu biti ostavljene u nekonzistentnom stanju. Neophodno je okružiti metodu transakcijom da bi se izbegao ovaj problem. U Springu, upravljanje JMS porukama je konzistentno sa drugim strategijama pristupa podacima. Na primer, moguće je obeležiti anotacijom *@Transactional* metodu koja zahteva upravljanje transakcijama.

Dalje, lekcija se fokusira na kreiranje POJO-a baziranih na porukama u Spring okviru. Ovde su razmatrane teme poput: osluškivanja JMS poruka, konverzija JMS poruka u novom, specifičnom kontekstu, upravljanje JMS transakcijama u ovom kontekstu, da bi, izlaganje bilo zaokruženo primenom Spring JMS šeme. Lekcija završava izlaganje analizom problema konekcije.

LITERATURA

U pripremanju Lekcije 10 korišćena je najaktuelnija pisana i web literatura.

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura:

1. Marten Deinum, Josh Long, and Daniel Rubio, Spring 5 Recipes Fourth Edition, Apress
2. Spring Framework Reference Documentation - <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/>
3. Craig Walls, Spring in Action, Manning
4. Craig Walls, Spring Boot in Action, Manning

Dopunska literatura:

1. <http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/>
2. <http://www.tutorialspoint.com/spring/>
3. <http://www.javatpoint.com/spring-tutorial>