



IT355 - WEB SISTEMI 2

Spring ORM

Lekcija 08

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 08

SPRING ORM

- ✓ Spring ORM
- ✓ Poglavlje 1: Problem direktnog korišćenja ORM okvira
- ✓ Poglavlje 2: JPA i Hibernate
- ✓ Poglavlje 3: Podešavanje produkcije ORM resursa u Springu
- ✓ Poglavlje 4: Hibernate kontekstualne sesije
- ✓ Poglavlje 5: JPA umetanje konteksta
- ✓ Poglavlje 6: Pokazna vežba 1 - Spring 4/5, Hibernate 5 i JPA 2
- ✓ Poglavlje 7: Pokazna vežba 2 - Spring 4/5, Hibernate 5 bez XML
- ✓ Poglavlje 8: Individualna vežba 8 - Spring MVC i Hibernate
- ✓ Poglavlje 9: Domaći zadatak 8
- ✓ 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

Ova lekcija će se baviti Spring mehanizmima za jednostavniji pristup podacima iz baze podataka.

Spring nudi viši nivo podrške za upravljanje bazama podataka poznat kao *objektno - relaciono mapiranje* (eng. *object/relational mapping (ORM)*). *Spring* podržava brojne savremene ORM okvire, međutim u ovoj lekciji će akcenat biti na *Hibernate ORM* i *JPA (Java Persistence API)*. ORM je savremena tehnologija čuvanja podataka u bazama podataka. ORM okvir čuva podatke na osnovu mapiranja obezbeđenih meta podataka (na primer, XML ili baziranih na anotacijama), poput mapiranja klasa i tabela, osobina i kolona, i tako dalje. ORM generiše SQL iskaze za perzistentne objekte tokom vremena izvršavanja i na taj način nije neophodno pisati specifične SQL iskaze baze podataka, ukoliko nije potrebno u potpunosti iskoristiti prednosti specifičnih alata baza podataka ili obezbediti vlastitu optimizaciju SQL iskaza. Kao rezultat, kreirana aplikacija je nezavisna od baze podataka i veoma lako može da se, u nekom budućem vremenu, usmeri ka drugoj bazi podataka. U poređenju sa direktnom primenom JDBC, ORM može značajno da umani aktivnosti pristupa podacima u kreiranoj web aplikaciji. *Hibernate ORM* je veoma popularan, *open - source* okvir visokih performansi, opšte prihvaćen u JAVA zajednici. *Hibernate* podržava većinu JDBC baza podataka i da koristi specifične dijalekte za pristup bazama podataka. Pored osnovnih ORM funkcionalnosti, *Hibernate* podržava i neke napredne poput: keširanja, kaskada i lenjog učitavanja. Takođe, donosi i podršku za vlastiti jezik upita poznatiji kao *HQL (Hibernate Query Language)*. *JPA* definiše skup standardnih anotacija i API - ja za perzistenciju objekata na platformama *JAVA SE* i *JAVA EE*.

JPA predstavlja deo *EJB 3.0* specifikacije u *JSR-220*. *JPA* je skup standardnih API - ja, koji traže odgovarajući *JPA* mehanizam za omogućavanje servisa perzistencije. *JPA* se može uporediti sa *JDBC API*, a *JPA* mehanizam sa *JDBC* drajverom. *Hibernate* može biti podešen kao *JPA* mehanizam kroz proširujući modul poznat pod nazivom *Hibernate EntityManager*.

Upravo, lekcija će dati glavni akcenat na *JPA* sa *Hibernate* mehanizmom za podršku.

▼ Poglavlje 1

Problem direktnog korišćenja ORM okvira

PRELAZAK NA VIŠI NIVO UPRAVLJANJA BAZOM PODATAKA

Spring poseduje instrumente za Hibernate primenu u radu sa bazama podataka.

Proširivanjem i usvajanjem znanja o primeni JDBC pristupa za upravljanje bazom podataka, moguće je preći na napredniji nivo. Ako se pretpostavi postojanje veoma složenog modela domena, manuelno pisanje koda za svaki entitet postaje veoma naporno i neophodno je tražiti alternativni pristup, kao što je Hibernate. Upravo, Spring poseduje instrumente za Hibernate primenu u radu sa bazama podataka.

Za demonstraciju naprednih ORM tehnika i alata za upravljanje bazama podataka, biće uveden novi primer -sistem za upravljanje kursovima centra za obuku. Prva klasa koja će biti kreirana naziva se Course i ona predstavlja entitetsku klasu ili perzistentnu klasu (entity class ili persistent class) zbog toga što reprezentuje realan entitet i njegove instance sačuvane u bazi podataka.

Da bi svaka entitetska klasa bila čuvana ORM okvirom, neophodan je podrazumevani konstruktor bez argumenata.

```
package com.metropolitan.course;

import java.util.Date;

/**
 *
 * @author Vladimir Milicevic
 */
public class Course {

    private Long id;
    private String title;
    private Date beginDate;
    private Date endDate;
    private int fee;

    // kreirati konstruktore, settere i gettere
```

```
}
```

Za svaku entitetsku klasu, neophodno je definisati odgovarajući identifikator kojim je jedinstveno određen entitet. ORM okvir će iskoristiti ovaj identifikator da odredi stanje entiteta. Ukoliko je njegovo stanje **null**, entitet će biti tretiran kao novi i nesačuvan. Za čuvanje ovog entiteta biće iskorišćena operacija **insert**, u suprotnom, ako već postoji, može da se koristi operacija **update**.

Za aktuelni primer, neophodan je *DAO interfejs* (*DAO - Data Access Object*) za enkapsulaciju logike pristupa podacima. Neka interfejs uvodi operacije prikazane sledećim listingom.

```
package com.metropolitan.course;

import java.util.List;

/**
 *
 * @author Vladimir Milicevic
 */
public interface CourseDao {

    public void store(Course course);

    public void delete(Long courseId);

    public Course findById(Long courseId);

    public List<Course> findAll();
}
```

HIBERNATE I JPA

Danas se Hibernate radni okvir smatra implementacijom JPA

U velikom broju slučajeva, kada se koristi **ORM** za čuvanje objekata, operacije **insert** i **update** su kombinovane u jednu operaciju (na primer, **store()** iz prethodnog listinga). Na ovaj način, obezbeđuje se da sam **ORM okvir** odluči da li će objekat biti dodat u bazu ili ažuriran, ukoliko već postoji u bazi podataka.

Sa namerom da **ORM okvir** čuva podatke u bazi podataka, neophodno je da bude upoznat sa mapiranjem meta - podataka za entitetske klase, a za to je neophodno obezbediti i odgovarajući format. Prirodni format za **Hibernate** je XML. Međutim, budući da svaki ORM okvir može da ima vlastiti format za mapiranje meta - podataka, **JPA** definiše skup anotacija za mapiranje meta - podataka u standardnom formatu koji je pogodniji za višestruku upotrebu u različitim ORM okvirima.

Poseban kvalitet za Hibernate je što podržava navedene JPA anotacije. U ovom kontekstu postoje tri različite strategije mapiranja i čuvanja objekata primenom Hibernate i JPA:

1. Primena Hibernate API za čuvanje objekata pomoću Hibernate XML mapiranja;
2. Primena Hibernate API za čuvanje objekata pomoću JPA anotacija;
3. Primena JPA za čuvanje objekata pomoću za čuvanje objekata pomoću JPA anotacija.

Ključni elementi Hibernate i JPA liče na one iz JDBC (sledeća tabela).

Concept	JDBC	Hibernate	JPA
Resource	Connection	Session	EntityManager
Resource factory	DataSource	SessionFactory	EntityManagerFactory
Exception	SQLException	HibernateException	PersistenceException

Slika 1.1 Osnovni koncepti za različite strategije pristupa podacima [izvor: spring.io]

Za Hibernate, osnovni interfejs za čuvanje objekata je **Session**, čija se instanca dobija iz **SessionFactory** instance. Za JPA, odgovarajući interfejs za čuvanje objekata je **EntityManager**, čija se instanca dobija iz **EntityManagerFactory** instance. Izuzetak koji izbacuje Hibernate je tipa **HibernateException**, dok JPA izbacuje izuzetak tipa **PersistenceException** ili neki drugi tip JAVA izuzetaka, poput **IllegalArgumentException** i **IllegalStateException**.

Svi navedeni izuzeci su potklase klase `RuntimeException` i ne forsiraju hvatanje i upravljanje.

Danas se Hibernate radni okvir smatra implementacijom JPA za objektno - relaciono mapiranje.

ČUVANJE OBJEKATA POMOĆU HIBERNATE API I XML MAPIRANJA

Neophodno je obezbediti pojedinačni fajl mapiranja za svaku entitetsku klasu ili veliki fajl mapiranja za nekoliko klasa.

Za mapiranje entitetskih klasa pomoću **Hibernate XML** mapiranja, neophodno je obezbediti pojedinačni fajl mapiranja za svaku entitetsku klasu ili veliki fajl mapiranja za nekoliko klasa. To praktično znači da je neophodno definisati fajl za svaku klasu sa ekstenzijom **.hbm.xml** i nazivom koji odgovara nazivu klase. Na ovaj način se pojednostavljuje održavanje. Deo ekstenzije obeležen sa **.hbm** ukazuje da se radi o *"Hibernate metadata."*

Fajl za mapiranje klase **Course** ima naziv **Course.hbm.xml** i nalazi se u istom paketu kao entitetska klasa.

```
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.metropolitan.course">
    <class name="Course" table="COURSE">
        <id name="id" type="long" column="ID">
            <generator class="identity"/>
        </id>
        <property name="title" type="string">
            <column name="TITLE" length="100" not-null="true"/>
        </property>
        <property name="beginDate" type="date" column="BEGIN_DATE"/>
        <property name="endDate" type="date" column="END_DATE"/>
        <property name="fee" type="int" column="FEE"/>
    </class>
</hibernate-mapping>
```

Svaka aplikacija koja koristi Hibernate zahteva globalni konfiguracioni fajl za podešavanje osobina poput podešavanja baze podataka (na primer, JDBC osobine konekcije), dijalekta baze podataka, lokacije za mapiranje meta - podataka i tako dalje. Hibernate će čitati ove podatke iz datoteke sa nazivom *hibernate.cfg.xml* koja se čuva u korenu za *classpath*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mysql/
course</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">vlada</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="com/metropolitan/course/Course.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

ALTERNATIVA ZA *.CFG.XML FAJL

Mapiranje može biti podešeno i u Java klasi

Kao što je istaknuto u prethodnom izlaganju, svaka aplikacija koja koristi *Hibernate* zahteva globalni konfiguracioni fajl za podešavanje osobina poput podešavanja baze podataka (na primer, JDBC osobine konekcije), dijalekta baze podataka, lokacije za mapiranje meta -

podataka i tako dalje. *Hibernate* će čitati ove podatke iz datoteke sa nazivom *hibernate.cfg.xml* koja se čuva u korenu za *classpath*.

Čitanje ovih podataka može biti i sa nekog drugog mesta. Na primer, data je implementaciona klasa interfejsa *CourseDao* u čijem konstruktoru su kodirana podešavanja koja su prethodno pokazana kroz demonstraciju listinga datoteke *hibernate.cfg.xml*.

```
public class HibernateCourseDao implements CourseDao {

    private final SessionFactory sessionFactory;
    private static ServiceRegistry serviceRegistry;

    public HibernateCourseDao() {

        Configuration configuration = new Configuration()
            .setProperty(AvailableSettings.URL, "jdbc:mysql://localhost:3306/course?zeroDateTimeBehavior=convertToNull")
            .setProperty(AvailableSettings.USER, "root")
            .setProperty(AvailableSettings.PASS, "")
            .setProperty(AvailableSettings.DIALECT,
MySQL5Dialect.class.getName())
            .setProperty(AvailableSettings.SHOW_SQL, String.valueOf(true))
            .setProperty(AvailableSettings.HBM2DDL_AUTO, "update")
            .addClass(Course.class);
        sessionFactory = configuration.buildSessionFactory(serviceRegistry);
    }
    ****

}
```

HIBERNATE API I XML MAPIRANJE - DODATNO IZLAGANJE

*Ukoliko se osobina **hbm2ddl.auto** podesi na **update**, Hibernate značajno ubrzava proces razvoja.*

Pre nego što je moguće čuvanje objekata, neophodno je kreirati tabele u šemi baze podataka u kojima će biti čuvani podaci objekata. Kada se koristi *Hibernate*, nije potrebno da se lično formiraju tabele. Ukoliko se osobina **hbm2ddl.auto** podesi na **update** (videti prethodni kod), Hibernate omogućava ažuriranje šeme baze podataka i kreiranje novih tabela, ukoliko je to potrebno. Primenom ovog mehanizma, značajno se ubrzava proces softverskog razvoja.

Sada je neophodno implementirati DAO interfejs u potpaketu *hibernate*.

```
package com.metropolitan.course.hibernate;

import com.metropolitan.course.Course;
import com.metropolitan.course.CourseDao;
import java.util.List;
import org.hibernate.Query;
```



```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
/**
 *
 * @author Vladimir Milicevic
 */
public class HibernateCourseDao implements CourseDao {

    private SessionFactory sessionFactory;

    // u slučaju da se koristi hibernate.cfg.xml
    //u suprotnom koristiti prethodno pokazani konstruktor

    public HibernateCourseDao() {
        Configuration configuration = new Configuration().configure();
        sessionFactory = configuration.buildSessionFactory();
    }

    public void store(Course course) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            session.saveOrUpdate(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public void delete(Long courseId) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            Course course = (Course) session.get(Course.class, courseId);
            session.delete(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public Course findById(Long courseId) {
        Session session = sessionFactory.openSession();
```

```

        try {
            return (Course) session.get(Course.class, courseId);
        } finally {
            session.close();
        }
    }

    public List<Course> findAll() {
        Session session = sessionFactory.openSession();
        try {
            Query query = session.createQuery("from Course");
            return query.list();
        } finally {
            session.close();
        }
    }
}

```

Prvi korak u primeni Hibernate - a je kreiranje objekta **Configuration** čiji je zadatak da učitava Hibernate podešavanja. Nakon toga, kreira se **Hibernate sesija** iz objekta **Configuration**.

U pokazanim DAO metodama, prvo se prvo se otvara sesija iz produkcije sesije. Za svaku operaciju, koja podrazumeva ažuriranje baze podataka, poput **save()**, **update()** i **delete()**, neophodno je pokrenuti **Hibernate transakciju** za datu sesiju. Ako je operacija uspešno obavljena, vrši se potvrđivanje transakcije operacijom **commit()**. U suprotnom, izbacuje se izuzetak i transakcija se vraća operacijom **rollback()**. Za operacije koje vrše samo čitanje, poput **get** ili HQL (**Hibernate Query Language**) upita, nije potrebno pokretanje transakcija. Konačno, neophodno je zatvoriti sesiju i osloboditi resurse koje je sesija držala.

Sledećom klasom **Main**, moguće je testirati kreirane **DAO** metode.

```

package com.metropolitan.course;

import com.metropolitan.course.hibernate.HibernateCourseDao;
import java.util.GregorianCalendar;
import java.util.List;

/**
 *
 * @author Vladimir Milicevic
 */
public class Main {

    public static void main(String[] args) {
        CourseDao courseDao = new HibernateCourseDao();
        Course course = new Course();
        course.setTitle("Core Spring");
        course.setBeginDate(new GregorianCalendar(2017, 8, 1).getTime());
        course.setEndDate(new GregorianCalendar(2017, 9, 1).getTime());
        course.setFee(1000);
        courseDao.store(course);
        List<Course> courses = courseDao.findAll();
    }
}

```

```
Long courseId = courses.get(0).getId();
course = courseDao.findById(courseId);
System.out.println("Course Title: " + course.getTitle());
System.out.println("Begin Date: " + course.getBeginDate());
System.out.println("End Date: " + course.getEndDate());
System.out.println("Fee: " + course.getFee());
courseDao.delete(courseId);
    }
}
```

▼ Poglavlje 2

JPA i Hibernate

ČUVANJE OBJEKATA POMOĆU HIBERNATE API I JPA ANOTACIJA

JPA anotacije su standardizovane i podržane kompatibilnim ORM okvirima, uključujući Hibernate.

JPA anotacije su standardizovane specifikacijom *JSR-220* pa su podržane svim kompatibilnim ORM okvirima, uključujući Hibernate. Primena navedenih anotacija je pogodnija za uređivanje mapiranja meta - podataka u istom izvornom fajlu.

Sledeći kod klase `Course`, pokazuje primenu JPA anotacija za definisanje mapiranja meta - podataka:

```
package com.metropolitan.course;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 *
 * @author Vladimir Milicevic
 */
@Entity
@Table(name = "COURSE")
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;
    @Column(name = "TITLE", length = 100, nullable = false)
    private String title;
    @Column(name = "BEGIN_DATE")
    private Date beginDate;
    @Column(name = "END_DATE")
    private Date endDate;
    @Column(name = "FEE")
```

```
private int fee;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public Date getBeginDate() {
    return beginDate;
}

public void setBeginDate(Date beginDate) {
    this.beginDate = beginDate;
}

public Date getEndDate() {
    return endDate;
}

public void setEndDate(Date endDate) {
    this.endDate = endDate;
}

public int getFee() {
    return fee;
}

public void setFee(int fee) {
    this.fee = fee;
}

public Course(Long id, String title, Date beginDate, Date endDate, int fee) {
    this.id = id;
    this.title = title;
    this.beginDate = beginDate;
    this.endDate = endDate;
    this.fee = fee;
}

public Course() {
}
```

```
}
```

Entitetska klasa mora da bude obeležena anotacijom **@Entity**. Za svaku osobinu, specificira se naziv kolone i detalji kolone primenom anotacije **@Column**. Identifikator klase mora da bude obeležen anotacijom **@Id**. Moguće je izabrati strategiju generisanja identifikatora primenom anotacije **@GeneratedValue**. Ona govori da je identifikator generisan kolonom tabele koja se odnosi na identitet.

Kada se koriste JPA anotacije, neophodno se specificirati pune nazive entitetskih klasa u datoteci *hibernate.cfg.xml* za čitanje anotacija pomoću Hibernate - a.

```
<hibernate-configuration>
<session-factory>
...
<!-- Za Hibernate XML mapiranja -->
<!--
<mapping resource="com/metropolitan/course/Course.hbm.xml" />
-->
<!-- Za JPA anotacije -->
<mapping class="com.metropolitan.course.Course" />
</session-factory>
</hibernate-configuration>
```

METODA ADDANNOTATEDCLASS()

*Za JPA anotacije za definisanje mapiranja moguće je koristiti metodu **addAnnotatedClass()***

U Hibernate DAO implementaciji, klasa **Configuration** je upotrebljena za čitanje XML mapiranja. Ako se koriste JPA anotacije za definisanje mapiranja meta - podataka za Hibernate, neophodno je koristiti metodu **addAnnotatedClass()** umesto prethodno uvedene metode **addClass()**. Na ovaj način čitaju se odavde meta - podaci umesto pokušaja da se lociraju i pročitaju iz *.hbm.xml datoteke.

U nastavku, moguće je kreirati kod testirati identičnom Main klasom kao u prethodnom slučaju.

```
public HibernateCourseDao() {

    Configuration configuration = new Configuration()
        .setProperty(AvailableSettings.URL, "jdbc:mysql://localhost:3306/
course?zeroDateTimeBehavior=convertToNull")
        .setProperty(AvailableSettings.USER, "root")
        .setProperty(AvailableSettings.PASS, "")
        .setProperty(AvailableSettings.DIALECT,
MySQL5Dialect.class.getName())
        .setProperty(AvailableSettings.SHOW_SQL, String.valueOf(true))
        .setProperty(AvailableSettings.HBM2DDL_AUTO, "update")
```

```
.addAnnotatedClass(Course.class);
sessionFactory = configuration.buildSessionFactory(serviceRegistry);
}
```

ČUVANJE OBJEKATA POMOĆU JPA

Hibernate može da se koristi kao mehanizam za JPA primenu.

JPA nije implementacija perzistencije iako obezbeđuje skup interfejsa za perzistenciju objekata. Iz navedenog razloga, JPA zahteva poseban mehanizam za implementaciju servisa perzistencije. Jedan od kompatibilnih mehanizama je i *Hibernate* koji koristi dodatni modul *Hibernate* **EntityManager**. Navedeno je potrebno sagledati i kroz prizmu platforme. Za *JAVA EE* aplikacije, JPA mehanizam se specifikira u *JAVA EE* kontejneru. Za *JAVA SE* aplikacije, mehanizam se specifikira lokalno. Konfiguracija JPA dešava se u glavnom XML fajlu *persistence.xml*, koji se čuva na lokaciji *META-INF*. Svaki od JPA konfiguracionih fajlova sastoji se od jednog ili više *<persistence-unit>* elemenata kojima su definisane perzistentne klase i načini kako se čuvaju. Svaki *<persistence-unit>* zahteva naziv za identifikaciju, na primer moguće je pridružiti naziv *course*.

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="course">
<properties>
<property name="hibernate.ejb.cfgfile" value="/hibernate.cfg.xml" />
</properties>
</persistence-unit>
</persistence>
```

U ovom JPA konfiguracionom fajlu, moguće je specifikirati *Hibernate* kao mehanizam za JPA podršku, referenciranjem na *Hibernate* konfiguracioni fajl lociran u korenu hijerarhije projekta.

Hibernate **EntityManager** će automatski locirati fajlove za XML mapiranje i JPA anotacije i nije potrebno njihovo eksplicitno specifikiranje. U suprotnom, doći će do izuzetka **hibernate.DuplicateMappingException**.

```
<hibernate-configuration>
<session-factory>
...
<!-- Nije potrebno specifikiranje datoteka mapiranja i klasa -->
<!--
<mapping resource="com/metropolitan/course/Course.hbm.xml" />
<mapping class="com.metropolitan.course.Course" />
-->
</session-factory>
</hibernate-configuration>
```

Alternativno referenciranju na Hibernate konfiguracioni fajl, moguće je centralizovati sve Hibernate konfiguracije u fajl [persistence.xml](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/
persistence_2_1.xsd">
    <persistence-unit name="course"/>
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class> com.metropolitan.course.Course</class>
    <properties>
        <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
        <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/course"/>
        <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
        <property name="hibernate.connection.username" value="root"/>
        <property name="hibernate.connection.password" value="vlada"/>
        <property name="hibernate.show_sql" value="true"/>
        <property name="hibernate.format_sql" value="true"/>
    </properties>

    </persistence-unit>

</persistence>
```

DODATNA RAZMATRANJA

Java EE kontejner upravlja menadžerom entiteta i direktno ga ubacuje u EJB.

U JAVA EE okruženju, Java EE kontejner upravlja menadžerom entiteta i direktno ga ubacuje u [EJB \(Enterprise JavaBeans\)](#). Međutim, kada se JPA koristi izvan Java EE kontejnera, na primer u JAVA SE aplikacijama, neophodno je lično kreirati i održavati menadžer entiteta.

Za korišćenje EntityManager biblioteka, neophodno je dodati sledeće zavisnosti u Maven projekat.

```
dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.1.Final</version>
</dependency>
```


Sada je moguće implementirati *CourseDao* interfejs u potpaketu *jpa*, primenom JPA u JAVA SE ili EE aplikaciji. Pre nego što se pozove JPA za perzistenciju objekata, neophodno je inicijalizovati *EntityManagerFactory* za produkciju menadžera entiteta.

```
package com.metropolitan.course.jpa;

import com.metropolitan.course.Course;
import com.metropolitan.course.CourseDao;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;

/**
 *
 * @author Vladimir Milicevic
 */
public class JpaCourseDao implements CourseDao {

    private EntityManagerFactory entityManagerFactory;

    public JpaCourseDao() {
        entityManagerFactory = Persistence.createEntityManagerFactory("course");
    }

    public void store(Course course) {
        EntityManager manager = entityManagerFactory.createEntityManager();
        EntityTransaction tx = manager.getTransaction();
        try {
            tx.begin();
            manager.merge(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            manager.close();
        }
    }

    public void delete(Long courseId) {
        EntityManager manager = entityManagerFactory.createEntityManager();
        EntityTransaction tx = manager.getTransaction();
        try {
            tx.begin();
            Course course = manager.find(Course.class, courseId);
            manager.remove(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        }
    }
}
```

```

        } finally {
            manager.close();
        }
    }

    public Course findById(Long courseId) {
        EntityManager manager = entityManagerFactory.createEntityManager();
        try {
            return manager.find(Course.class, courseId);
        } finally {
            manager.close();
        }
    }

    @Override
    public List<Course> findAll() {
        EntityManager manager = entityManagerFactory.createEntityManager();
        try {
            Query query = manager.createQuery(
                "select course from Course course");
            return query.getResultList();
        } finally {
            manager.close();
        }
    }
}

```

Produkcija menadžera entiteta obavlja se statičkom metodom **`createEntityManagerFactory()`** klase **`javax.persistence.Persistence`**. Ovde je neophodno proslediti naziv iz elementa `<persistence-unit>` koji se čuva u datoteci `persistence.xml`.

U prethodnim DAO metodama, prvo je kreiran menadžer entiteta iz produkcije menadžera entiteta. Zatim, za svaku operaciju koja podrazumeva ažuriranje baze podataka, kao što su `merge()` i `remove()`, neophodno je pokrenuti JPA transakciju za menadžera entiteta. Metode koje vrše samo čitanje, na primer `find()`, ne zahtevaju pokretanje transakcije. Na kraju, neophodno je zatvoriti menadžera entiteta i osloboditi resurse koje je on zauzeo.

Sledećom `Main` klasom je moguće testirati kreirani DAO, ali ovaj put je neophodno kreirati JPA DAO objekat.

```

package com.metropolitan.course;

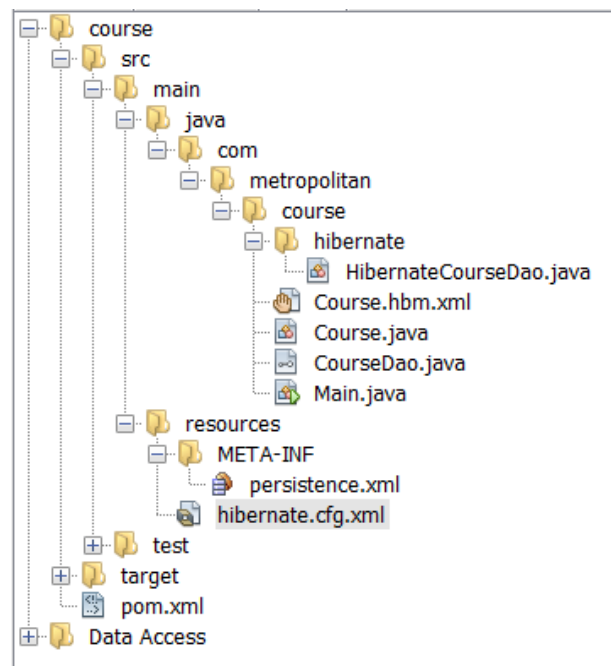
...
public class Main {
    public static void main(String[] args) {
        CourseDao courseDao = new JpaCourseDao();
        ...
    }
}

```

DEMONSTRACIJA

Sledi demonstracija kreiranog primera

Kreirani primer ima strukturu prikazanu sledećom slikom.



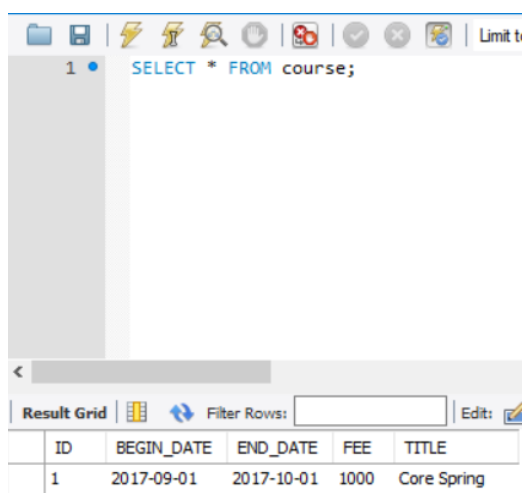
Slika 2.1 Struktura kreiranog projekta [izvor: autor]

Sledeće dve slike pokazuju ispravnost urađenog primera. Prvom slikom se na konzoli vidi rezultat izvršavanja, dok drugom slikom je prikazan *MySql Workbench* sa sačuvanim zapisom u tabeli *course*, istoimene baze podataka.

```
INFO: HHH000000: Hibernate Commons Annotations (5.0.4.Final)
jul 21, 2017 6:12:26 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH000040: Using Hibernate built-in connection pool (not for production use!)
jul 21, 2017 6:12:26 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH000040: using driver [com.mysql.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/course]
jul 21, 2017 6:12:26 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH000046: Connection properties: {user=root, password=****}
jul 21, 2017 6:12:26 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH000006: Autocommit mode: false
jul 21, 2017 6:12:26 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
[ Fri Jul 21 18:12:27 CEST 2017 WARN: Establishing SSL connection without server's identity verification is not recommended.
be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verif
able SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
jul 21, 2017 6:12:27 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
jul 21, 2017 6:12:27 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
INFO: HHH000397: Using ASTQueryTranslatorFactory

Course Title: Core Spring
Begin Date: 2017-09-01 00:00:00.0
End Date: 2017-10-01 00:00:00.0
Fee: 1000
```

Slika 2.2 Izvršavanje programa [izvor: autor]



The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons. Below the toolbar, a SQL query is entered in a text area: `SELECT * FROM course;`. Below the query area, there is a section labeled "Result Grid" which contains a table with the following data:

ID	BEGIN_DATE	END_DATE	FEE	TITLE
1	2017-09-01	2017-10-01	1000	Core Spring

Slika 2.3 Zapis u bazi podataka [izvor: autor]

▼ Poglavlje 3

Podešavanje produkcije ORM resursa u Springu

PROBLEM PODEŠAVANJA ORM RESURSA

Spring obezbeđuje nekoliko produkcijskih zrna za kreiranje produkcije Hibernate sesije.

Kada se samostalno koristi *ORM* okvir, neophodno je manuelno podesiti produkciju njegovih resursa njegovim API - jem. Za *Hibernate* i *JPA*, neophodno je kreirati produkcije sesije ili menadžera entiteta, respektivno iz nativnih *Hibernate API* ili *JPA*. Ne postoji druga mogućnost već manuelno podešavanje navedenih objekata bez Spring podrške.

Za rešavanje navedenog problema, Spring obezbeđuje nekoliko produkcionih zrna za kreiranje Hibernate produkcije sesije ili JPA menadžera entiteta. Sva navedena zrna su realizovana kao zrna jedinci (Eng. *singleton*) u *Spring IoC kontejneru*. Navedene produkcije moguće je deliti između većeg broja zrna putem umetanja zavisnosti. Dalje, omogućeno je integrisanje Hibernate produkcija sesija i JPA menadžera entiteta sa ostalim Spring alatima kao što su izvor podataka (eng. data source) ili menadžeri transakcija (eng. transaction managers)

PODEŠAVANJE PRODUKCIJE HIBERNATE SESIJE U SPRINGU

Obezbeđivanje prihvatanja produkcije sesije preko umetanja zavisnosti.

Za *Hibernate*, *Spring* obezbeđuje *LocalSessionFactoryBean* za direktno kreiranje *Hibernate SessionFactory*. U slučaju *JPA*, *Spring* je predvideo nekoliko opcija za konstruisanje *EntityManagerFactory*. U konkretnom razmatranju, akcenat će biti na mehanizmima vraćanja *EntityManagerFactory* iz *JNDI (Java Naming and Directory Interface)* i korišćenja zrna tipa *LocalEntityManagerFactoryBean* i *LocalContainerEntityManagerFactoryBean* kao i isticanju razlika između svake od navedenih opcija.

Za početak, neophodno je modifikovati klasu *HibernateCourseDao* tako da prihvata produkciju sesije preko umetanja zavisnosti, umesto direktnog kreiranja iz konstruktora putem nativnog *Hibernate API*.

```
/**
 *
 * @author Vladimir Milicevic
 */
public class HibernateCourseDao implements CourseDao {

    private final SessionFactory sessionFactory;

    public HibernateCourseDao(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;

        ***
    }
}
```

U nastavku, neophodno je kreirati konfiguracionu klasu za primenu *Hibernate ORM* okvira. Takođe, moguće je kreirati *HibernateCourseDao* instancu kojom će rukovati *Spring*. Sledi listing konfiguracione klase.

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

    @Bean
    public CourseDao courseDao(SessionFactory sessionFactory) {
        return new HibernateCourseDao(sessionFactory);
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
        sessionFactoryBean.setHibernateProperties(hibernateProperties());
        sessionFactoryBean.setAnnotatedClasses(Course.class);
        return sessionFactoryBean;
    }

    private Properties hibernateProperties() {
        Properties properties = new Properties();
        properties.setProperty(AvailableSettings.URL,
            "jdbc:mysql://localhost:3306/course");
        properties.setProperty(AvailableSettings.USER, "root");
        properties.setProperty(AvailableSettings.PASS, "");
        properties.setProperty(AvailableSettings.DIALECT,
            MySQLDialect.class.getName());
        properties.setProperty(AvailableSettings.SHOW_SQL, String.valueOf(true));
        properties.setProperty(AvailableSettings.HBM2DDL_AUTO, "update");
        return properties;
    }
}
```

Ako se pogleda pažljivo listing, moguće je uočiti da su sve osobine koje su ranije podešene objektom *Configuration* sada prosledene *Properties* objektu i dodate u zрно tipa *LocalSessionFactoryBean*. Entitetska klasa (obeležena anotacijom *@Entity*) je prosledena metodom *setAnnotatedClasses()*. Konstruisani objekat *SessionFactory* je prosleđen *HibernateCourseDao* putem konstruktora.

PODEŠAVANJE PRODUKCIJE HIBERNATE SESIJE U SPRINGU - DRUGI NAČIN

Učitavanje resursa iz putanje klase.

Ukoliko se radi na projektu koji i dalje koristi Hibernate XML fajlove za mapiranje, moguće je koristiti osobinu *mappingLocations* za specifikaciju mapirajućih datoteka. Zrno tipa *LocalSessionFactoryBean* takođe omogućava primenu *Spring* podrške za učitavanje resursa iz mapirajućih datoteka sa različitih tipova lokacija. Moguće je specificirati putanje do resursa (mapirajućih datoteka) primenom osobine *mappingLocations* čiji je tip *Resource[]*.

```
@Bean
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
    sessionFactoryBean.setDataSource(dataSource());
    sessionFactoryBean.setMappingLocations(
        new ClassPathResource("com/metropolitan/course/Course.hbm.xml"));
    sessionFactoryBean.setHibernateProperties(hibernateProperties());
    return sessionFactoryBean;
}
```

DŽOKER ZA RESURSE

Moguće je koristiti džoker u putanji resursa za podudaranje sa većim brojem mapirajućih datoteka.

Kada se koristi *Spring* podrška za učitavanje resursa, moguće je koristiti *džoker* (eng. *wild card*) u putanji resursa za podudaranje sa većim brojem mapirajućih datoteka. **Zbog navedenog nije potrebno podešavati njihove lokacije svaki put kada se kreira nova entitetska klasa.** Za realizovanje navedenog zadatka neophodna je primena zrna *ResourcePatternResolver* unutar konfiguracione klase. Ključni korak predstavlja primena klase *ResourcePatternUtils* i implementacija *ResourceLoaderAware* interfejsa. Poziv metode *getResourcePatternResolver()* vraća *ResourcePatternResolver* baziran na *ResourceLoader* (videti sledeći listing).

```
/**
 *
 * @author Vlada
 */
@Configuration
```

```
public class CourseConfiguration implements ResourceLoaderAware{

    private ResourcePatternResolver resourcePatternResolver;

    @Override
    public void setResourceLoader(ResourceLoader resourceLoader) {
        this.resourcePatternResolver
            = ResourcePatternUtils.getResourcePatternResolver(resourceLoader);
    }
    ****
}
```

Sada je moguće koristiti zрно *ResourcePatternResolver* za rešavanje šablona resursa. U konfiguracionu klasu je neophodno dodati sledeće modifikacije:

```
@Bean
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
    Resource[] mappingResources
        = resourcePatternResolver.getResources("classpath:com/metropolitan/
course/*.hbm.xml");
    sessionFactoryBean.setMappingLocations(mappingResources);
    ***
    return sessionFactoryBean;
}
```

Sada je potrebno modifikovati Main klasu za prihvatanje *HibernateCourseDao* instance iz Spring IoC kontejnera.

```
public class Main {
    public static void main(String[] args) {
        ApplicationContext context =
            new AnnotationConfigApplicationContext(CourseConfiguration.class);
        CourseDao courseDao = context.getBean(CourseDao.class);

        ...
    }
}
```

KREIRANJE I UPOTREBA IZVORA PODATAKA

Demonstracija primene izvora podataka iz Spring IoC kontejnera.

U prethodnom izlaganju je produkcionim zrnom kreirana produkcija sesije učitavanjem *Hibernate* konfiguracione datoteke, koja je uključivala i podešavanja baze podataka.

Sada je moguće razmatrati drugačiji scenario - postoji izvor podataka (eng. *data source*) definisan u *Spring IoC kontejneru*. Ukoliko se koristi ovakav izvor podataka, neophodno ga je umetnuti u osobinu *dataSource* zrna *LocalSessionFactoryBean*. Izvor podataka definisan u ovoj osobini će redefinisati podešavanja iz *Hibernate* konfiguracija. U ovom slučaju, *Hibernate*

podešavanja ne bi trebalo da definišu provajdera konekcije zbog izbegavanje besmislenog dupliranja podešavanja.

```
@Configuration
public class CourseConfiguration{

    ****

    @Bean
    public DataSource dataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setUsername("user");
        dataSource.setPassword("");
        dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/course");
        dataSource.setMinimumIdle(2);
        dataSource.setMaximumPoolSize(5);
        return dataSource;
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {
        LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
        sessionFactoryBean.setDataSource(dataSource);
        sessionFactoryBean.setHibernateProperties(hibernateProperties());
        sessionFactoryBean.setAnnotatedClasses(Course.class);
        return sessionFactoryBean;
    }

    private Properties hibernateProperties() {
        Properties properties = new Properties();
        properties.setProperty(AvailableSettings.DIALECT,
            MySQLDialect.class.getName());
        properties.setProperty(AvailableSettings.SHOW_SQL, String.valueOf(true));
        properties.setProperty(AvailableSettings.HBM2DDL_AUTO, "update");
        return properties;
    }

}
```

Dalje je moguće u potpunosti ignorisati *Hibernate* konfiguracionu datoteku spajanjem svih podešavanja unutar zrna *LocalSessionFactoryBean*. Na primer, moguće je specificirati paket sa obeleženim klasama u okviru osobine *packageToScan*. Ostale osobine, kao što je dijalekt baze podataka, moguće je podesiti osobinom *hibernateProperties*.

```
@Configuration
public class CourseConfiguration {
    ...
    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
        sessionFactoryBean.setDataSource(dataSource());
        sessionFactoryBean.setPackagesToScan("com.metropolitan.course");
        sessionFactoryBean.setHibernateProperties(hibernateProperties());
    }
}
```

```

    return sessionFactoryBean;
}

private Properties hibernateProperties() {
    Properties properties = new Properties();
    properties.put("hibernate.dialect",
        org.hibernate.dialect.MySQLDialect.class.getName());
    properties.put("hibernate.show_sql", true);
    properties.put("hibernate.hbm2ddl.auto", "update");
    return properties;
}
}

```

Sada je moguće iz projekta izbrisati datoteku sa Hibernate konfiguracijama (npr. **hibernate.cfg.xml**) jer su sva podešavanja predata Springu.

PODEŠAVANJE PRODUKCIJE JPA MENADŽERA ENTITETA U SPRINGU

Takođe, neophodno je modifikovati konkretan DAO primenom umetanja zavisnosti..

U ovom svetlu, prvo će biti modifikovan *JpaCourseDao* za prihvatanje produkcije JPA menadžera entiteta putem umetanja zavisnosti, umesto direktnog kreiranja konstruktorom.

```

package com.metropolitan.course;
...
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class JpaCourseDao implements CourseDao {
    private final EntityManagerFactory entityManagerFactory;

    public JpaCourseDao (EntityManagerFactory entityManagerFactory) {
        this.entityManagerFactory = entityManagerFactory;
    }
    ...
}

```

JPA specifikacija definiše kako je moguće obezbediti produkciju *JPA menadžera entiteta* u okruženjima poput *Java SE* i *Java EE*. U okruženju Java EE, produkcija menadžera entiteta se obavlja ručno pozivom statičke metode *createEntityManagerFactory()* klase *Persistence*.

Da bi bilo moguće obaviti postavljene zadatke, neophodno je kreirati zrno, u okviru konfiguracione klase, za upotrebu *JPA*. Spring obezbeđuje produkciono zrno *LocalEntityManagerFactoryBean* za kreiranje produkcije menadžera entiteta u *Spring IoC*

kontejneru. Takođe, ovde je neophodno postojanje naziva jedinice perzistencije definisanog u JPA konfiguracionoj datoteci. Takođe, moguće je deklarirati instancu *JpaCourseDao* kojom upravlja Spring.

```
package com.metropolitan.config;

import com.metropolitan.course.CourseDao;
import com.metropolitan.course.jpa.JpaCourseDao;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.jpa.LocalEntityManagerFactoryBean;
import javax.persistence.EntityManagerFactory;

@Configuration
public class CourseConfiguration {
    @Bean
    public CourseDao courseDao(EntityManagerFactory entityManagerFactory) {
        return new JpaCourseDao(entityManagerFactory);
    }

    @Bean
    public LocalEntityManagerFactoryBean entityManagerFactory() {
        LocalEntityManagerFactoryBean emf = new LocalEntityManagerFactoryBean();
        emf.setPersistenceUnitName("course");
        return emf;
    }
}
```

PRIMENA PRODUKCIJE JPA MENADŽERA ENTITETA

Pokretanjem Main klase moguće je proveriti obavljene zadatke.

Sada, moguće je testirati instancu *JpaCourseDao*, sledećom *Main* klasom, njenim preuzimanjem iz *Spring IoC kontejnera*.

```
package com.metropolitan.course;

...
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
    public static void main(String[] args) {
        ApplicationContext context =
            new AnnotationConfigApplicationContext(CourseConfiguration.class);
        CourseDao courseDao = context.getBean(CourseDao.class);
        ...
    }
}
```

PRODUKCIJA JPA MENADŽERA ENTITETA - JAVA EE

Spring omogućava fleksibilniji način kreiranja produkcije JPA menadžera entiteta, primenom alternativnog produkcionog zrna.

Ako se koristi Java EE platforma, produkciju menadžera entiteta moguće je obaviti iz Java EE kontejnera koristeći JNDI. U Spring okviru, za JNDI pretragu se koristi objekat *JndiLocatorDelegate*. Moguće je koristiti i kreiranje zrna tipa *JndiObjectFactoryBean*, a li je prvi način značajno efikasniji i jednostavniji.

```
@Bean
public EntityManagerFactory entityManagerFactory() throws NamingException {
    return JndiLocatorDelegate.createDefaultResourceRefLocator()
        .lookup("jpa/coursePU", EntityManagerFactory.class);
}
```

LocalEntityManagerFactoryBean kreira produkciju menadžera entiteta učitavanjem datoteke JPA konfiguracija (npr. *persistence.xml*). Spring podržava jednostavniji i fleksibilniji način za produkovanje menadžera entiteta primenom nekog drugog produkcionog zrna - *LocalContainerEntityManagerFactoryBean*. Na ovaj način su redefinisane pojedine konfiguracije iz JPA konfiguracione datoteke, kao što su izvor podataka i dijalekt baze podataka. Na ovaj način programeri imaju dodatne pogodnosti i olakšanja za podešavanje produkcije menadžera entiteta.

```
@Configuration
public class CourseConfiguration {
    ...
    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource
dataSource) {
        LocalContainerEntityManagerFactoryBean emf =
            new LocalContainerEntityManagerFactoryBean();
        emf.setPersistenceUnitName("course");
        emf.setDataSource(dataSource);
        emf.setJpaVendorAdapter(jpaVendorAdapter());
        return emf;
    }

    private JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter jpaVendorAdapter = new HibernateJpaVendorAdapter();
        jpaVendorAdapter.setShowSql(true);
        jpaVendorAdapter.setGenerateDdl(true);
        jpaVendorAdapter.setDatabasePlatform(MySqlDialect.class.getName());
        return jpaVendorAdapter;
    }

    @Bean
    public DataSource dataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setUsername("root");
    }
}
```

```
dataSource.setPassword("");  
dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/course");  
dataSource.setMinimumIdle(2);  
dataSource.setMaximumPoolSize(5);  
return dataSource;  
}  
}
```

HIBERNATEJPAVENDORADAPTER - DOPUNSKO RAZMATRANJE

Hibernate okvir kao implementacija za JPA API koristi `HibernateJpaVendorAdapter`.

Ako se obrati pažnja na prethodnu konfiguracionu klasu moguće je primetiti da je obavljeno umetanje izvora podataka u produkciju menadžera entiteta. Na ovaj način su nadjačana podešavanja baze podataka navedena u JPA konfiguracionoj datoteci.

Moguće je podesiti i specifične osobine za JPA primenom adapterske klase `LocalContainerEntityManagerFactoryBean`. Primenom `Hibernate` okvira kao implementacije za `JPA API`, trebalo bi izabrati `HibernateJpaVendorAdapter`. Ostale osobine koje nisu podržane ovim adapterom mogu biti podešene primenom osobine `jpaProperties`. Sada je moguće pojednostaviti JPA konfiguracionu datoteku (npr. `persistence.xml`) jer je veći deo podešavanja dodeljen Springu. Navedeno je prikazano sledećim listingom:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence  
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"  
version="2.1">  
  
<persistence-unit name="course" transaction-type="RESOURCE_LOCAL">  
    <class>com.metropolitan.course.Course</class>  
</persistence-unit>  
  
</persistence>
```

Spring, takođe, dozvoljava podešavanje JPA `EntityManagerFactory` bez postojanja datoteke `persistence.xml`. Programeri imaju mogućnost da sva ova podešavanja obave u `Spring` konfiguracionoj datoteci. Umesto osobine `persistenceUnitName`, u ovom slučaju, neophodno je podesiti osobinu `packagesToScan`. Kada je ovo urađeno, moguće je u potpunosti obrisati datoteku `persistence.xml` iz Spring projekta.

```
@Bean  
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {  
    LocalContainerEntityManagerFactoryBean emf =  
        new LocalContainerEntityManagerFactoryBean();  
    emf.setDataSource(dataSource());  
}
```

```
emf.setPackagesToScan("com.metropolitan.course");  
emf.setJpaVendorAdapter(jpaVendorAdapter());  
return emf;  
}
```

ZADATAK ZA SAMOSTALNI RAD 1

Pokušajte sami

1. Pokušajte da implementirate kod za podešavanje produkcije ORM resursa u Springu priložen u ovom objektu učenja;
2. Ažurirajte glavnu klasu *Main.java* za demonstraciju urađenih modifikacija nad projektom *course*.
3. Pokušajte da pokrenete vaš prvi ORM primer i pratite rezultate izvršavanja.

VIDEO MATERIJALI

Integracija Spring sa Hibernate i JPA.

SessionFactory creation in Hibernate 5

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

Spring 4 and JPA with Hibernate

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

▼ Poglavlje 4

Hibernate kontekstualne sesije

KREIRANJE DAO KLASA

Kada se koriste kontekstualne sesije kao pristup, DAO metode zahtevaju pristup produkciji sesije.

Problematiku rada sa ORM konceptima moguće je dalje širiti. Neka je sada cilj pisanje DAO klase primenom čistog Hibernate API pri čemu se i dalje oslanja na primenu Springom upravljanih transakcija.

Takođe bi trebalo napomenuti da od verzije Spring 3, produkcijom sesija može da upravlja kontekstualnim sesijama i da ih vraća iz poziva metode `getCurrentSession()` definisane u `org.hibernate.SessionFactory`. U okviru jedne transakcije, biće vraćena jedinstvena sesija za svaki poziv `getCurrentSession()` metode. Na ovaj način je obezbeđeno postojanje samo jedne Hibernate sesije po transakciji, a to funkcioniše veoma dobro sa Spring podrškom za upravljanje transakcijama.

Kada se koriste kontekstualne sesije kao pristup, sve DAO metode zahtevaju pristup produkciji sesije što može biti umetnuto preko setter metode ili argumenta konstruktora. Tada, u svakoj DAO metodi, dobija se kontekstualna sesija i koristi se za čuvanje objekata.

```
/**
 *
 * @author Vladimir Milicevic
 */
public class HibernateCourseDao implements CourseDao {

    private final SessionFactory sessionFactory;

    public HibernateCourseDao(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional
    @Override
    public Course store(Course course) {
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(course);
        return course;
    }

    @Transactional
    @Override
```

```
public void delete(Long courseId) {
    Session session = sessionFactory.getCurrentSession();
    Course course = session.get(Course.class, courseId);
    session.delete(course);
}

@Transactional(readOnly = true)
@Override
public Course findById(Long courseId) {
    Session session = sessionFactory.getCurrentSession();
    return session.get(Course.class, courseId);
}

@Transactional(readOnly = true)
@Override
public List<Course> findAll() {
    Session session = sessionFactory.getCurrentSession();
    return session.createQuery("from Course", Course.class).list();
}
}
```

REGISTROVANJE MENADŽERA TRANSAKCIJA

DAO metode obeležene su anotacijom `@Transactional`.

Pre samog procesa kreiranja novih detalja konfiguracione klase, neophodno je još jednom staviti akcenat na prethodno priloženi kod DAO klase. Moguće je primetiti da su sve DAO metode obeležene anotacijom `@Transactional`. Ovo je obavezno iz razloga integracije Springa i okvira Hibernate kroz podršku Hibernate kontekstualnih sesija. Spring ima vlastitu implementaciju Hibernate interfejsa `CurrentSessionContext`. Spring će pokušati da reši transakciju i neće uspeti uz komentar da nijedna Hibernate sesija nije povezana sa odgovarajućom niti. Da bi navedeno bilo postignuto, neophodno je obeležiti svaku metodu (ili celu dao klasu) anotacijom `@Transactional`. Na ovaj način je obezbeđeno da će sve operacije perzistencije, unutar DAO metoda, biti izvršene unutar iste transakcije, a otuda i iste sesije. Dalje, ukoliko metoda neke komponente iz servisnog sloja poziva više DAO metoda i nameće vlastitu transakciju ovim metodama, one će opet biti izvršene u okviru jedne sesije.

Nakon izvršene dodatne analize, sada je moguće u konfiguracionoj klasi zrna obaviti registrovanje instance menadžera transakcija. Takođe, neophodno je obezbediti i deklarativno upravljanje transakcijama uvođenjem nove anotacije `@EnableTransactionManagement`. Navedeno je moguće ilustrovati sledećim listingom.

```
@Configuration
@EnableTransactionManagement
public class CourseConfiguration {
    @Bean
    public CourseDao courseDao(SessionFactory sessionFactory) {
        return new HibernateCourseDao(sessionFactory);
    }
}
```



```
@Bean
public HibernateTransactionManager transactionManager(SessionFactory
 sessionFactory) {
    return new HibernateTransactionManager(sessionFactory);
}
}
```

HIBERNATE KONTEKSTUALNE SESIJE I IZUZECI

Za prevođenje Hibernate izuzetaka u izuzetke tipa `DataAccessException` koristi se `@Repository`.

Ukoliko se javi izuzetak prilikom poziva native metode unutar Hibernate sesije, izbačeni izuzetak će biti tipa `HibernateException`. Ukoliko programer želi da Hibernate izuzetke prevede u Spring izuzetke tipa `DataAccessException` i na taj način konzistentno upravlja izuzecima, neophodno je da primeni anotaciju `@Repository` na DAO klasu i na taj način zahteva prevođenje izuzetaka. Navedeno je moguće ilustrovati sledećim listingom:

```
....
import org.springframework.stereotype.Repository;

@Repository
public class HibernateCourseDao implements CourseDao {
    ...
}
```

Za prevođenje Hibernate izuzetaka u izuzetke koji pripadaju Spring `DataAccessException` hijerarhiji izuzetaka, zaduženo je zрно `PersistenceExceptionTranslationPostProcessor`. Ovo zрно prevodi samo izuzetke koji potiču iz zrna obeleženih anotacijom `@Repository`. Prilikom upotrebe Java baziranih konfiguracija, ovo zрно je automatski registrovano u `AnnotationConfigApplicationContext` i otuda ne postoji potreba za njegovim eksplicitnim deklarisanjem.

U Spring okviru, `@Repository` je stereotipna anotacija, a to znači da obezbeđuje više funkcionalnosti. Obeležavanjem klase ovom anotacijom, klasa postaje komponenta koja može biti automatski pronađena kroz mehanizam skeniranja komponentata. Takođe, moguće je u okviru ove anotacije pridružiti naziv komponenti i omogućiti automatsko povezivanje produkcije sesije preko Spring IoC kontejnera.

```
@Repository("courseDao")
public class HibernateCourseDao implements CourseDao {
    private final SessionFactory sessionFactory;
    public HibernateCourseDao (SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    ...
}
```

Konačno, neophodno je dodati `@ComponentScan` anotaciju i obrisati originalnu deklaraciju zrna `HibernateCourseDao`.

```
@Configuration
@EnableTransactionManagement
@ComponentScan("com.metropolitan.course")
public class CourseConfiguration { ... }
```

ZADATAK ZA SAMOSTALNI RAD 2

Pokušajte sami

1. Pokušajte da implementirate kod iz OU "Hibernate kontekstualne sesije";
2. Ažurirajte glavnu klasu `Main.java` za demonstraciju urađenih modifikacija nad projektom `course`.
3. Pokušajte da pokrenete vaš prvi ORM primer i pratite rezultate izvršavanja.

▼ Poglavlje 5

JPA umetanje konteksta

ČUVANJE OBJEKATA POMOĆU JPA UMETANJA KONTEKSTA

Alternativa primeni JpaTemplate je korišćenje JPA umetanja konteksta.

U Java EE okruženju, Java EE kontejner upravlja menadžerima entiteta i vrši direktno umetanje ovih menadžera u *EJB (Enterprise Java Bean)*. EJB komponente na jednostavan način izvode operacije perzistencije za umetnuti menadžer entiteta bez obraćanja pažnje na kreiranje menadžera entiteta i upravljanje transakcijama.

Po osnovnim podešavanjima, anotacija *@PersistenceContext* se koristi za umetanje menadžera entiteta u EJB komponente. Spring, takođe, ima mogućnost interpretacije ove anotacije na način umetanja menadžera entiteta u osobinu. Spring obezbeđuje da sve operacije perzistencije, u okviru iste transakcije, budu rukovanje istim menadžerom entiteta.

Za korišćenje pristupa umetanja konteksta, neophodno je deklarirati polje za menadžera entiteta u DAO klasi i obeležiti ga anotacijom *@PersistenceContext*. Spring će obaviti njegovo umetanje u kreirano polje i na taj način obezbediti izvođenje operacija perzistencije za objekte.

```
package com.metropolitan.course.hibernate;

import com.metropolitan.course.Course;
import com.metropolitan.course.CourseDao;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import org.springframework.transaction.annotation.Transactional;

/**
 *
 * @author Vladimir Milicevic
 */
public class JpaCourseDao implements CourseDao {

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    @Override
    public Course store(Course course) {
```

```

        return entityManager.merge(course);
    }

    @Transactional
    @Override
    public void delete(Long courseId) {
        Course course = entityManager.find(Course.class, courseId);
        entityManager.remove(course);
    }

    @Transactional(readOnly = true)
    @Override
    public Course findById(Long courseId) {
        return entityManager.find(Course.class, courseId);
    }

    @Transactional(readOnly = true)
    @Override
    public List<Course> findAll() {
        TypedQuery<Course> query
            = entityManager.createQuery("select c from Course c", Course.class);
        return query.getResultList();
    }
}

```

JPA UMETANJA KONTEKSTA - PODEŠAVANJA

PersistenceAnnotationBeanPostProcessor je moguće automatski registrovati Java konfiguracijama.

Ako se pogleda još jednom priložena DAO klasa moguće je primetiti da su njene metode obeležene anotacijom [@Transactional](#). Na ovaj način je obezbeđeno izvođenje operacija perzistencije jedne metode u okviru iste transakcije, a otuda i istog menadžera entiteta.

U konfiguracionoj datoteci zna, neophodno je deklarirati instancu tipa `JpaTransactionManager` i omogućiti deklarativno upravljanje transakcijama uvođenjem anotacije [@EnableTransactionManagement](#).

Instanca tipa [PersistenceAnnotationBeanPostProcessor](#) je registrovana automatski kada se koriste Java bazirane konfiguracije za umetanje menadžera entiteta u osobine obeležene anotacijom [@PersistenceContext](#).

```

/**
 *
 * @author Vlada
 */
@Configuration
@EnableTransactionManagement
public class CourseConfiguration {

```

```

@Bean
public CourseDao courseDao() {
    return new JpaCourseDao();
}

@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean emf
        = new LocalContainerEntityManagerFactoryBean();
    emf.setDataSource(dataSource());
    emf.setJpaVendorAdapter(jpaVendorAdapter());
    return emf;
}

private JpaVendorAdapter jpaVendorAdapter() {
    HibernateJpaVendorAdapter jpaVendorAdapter = new
HibernateJpaVendorAdapter();
    jpaVendorAdapter.setShowSql(true);
    jpaVendorAdapter.setGenerateDdl(true);
    jpaVendorAdapter.setDatabasePlatform(MySqlDialect.class.getName());
    return jpaVendorAdapter;
}

@Bean
public JpaTransactionManager transactionManager(EntityManagerFactory
entityManagerFactory) {
    return new JpaTransactionManager(entityManagerFactory);
}

@Bean
public DataSource dataSource() {...
}
}

```

PRIMENA ANOTACIJE PERSISTENCEUNIT

Umetanje produkcije menadžera entiteta u osobinu primenom anotacije `PersistenceUnit`

Zrno tipa `PersistenceAnnotationBeanPostProcessor`, takođe, može obaviti umetanje produkcije menadžera entiteta u osobinu primenom anotacije `@PersistenceUnit`. Na ovaj način kreiranje menadžera entiteta i upravljanje transakcijama je prepušteno programeru. Navedeno važi i za umetanje produkcije menadžera entiteta preko setter metode.

```

public class JpaCourseDao implements CourseDao {
    @PersistenceContext
    private EntityManager entityManager;

    @PersistenceUnit
    private EntityManagerFactory entityManagerFactory;
}

```

```
...  
}
```

JPA UMETANJA KONTEKSTA - IZUZECI

DataAccessException su dostupni primenom anotacije @Repository na DAO klasu.

Prilikom pozivanja nativnih metoda za JPA menadžera entiteta, izbačeni izuzetak će biti nativnog tipa *PersistenceException* ili nekog drugog *Java SE* tipa poput *IllegalArgumentException* i *IllegalStateException*. Ukoliko je cilj prevođenje JPA izuzetaka u Spring tip *DataAccessException*, neophodno je primeniti anotaciju *@Repository* na DAO klasu.

```
@Repository("courseDao")  
public class JpaCourseDao implements CourseDao {  
    ...  
}
```

Instanca tipa *PersistenceExceptionTranslationPostProcessor* prevodi JPA izuzetke u izuzetke iz Spring *DataAccessException* hijerarhije. Kada se koriste Java bazirane konfiguracije ovo zрно će automatski biti registrovanu u *AnnotationConfigApplicationContext*. Otuda, nema potrebe za njegovim eksplicitnim deklarisanjem.

ZADATAK ZA SAMOSTALNI RAD 3

Pokušajte sami

1. Pokušajte da implementirate kod iz OU "JPA umetanje konteksta";
2. Ažurirajte glavnu klasu *Main.java* za demonstraciju urađenih modifikacija nad projektom *course*.

✓ Poglavlje 6

Pokazna vežba 1 - Spring 4/5, Hibernate 5 i JPA 2

ZAHTEVI ZADATKA (45 MINUTA)

Neophodno je definisati zahteve po kojima se razvija aplikacija.

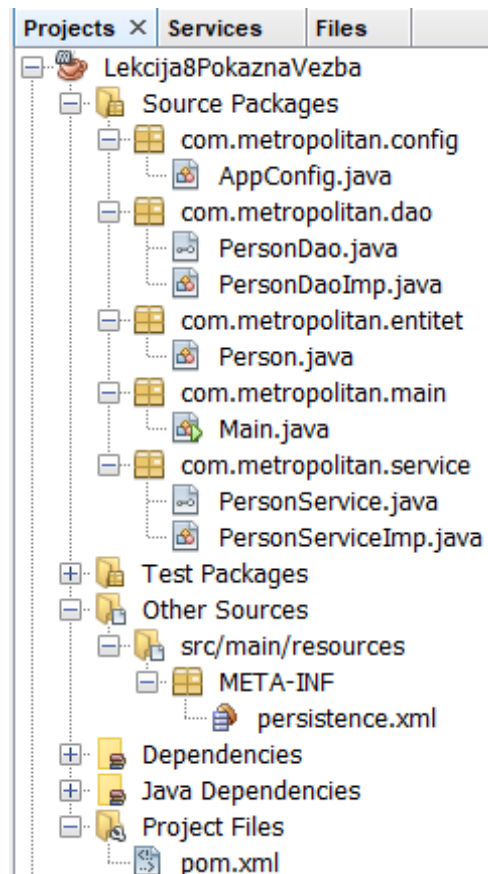
Cilj uvođenja ovog primera jeste utvrđivanje mehanizama primene Spring okvira zajedno sa JPA, koristeći okvir Hibernate kao provajdera perzistencije.

Prilikom izrade ovog zadatka biće korišćeni sledeći alati i tehnologije:

- [Spring](#) 5.1.0.RELEASE
- [Hibernate](#) 5.2.9.Final
- [JPA](#) 2.1
- Konektor za bazu podataka [postgresql-42.2.4](#) ili noviji
- DB server - [PostgreSQL9.6](#) ili noviji
- [NetBeans IDE 8.2](#)
- [Maven 3.5.1](#)
- [JavaSE 1.8](#)

U dosadašnjem izlaganju uglavnom je korišćen MySQL DBMS. Kroz praksu je pokazano da PostgreSQL se jako dobro slaže sa Spring okvirom po pitanju kreiranja i održavanja softvera koji koristi ORM koncepte u radu sa bazama podataka. Za ovaj DBMS kažu da je najnapredniji sistem za upravljanje bazom podataka otvorenog koda. Preuzimanje i instalacija su dostupni sa sledećeg linka: <https://www.postgresql.org/>

Sledećom slikom je data željena struktura pokaznog primera.



Slika 6.1 Struktura primera [izvor: autor]

ENTITETSKA KLASA ZA ORM MAPIRANJE

Sledi kreiranje klase za preslikavanje u tabelu baze podataka.

Sledeći zadatak koji će biti obavljen, jeste kreiranje Java klase koja će u ovom primeru biti reprezent konkretne tabele iz baze podataka. Ova klasa omogućava da se kreira objekat tipa *Person* koji je predmet perzistencije. Klasa je obeležena anotacijom *@Entity*, a njena polja anotacijama *@Id* i *@Column* ukazujući da se radi o primarnom ključu i ostalim kolonama tabele u koju se klasa mapira.

Klasa pripada zasebnom paketu i njen listing upravo sledi.

```
package com.metropolitan.entitet;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 *
```



```
* @author Vlada
*/
@Entity
@Table(name = "PERSONS")
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "FIRST_NAME")
    private String firstName;

    @Column(name = "LAST_NAME")
    private String lastName;

    @Column(name = "EMAIL")
    private String email;

    public Person() {
    }

    public Person(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
```

```

        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

DODAVANJE ZAVISNOSTI U PROJEKAT

Definisanje pom.xml datoteke za upravljanje zavisnostima

U prethodnoj sekciji je dat pregled alata i tehnologija koji će biti korišćeni prilikom razvoja ovog primera. Sledećom slikom su prikazane zavisnosti kojima je data podrška za kreiranje *konteksta Spring aplikacije*, primenu *Spring ORM* podokvira, korišćenje *Hibernate* okvira i *Postgresql* baze podataka, respektivno, što je i obeleženo različitim bojama.

```

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>5.1.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.1.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>5.1.0.RELEASE</version>
</dependency>
<!-- Spring ORM -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>5.1.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.2.9.Final</version>
</dependency>
<!-- Postgresql konektor -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.4</version>
</dependency>

```

Slika 6.2 Zavisnosti u pom.xml datoteci Spring projekta [izvor: autor]

Sledi kompletan listing *pom.xml* datoteke:

```

<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.metropolitan</groupId>
  <artifactId>Lekcija8PokaznaVezba</artifactId>

```

```
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<dependencies>
  <!-- Spring Context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>
  <!-- Spring ORM -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>

  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.9.Final</version>
  </dependency>
  <!-- Postgresql konektor -->
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.4</version>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-dbcp2</artifactId>
    <version>2.1.1</version>
  </dependency>
</dependencies>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<build>
  <sourceDirectory>src/main/java</sourceDirectory>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
    </plugin>
  </plugins>
</build>
```

```

        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
</plugins>
</build>
</project>

```

KREIRANJE DAO (DATA ACCESS OBJECT) KLASE

Sledi kreiranje klase za manipulisanje perzistentnim objektima.

Sledi kreiranje klase za manipulisanje perzistentnim objektima u servisnom nivou Spring aplikacije. Prvo će biti kreiran odgovarajući interfejs koji će omogućiti kreiranje novih perzistentnih objekata, a samim tim i njihovo čuvanje u bazi podataka, a zatim i prikazivanje liste objekata čije se informacije čuvaju u bazi podataka.

DAO interfejs i odgovarajuća implementaciona klasa, čuvaju se u vlastitom paketu. Sledi listing interfejsa.

```

package com.metropolitan.dao;

import com.metropolitan.entitet.Person;
import java.util.List;

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

    void add(Person person);

    List<Person> listPersons();
}

```

Implementaciona klasa će biti obeležena anotacijom [@Repository](#) i biće dostupna za pretragu kroz skeniranje komponenata. Dalje, anotacija [@PersistenceContext](#) će biti upotrebljena za umetanje menadžera entiteta u DAO klasu.

Sledi listing implementacione klase.

```

package com.metropolitan.dao;
import com.metropolitan.entitet.Person;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaQuery;

```

```
import javax.persistence.criteria.Root;

import org.springframework.stereotype.Repository;

/**
 *
 * @author Vlada
 */
@Repository
public class PersonDaoImp implements PersonDao {

    @PersistenceContext
    private EntityManager em;

    @Override
    public void add(Person person) {
        em.persist(person);
    }

    @Override
    public List<Person> listPersons() {
        CriteriaQuery<Person> criteriaQuery =
em.getCriteriaBuilder().createQuery(Person.class);
        @SuppressWarnings("unused")
        Root<Person> root = criteriaQuery.from(Person.class);
        return em.createQuery(criteriaQuery).getResultList();
    }
}
```

JEDINICA PERZISTENCIJE

Jedinica perzistencije podešava menadžera entiteta.

Jedinica perzistencije (eng. *persistence unit*) predstavlja XML datoteku karakterističnu za JPA aplikacije. Ova datoteka, pod nazivom *persistence.xml* kreira se u folderu projekta *src/main/java/META-INF* i ima veoma važan zadatak -podešava menadžera entiteta u okviru aplikativnog servera ili konzolne aplikacije i sadrži informacije o bazi podataka (korisničko ime, lozinku, url konekcije i slično) i ORM mapiranju.

Sledi listing datoteke *persistence.xml*.

U priloženom listingu se posebno ističe jedna linija gde osobina **hibernate.hbm2ddl.auto** ima vrednost **update**. To znači da Hibernate proverava postojeće tabele i kolone u bazi podataka. Ukoliko ne postoje on će ih automatski kreirati. Hibernate, u ovom slučaju ne briše postojeće tabele i nijedan podataka neće biti izgubljen nakon navedene provere.

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="local">
    <description> Spring Hibernate JPA Configuration Example</description>
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
      <property name="javax.persistence.jdbc.driver"
value="org.postgresql.Driver" /> <!-- DB Driver -->
      <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost/person" /> <!-- BD Mane -->
      <property name="javax.persistence.jdbc.user" value="postgres" /> <!--
DB User -->
      <property name="javax.persistence.jdbc.password" value="vlada" /> <!--
DB Password -->

      <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/> <!-- DB Dialect -->
      <property name="hibernate.hbm2ddl.auto" value="update" /> <!-- create /
create-drop / update -->

      <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in
console -->
      <property name="hibernate.format_sql" value="true" /> <!-- Show SQL
formatted -->
    </properties>

  </persistence-unit>
</persistence>
```

SPRING KONFIGURACIJA

Sledi kreiranje Java konfiguracione klase

Sledi kreiranje konfiguracione klase, obeležene anotacijom [@Configuration](#). Ova klasa bi trebalo da omogući i upravljanje transakcijama pa će biti obeležena i anotacijom [@EnableTransactionManagement](#), a takođe će omogućiti i skeniranje komponentata.

Klasa registruje dva zrna:

- produkciono zrno [LocalEntityManagerFactoryBean](#) - za instanciranje menadžera entiteta koji je podešen pomoću [persistence.xml](#);
- zrno [JpaTransactionManager](#) - za upravljanje transakcijama.

Sledi listing klase [AppConfig.java](#) koja se čuva u vlastitom paketu.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ComponentScans;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalEntityManagerFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableTransactionManagement
@ComponentScans(value = {
    @ComponentScan("com.metropolitan.dao")
    ,
    @ComponentScan("com.metropolitan.service")})
public class AppConfig {

    @Bean
    public LocalEntityManagerFactoryBean getEntityManagerFactoryBean() {
        LocalEntityManagerFactoryBean factoryBean = new
LocalEntityManagerFactoryBean();
        factoryBean.setPersistenceUnitName("local");
        return factoryBean;
    }

    @Bean
    public JpaTransactionManager getJpaTransactionManager() {
        JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(getEntityManagerFactoryBean().getObject()
);
        return transactionManager;
    }
}
```

KLASA ZA DEMONSTRACIJU

Neophodna je Main klasa za demonstraciju urađenog posla.

Neophodna je Main klasa za demonstraciju urađenog posla. Sledi njen listing:

```
package com.metropolitan.main;

import com.metropolitan.config.AppConfig;
import com.metropolitan.entitet.Person;
import com.metropolitan.service.PersonService;
import java.util.List;
```

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

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

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context
            = new AnnotationConfigApplicationContext(AppConfig.class);

        PersonService personService = context.getBean(PersonService.class);

        // Add Persons
        personService.add(new Person("Petar", "Peric", "pera@example.com"));
        personService.add(new Person("Jovan", "Jovanovic", "jova@example.com"));
        personService.add(new Person("Milica", "Milic", "mica@example.com"));

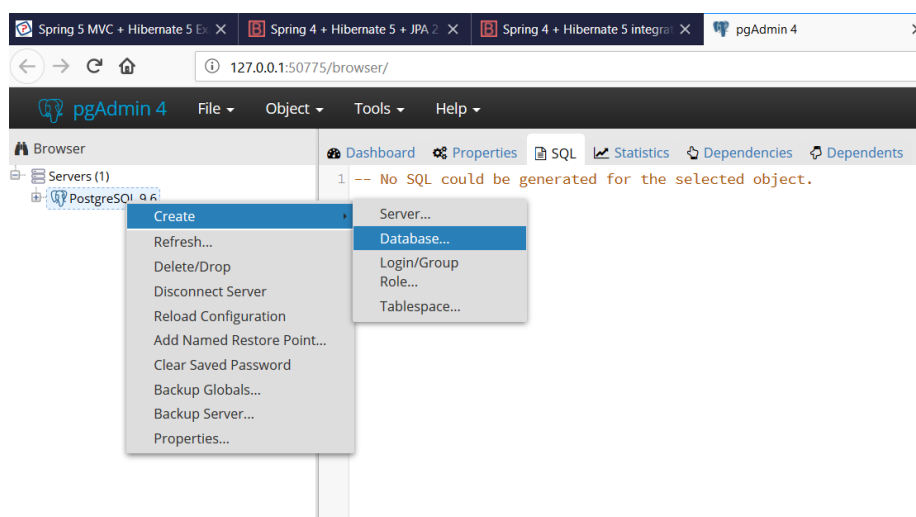
        // Get Persons
        List<Person> persons = personService.listPersons();
        for (Person person : persons) {
            System.out.println("Id = " + person.getId());
            System.out.println("First Name = " + person.getFirstName());
            System.out.println("Last Name = " + person.getLastName());
            System.out.println("Email = " + person.getEmail());
            System.out.println();
        }

        context.close();
    }
}
```

KREIRANJE BAZE PODATAKA

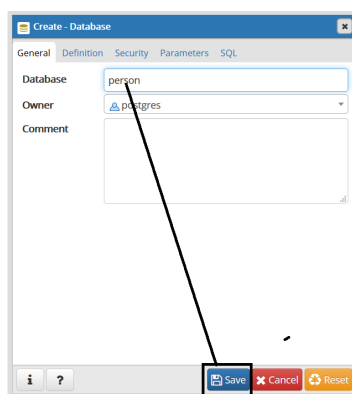
Kreira se baza podataka primenom PostgreSQL9.6 RDBMS.

Koristeći klijent [pgAdmin4](#) dostupna je klijent veb aplikacija servera baze podataka [PostgreSQL9.6](#). Veoma jednostavno klikom na čvor servera, a zatim i desnim klikom biramo opciju za kreiranje nove baze podataka, baš kao na sledećoj slici.



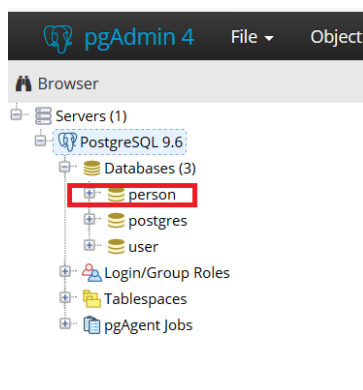
Slika 6.3 Kreiranje nove baze podataka. [izvor: autor]

Sada je dostupan nov prozor u kojem unosimo željeni naziv za bazu podataka, a zatim je čuvamo baš kao na sledećoj slici.



Slika 6.4 Dodela naziva bazi podataka [izvor: autor]

Konačno, baza podataka je dostupna za kreiranje tabela i dalji rad (sledeća slika).



Slika 6.5 Kreirana baza podataka [izvor: autor]

DEMONSTRACIJA KREIRANE APLIKACIJE

Pokreće se program i prate se rezultati njegovog izvršavanja.

Konačno, program se prevodi, pokreće i prate se rezultati njegovog izvršavanja. Izlaz na konzoli je prikazan sledećom slikom.

```
INFO: Using DataSource [org.apache.commons.dbcp2.BasicDataSource@6af93788] of Hibernate SessionFactory for HibernateTransactionManager
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
Okt 20, 2018 11:31:56 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select user0_.id as id1_0_, user0_.EMAIL as EMAIL2_0_, user0_.FIRST_NAME as FIRST_NAME3_0_, user0_.LAST_NAME as LAST_NAME4_0_ from
  users user0_
  where user0_.id = ?
First Name = Vlada
Last Name = Milicevic
Email = vlada@example.com

Id = 9
First Name = Zoran
Last Name = Zoranovic
Email = zoki@example.com

Id = 10
First Name = Milena
Last Name = Milic
Email = mila@example.com

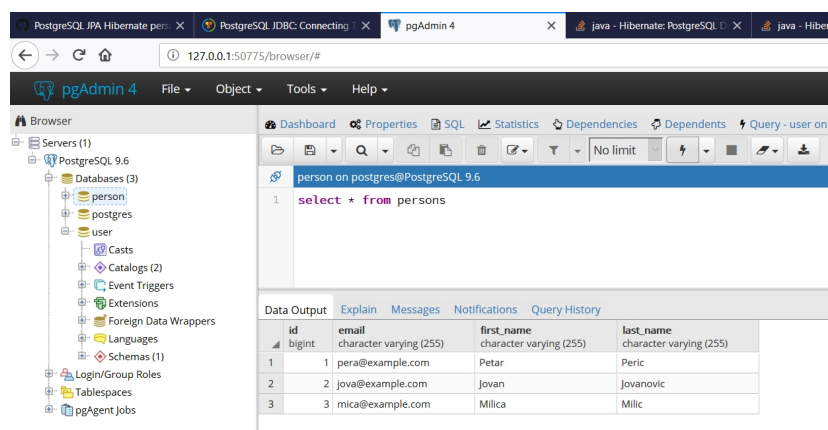
Okt 20, 2018 11:31:56 PM org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
INFO: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@7daf6ecc: startup date [Sat Oct 20 23:31:52 CEST 2018]
BUILD SUCCESS

Total time: 5.165s
Finished at: Sat Oct 20 23:31:56 CEST 2018
Final Memory: 6M/135M
```

Slika 6.6 Rezultat izvršavanja aplikacije na konzoli [izvor: autor]

Sada možemo proveriti i da li odgovarajući zapisi postoje u bazi podataka, a to je prikazano slikama 7 i 8.

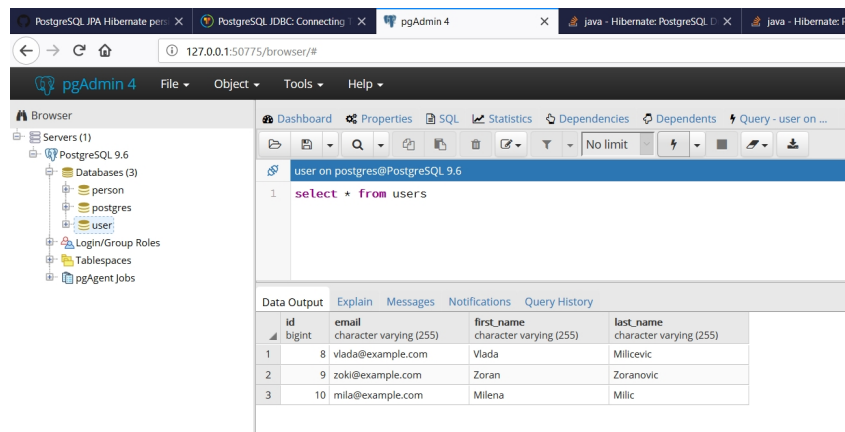
Kompletno urađen zadatak je dostupan za preuzimanje na kraju ovog objekta učenja.



The screenshot shows the pgAdmin 4 web interface. On the left, a tree view shows the database structure. In the center, a SQL query is entered: `select * from persons`. On the right, the 'Data Output' tab displays the results of the query in a table format.

id	email	first_name	last_name
1	pera@example.com	Petar	Peric
2	jova@example.com	Jovan	Jovanovic
3	mica@example.com	Milica	Milic

Slika 6.7 Provera pisanjem SQL upita [izvor: autor]



Slika 6.8 Zapisi sačuvani u bazi podataka [izvor: autor]

VIDEO MATERIJAL

How to use Spring JPA with PostgreSQL | Spring Boot

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

▼ Poglavlje 7

Pokazna vežba 2 - Spring 4/5, Hibernate 5 bez XML

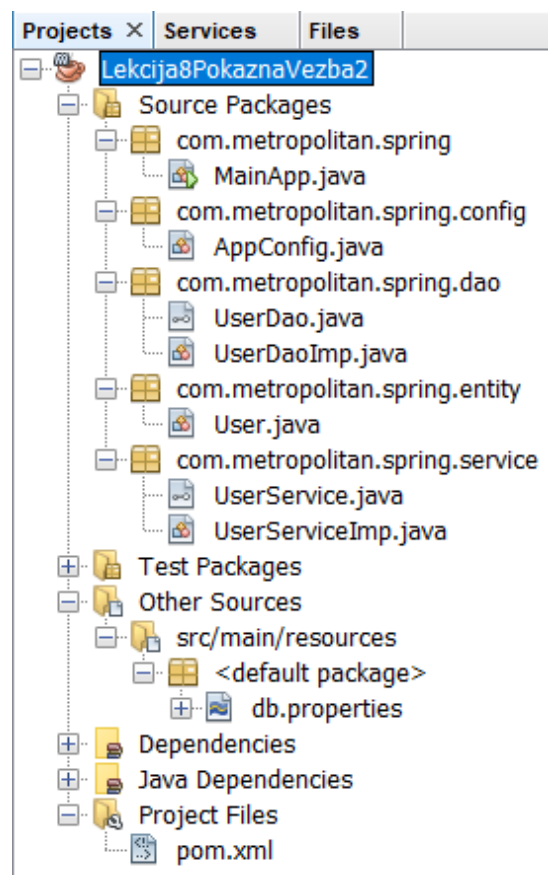
ZAHTJEVI ZADATKA (45 MIN)

Neophodno je definisati zahteve po kojima se razvija aplikacija.

Cilj uvođenja ovog primera jeste utvrđivanje mehanizama primene Spring okvira kroz integraciju sa ORM okvirom Hibernate bez imalo XML koda. Prilikom izrade ovog zadatka biće korišćeni sledeći alati i tehnologije:

- [Spring](#) 5.1.0.RELEASE
- [Hibernate](#) 5.2.9.Final
- Konektor za bazu podataka [postgresql-42.2.4](#) ili noviji
- DB server - [PostgreSQL9.6](#) ili noviji
- [NetBeans IDE 8.2](#)
- [Maven 3.5.1](#)
- [JavaSE 1.8](#)

Sledećom slikom je data željena struktura pokaznog primera.



Slika 7.1 Struktura Maven Spring projekta [izvor: autor]

DODAVANJE ZAVISNOSTI U PROJEKAT

Definisanje pom.xml datoteke za upravljanje zavisnostima

U prethodnoj sekciji je dat pregled alata i tehnologija koji će biti korišćeni prilikom razvoja ovog primera. Sledećom slikom su prikazane zavisnosti kojima je data podrška za kreiranje *konteksta Spring aplikacije*, primenu *Spring ORM* podokvira, korišćenje *Hibernate* okvira i *Postgresql* baze podataka, respektivno.

Sledi kompletan listing *pom.xml* datoteke:

```
<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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.boraji.tutorial.hibernate</groupId>
    <artifactId>Lekcija8PokaznaVezba2</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>Lekcija8PokaznaVezba2</name>
```

```
<url>http://maven.apache.org</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
  <!-- Spring Context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>
  <!-- Spring ORM -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.0.RELEASE</version>
  </dependency>
  <!-- Postgresql Connector -->
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.4</version>
  </dependency>
  <!-- Hibernate 5.2.9 Final -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.9.Final</version>
  </dependency>
  <!-- Apache Commons DBCP -->
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-dbcp2</artifactId>
    <version>2.1.1</version>
  </dependency>
</dependencies>

<build>
  <sourceDirectory>src/main/java</sourceDirectory>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

ENTITETSKA KLASA ZA ORM MAPIRANJE

Sledi kreiranje klase za preslikavanje u tabelu baze podataka.

Sledeći zadatak koji će biti obavljen, jeste kreiranje Java klase *User.java* koja će u ovom primeru biti reprezent konkretne tabele iz baze podataka. Ova klasa omogućava da se kreira objekat tipa *Person* koji je predmet perzistencije. Klasa je obeležena anotacijom *@Entity*, a njena polja anotacijama *@Id* i *@Column* ukazujući da se radi o primarnom ključu i ostalim kolonama tabele u koju se klasa mapira.

Klasa pripada zasebnom paketu i njen listing upravo sledi.

```
package com.metropolitan.spring.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "USERS")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "FIRST_NAME")
    private String firstName;

    @Column(name = "LAST_NAME")
    private String lastName;

    @Column(name = "EMAIL")
    private String email;

    public User() {}

    public User(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
}
```

DAO KLASA

Kreira se DAO nivo aplikacije

U nastavku se kreira nivo aplikacije čija je implementaciona klasa obeležena [@Repository](#). Prvo je neophodno priložiti odgovarajući interfejs.

```
package com.metropolitan.spring.dao;

import java.util.List;

import com.metropolitan.spring.entity.User;

public interface UserDao {

    void add(User user);

    List<User> listUsers();

}
```

Za dodavanje novih korisnika i prikazivanje postojećih neophodno je redefinisati metode interfejsa. Sledi implementaciona klasa koja je smeštena u isti paket kao i interfejs.

U implementacionu klasu je umetnuto zrno [Hibernate produkcije sesije](#) anotacijom [@Autowired](#). Ovaj objekat za tekuću sesiju izvodi operacije perzistencije u metodama implementacione klase.

```
package com.metropolitan.spring.dao;

import java.util.List;

import javax.persistence.TypedQuery;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.metropolitan.spring.entity.User;

/**
 * @author Vlada
 *
 */
@Repository
public class UserDaoImp implements UserDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void add(User user) {
        sessionFactory.getCurrentSession().save(user);
    }

    @Override
    public List<User> listUsers() {
        @SuppressWarnings("unchecked")

```



```

        TypedQuery<User> query =
sessionFactory.getCurrentSession().createQuery("from User");
        return query.getResultList();
    }
}

```

SERVISNI NIVO APLIKACIJE

Kreiranje interfejsa i implementacione klase servisnog nivoa.

Sledeći korak jeste kreiranje interfejsa i implementacione klase servisnog nivoa. Prvo je priložen listing servisnog interfejsa.

```

package com.metropolitan.spring.service;

import java.util.List;

import com.metropolitan.spring.entity.User;

public interface UserService {

    void add(User user);

    List<User> listUsers();
}

```

Implementaciona klasa umeće zrno *UserDao* i obeležava sve metode anotacijom *@Transactional* o kojoj je bilo reči u prethodnom teorijskom izlaganju.

```

package com.metropolitan.spring.service;

import com.metropolitan.spring.dao.UserDao;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.metropolitan.spring.entity.User;

/**
 * @author Vlada
 *
 */
@Service
public class UserServiceImp implements UserService {

    @Autowired

```

```
private UserDao userDao;

@Transactional
@Override
public void add(User user) {
    userDao.add(user);
}

@Transactional(readOnly = true)
@Override
public List<User> listUsers() {
    return userDao.listUsers();
}
}
```

DATOTEKA OSOBINA I KONFIGURISANJE APLIKACIJE

Podešavanje baze podataka je obavljeno eksterno u posebnoj datoteci.

U tekućem primeru, podešavanje baze podataka je obavljeno eksterno u posebnoj datoteci. Ova datoteka je nazvana *db.properties* čuva se na lokaciji *src/main/resources* i ima sledeći listing:

```
# PostgreSQL properties
db.driver=org.postgresql.Driver
db.url=jdbc:postgresql://localhost:5432/user
db.username=postgres
db.password=vlada

# Hibernate properties
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Konačno, neophodno je podesiti zrna za izvor podatka, produkciju sesije i upravljanje Hibernate transakcijama. Sledi listing konfiguracione klase projekta. Moguće je primetiti da su ova klasa i datoteka *db.properties* u direktnoj vezi.

```
package com.metropolitan.spring.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ComponentScans;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import com.metropolitan.spring.entity.User;
/**
 * @author Vlada
 *
 */
@Configuration
@PropertySource("classpath:db.properties")
@EnableTransactionManagement
@ComponentScans(value = {
    @ComponentScan("com.metropolitan.spring.dao"),
    @ComponentScan("com.metropolitan.spring.service")
})
public class AppConfig {

    @Autowired
    private Environment env;

    @Bean
    public DataSource getDataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName(env.getProperty("db.driver"));
        dataSource.setUrl(env.getProperty("db.url"));
        dataSource.setUsername(env.getProperty("db.username"));
        dataSource.setPassword(env.getProperty("db.password"));
        return dataSource;
    }

    @Bean
    public LocalSessionFactoryBean getSessionFactory() {
        LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
        factoryBean.setDataSource(getDataSource());

        Properties props=new Properties();
        props.put("hibernate.show_sql", env.getProperty("hibernate.show_sql"));
        props.put("hibernate.hbm2ddl.auto",
env.getProperty("hibernate.hbm2ddl.auto"));

        factoryBean.setHibernateProperties(props);
        factoryBean.setAnnotatedClasses(User.class);
        return factoryBean;
    }

    @Bean
    public HibernateTransactionManager getTransactionManager() {
        HibernateTransactionManager transactionManager = new
        HibernateTransactionManager();
    }
}
```

```
        transactionManager.setSessionFactory(getSessionFactory().getObject());  
        return transactionManager;  
    }  
}
```

MAIN KLASA I DEMONSTRACIJA

Neophodno je kreirati klasu koja će omogućiti demonstraciju poziva perzistentnih metoda.

Kao i u prethodnom primeru, neophodna je klasa koja će omogućiti demonstraciju poziva perzistentnih metoda. Sledi njen listing:

```
package com.metropolitan.spring;  
  
import com.metropolitan.spring.config.AppConfig;  
import java.sql.SQLException;  
import java.util.List;  
  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import com.metropolitan.spring.entity.User;  
import com.metropolitan.spring.service.UserService;  
  
/**  
 * @author Vlada  
 *  
 */  
public class MainApp {  
    public static void main(String[] args) throws SQLException {  
        AnnotationConfigApplicationContext context =  
            new AnnotationConfigApplicationContext(AppConfig.class);  
  
        UserService userService = context.getBean(UserService.class);  
  
        // Add Users  
        userService.add(new User("Vlada", "Milicevic", "vlada@example.com"));  
        userService.add(new User("Zoran", "Zoranovic", "zoki@example.com"));  
        userService.add(new User("Milena", "Milic", "mila@example.com"));  
  
        // Get Users  
        List<User> users = userService.listUsers();  
        for (User user : users) {  
            System.out.println("Id = "+user.getId());  
            System.out.println("First Name = "+user.getFirstName());  
            System.out.println("Last Name = "+user.getLastName());  
            System.out.println("Email = "+user.getEmail());  
            System.out.println();  
        }  
    }  
}
```

```
context.close();
}
}
```

Sledećom slikom je prikazan izlaz na konzoli.

```
INFO: Using DataSource [org.apache.commons.dbcp2.BasicDataSource@6af93788] of Hibernate SessionFactory
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
Hibernate: insert into USERS (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
okt 21, 2018 10:40:26 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH0000397: Using ASTQueryTranslatorFactory
Hibernate: select user0 .id as idl_0_, user0_.EMAIL as EMAIL2_0_, user0_.FIRST_NAME as FIRST_NAME3_0_, u
Id = 8
First Name = Vlada
Last Name = Milicevic
Email = vlada@example.com
Id = 9
First Name = Zoran
Last Name = Zoranovic
Email = zoki@example.com
Id = 10
First Name = Milena
Last Name = Milic
Email = mila@example.com
```

Slika 7.2 Izlaz na konzoli [izvor: autor]

Konačno je proverena i baza podataka i tamo postoje odgovarajući zapisi.

Data Output	Explain	Messages	Notifications	Query History
id bigint	email character varying (255)	first_name character varying (255)	last_name character varying (255)	
1	8 vlada@example.com	Vlada	Milicevic	
2	9 zoki@example.com	Zoran	Zoranovic	
3	10 mila@example.com	Milena	Milic	

Slika 7.3 Sačuvani zapisi u bazi podataka [izvor: autor]

Preuzmite kompletno urađeni primer odmah iza ovog objekta učenja.

▼ Poglavlje 8

Individualna vežba 8 - Spring MVC i Hibernate

ZADATAK (90 MIN)

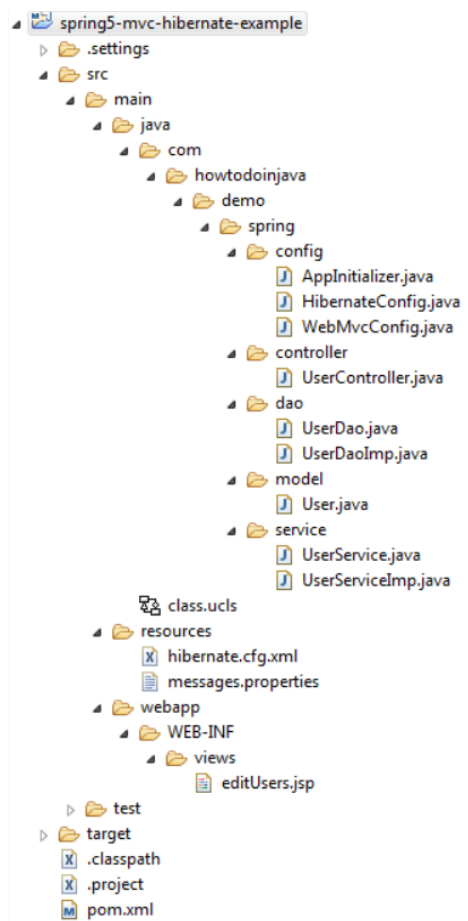
Individualni rad na razvoju Spring MVC aplikacije sa Hibernate.

Cilj uvođenja ovog primera jeste utvrđivanje mehanizama primene Spring MVC okvira kroz integraciju sa ORM okvirom *Hibernate* . Prilikom izrade ovog zadatka biće korišćeni sledeći alati i tehnologije:

- [Spring](#) 5.1.0.RELEASE
- [Hibernate](#) 5.2.9.Final
- Konektor za bazu podataka [postgresql-42.2.4](#)
- DB server - [PostgreSQL9.6](#) ili noviji
- [NetBeans IDE 8.2](#)
- [Maven 3.5.1](#)
- [JavaSE 1.8](#)
- [Hibernate validator 5.4.1.Final](#)
- [Servlets 3.1.0](#)
- [Tomcat 7 maven plugin 2.2](#)

Vaš prvi zadatak je da kreirate *pom.xml*. Konačna struktura projekta bi trebalo da bude kao na sledećoj slici.

Urađeni zadatak u prilogu ovog objekta učenja bi trebalo da vas vodi ka konačnom rešenju. Zahtevi se ne podudaraju u potpunosti.

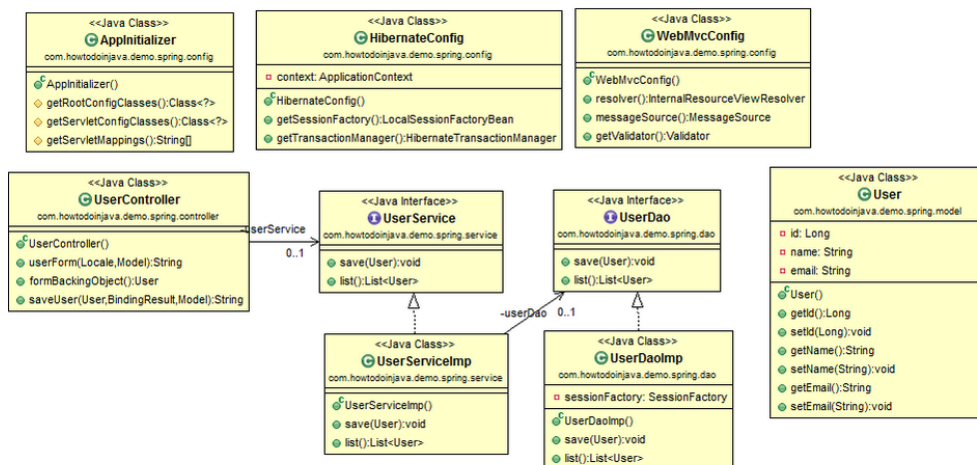


Slika 8.1 Ciljana struktura projekta [izvor: autor]

DIJAGRAM KLASA

Još malo diskusije u vezi sa zahtevima

Sledeći dijagram klasa bi trebalo da odgovara konačnom rešenju.



Slika 8.2 Dijagram klasa rešenja [izvor: autor]

PODEŠAVANJE APLIKACIJE - DISPATCHERSERVLET I SPRING MVC PODEŠAVANJA

DispatcherServlet i Spring MVC podešavanja

Predlog podešavanja servleta.

```
public class AppInitializer extends
    AbstractAnnotationConfigDispatcherServletInitializer {

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

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

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

Predlog podešavanja za Spring MVC

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.naziv.vaseg.paketa" })
public class WebMvcConfig implements WebMvcConfigurer {

    @Bean
    public InternalResourceViewResolver resolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setViewClass(JstlView.class);
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource source = new ResourceBundleMessageSource();
        source.setBasename("messages");
        return source;
    }

    @Override
    public Validator getValidator() {
```



```

        LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
        validator.setValidationMessageSource(messageSource());
        return validator;
    }
}

```

PODEŠAVANJE APLIKACIJE - HIBERNATE

Hibernate podešavanja

Predlog izgleda konfiguracione datoteke *HibernateConfig.java*.

```

@Configuration
@EnableTransactionManagement
public class HibernateConfig {

    @Autowired
    private ApplicationContext context;

    @Bean
    public LocalSessionFactoryBean getSessionFactory() {
        LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
        factoryBean.setConfigLocation(context.getResource("classpath:hibernate.cfg.xml"));
        factoryBean.setAnnotatedClasses(User.class);
        return factoryBean;
    }

    @Bean
    public HibernateTransactionManager getTransactionManager() {
        HibernateTransactionManager transactionManager = new
        HibernateTransactionManager();
        transactionManager.setSessionFactory(getSessionFactory().getObject());
        return transactionManager;
    }
}

```

Predlog izgleda konfiguracione datoteke *hibernate.cfg.xml*. Neophodno je da unesete vaše podatke za korisničko ime, lozinku i naziv baze podataka.

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.archive.autodetection">class,hbm</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="hibernate.show_sql">true</property>
    
```

```
<property
name="hibernate.connection.driver_class">org.postgresql.Driver</property>
  <property name="hibernate.connection.username">vasUserName</property>
  <property name="hibernate.connection.password">vasPassword</property>
  <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/
dbname</property>
  <property name="hibernate.hbm2ddl.auto">create</property>

  <property name="hibernate.c3p0.min_size">5</property>
  <property name="hibernate.c3p0.max_size">20</property>
  <property name="hibernate.c3p0.acquire_increment">2</property>
  <property name="hibernate.c3p0.acquire_increment">1800</property>
  <property name="hibernate.c3p0.max_statements">150</property>
</session-factory>
</hibernate-configuration>
```

Modifikujtenaknadnoovaj primer tako što ćete podatke baze podataka čuvati u datoteci osobina.

SPRING KONTROLER I MAPIRANJA

Kreiranje kontrolera za upravljanje formama sa JSP stranica

Predlog izgleda MVC kontrolera:

```
@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/")
    public String userForm(Locale locale, Model model) {
        model.addAttribute("users", userService.list());
        return "editUsers";
    }

    @ModelAttribute("user")
    public User formBackingObject() {
        return new User();
    }

    @PostMapping("/addUser")
    public String saveUser(@ModelAttribute("user") @Valid User user,
                           BindingResult result, Model model) {

        if (result.hasErrors()) {
            model.addAttribute("users", userService.list());
            return "editUsers";
        }
    }
}
```

```

        userService.save(user);
        return "redirect:/";
    }
}

```

Predlog izgleda JSP stranice:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>Spring5 MVC Hibernate Demo</title>
        <style type="text/css">
            .error {
                color: red;
            }
            table {
                width: 50%;
                border-collapse: collapse;
                border-spacing: 0px;
            }
            table td {
                border: 1px solid #565454;
                padding: 20px;
            }
        </style>
    </head>
    <body>
        <h1>Input Form</h1>
        <form:form action="addUser" method="post" modelAttribute="user">
            <table>
                <tr>
                    <td>Name</td>
                    <td>
                        <form:input path="name" /> <br/>
                        <form:errors path="name" cssClass="error" />
                    </td>
                </tr>
                <tr>
                    <td>Email</td>
                    <td>
                        <form:input path="email" /> <br/>
                        <form:errors path="email" cssClass="error" />
                    </td>
                </tr>
                <tr>
                    <td colspan="2"><button type="submit">Submit</button></td>
                </tr>
            </table>

```

```

</form:form>

<h2>Users List</h2>
<table>
  <tr>
    <td><strong>Name</strong></td>
    <td><strong>Email</strong></td>
  </tr>
  <c:forEach items="${users}" var="user">
    <tr>
      <td>${user.name}</td>
      <td>${user.email}</td>
    </tr>
  </c:forEach>
</table>
</body>
</html>

```

SERVIS I DAO

Predlog servisnog i DAO nivoa

Predlog servisnog nivoa:

```

//interfejs
public interface UserService {
    void save(User user);

    List<User> list();
}

//implementaciona klasa
@Service
public class UserServiceImp implements UserService {

    @Autowired
    private UserDao userDao;

    @Transactional
    public void save(User user) {
        userDao.save(user);
    }

    @Transactional(readOnly = true)
    public List<User> list() {
        return userDao.list();
    }
}

```

Predlog DAO nivoa:

```
//interfejs
public interface UserDao {
    void save(User user);
    List<User> list();
}

// implemenatciona klasa
@Repository
public class UserDaoImp implements UserDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void save(User user) {
        sessionFactory.getCurrentSession().save(user);
    }

    @Override
    public List<User> list() {
        @SuppressWarnings("unchecked")
        TypedQuery<User> query = sessionFactory.getCurrentSession().createQuery("from
User");
        return query.getResultList();
    }
}
```

ENTITET I RESURS PORUKA

Još malo kodiranja za kraj aplikacije

Predlog izgleda entitetske klase:

```
@Entity
@Table(name = "nazivVaseTabele")
public class User {

    @Id
    @GeneratedValue
    @Column(name = "USER_ID")
    private Long id;

    @Column(name = "USER_NAME")
    @Size(max = 20, min = 3, message = "{user.name.invalid}")
    @NotEmpty(message="Please Enter your name")
    private String name;

    @Column(name = "USER_EMAIL", unique = true)
    @Email(message = "{user.email.invalid}")
```

```
@NotEmpty(message="Please Enter your email")
private String email;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}
}
```

Predlaže se dodavanje i datoteke *messages.properties*:

```
user.name.invalid = Name must be between {2} and {1} characters.
user.email.invalid = Please enter valid email address.
```

POČETNI EKRAN APLIKACIJE

Predlog izgleda početnog ekrana

Već je priložen predloženi kod JSP pogleda na osnovu kojeg bi početni ekran trebalo da ima sledeći izgled.

Input Form

Name	<input type="text"/>
Email	<input type="text"/>
<input type="submit" value="Submit"/>	

Users List

Name	Email
------	-------

Slika 8.3 Predlog izgleda početnog ekrana [izvor: autor]

INDIVIDUALNA VEŽBA - VIDEO MATERIJAL

Spring 5 MVC + Hibernate 5 Example

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

▼ Poglavlje 9

Domaći zadatak 8

DOMAĆI ZADATAK (120 MIN)

Cilj domaćeg zadatka je da student provežba naučeno na vežbama

1. Preuzmite primer iz Lekcije Spring MVC za rezervaciju terena.
2. Omogućite čuvanje korisnika u bazi podataka integracijom sa Spring ORM okvirom;
3. Napraviti JSP stranice za dodavanje i brisanje korisnika, izmenu podataka o korisnicima i prikazivanje liste svih korisnika;
4. Podesiti bazu podataka u datoteci osobina;
5. Koristiti PostgreSQL ili MySQL po želji;

Nakon obaveznog zadatka, studenti na mail dobijaju dodatne različite zadatke od predmetnog asistenta.

▼ Poglavlje 10

Zaključak

ZAKLJUČAK

Lekcija 08 bavila se scenarijima pristupa podacima podržanih Springom.

Ova lekcija se bavila načinima podrške koju Spring ovih obezbeđuje za *Hibernate* i *JPA* implementacije objektno - relacionog mapiranja. Naučeno je kako se podešava *DataSource* za pristup bazi podataka i kako se koriste Springovi mehanizmi *HibernateTemplate* i *JpaTemplate* za oslobađanje koda od obimnog i napornog rukovanja.

Posebno je pokazano kako se koriste alati osnovnih klasa za kreiranje DAO klasa sa *Hibernate* i *JPA* i, takođe, kako se koristi Spring podrška za stereotipne anotacije i skeniranje komponenta za lakšu izgradnju DAO klasa. Celokupna Spring podrška podešena je primenom savremenog pristupa baziranog na primeni Java konfiguracija.

Takođe, pokazano je i elegantno rukovanje i preslikavanje Hibernate i JPA izuzetaka u izuzetke *Spring DataAccessException* hijerarhije.

LITERATURA

U pripremanju Lekcije 08 korišćena je najaktuelnija pisana i web literatura.

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura:

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