Natječaj za radno mjesto “SW arhitekt”

Izrada jednostavne J2EE aplikacije

Tehnička dokumentacija

Verzija 1.0

Silvio Vuk

2.7.2013

Sadržaj

Popis kratica:

AOP - Aspect-Oriented Programming
API - Application Programming Interface
DBMS - Database Management System
GUI - Graphical User Interface
JDO - Java Data Objects
JMX - Java Management Extensions
JPA - Java Persistence API
JSP - JavaServer Pages
MVC - Model–View–Controller
OJB - Apache ObJectRelationalBridge
POJO - Plain Old Java Object
POM - Project Object Model
SQL - Structured Query Language
URL - Uniform Resource Locator

1. Opis aplikacije

U ovome testnom projektu, uz pomoć tehnologija navedenih kasnije u dokumentu, korisnik može dohvatiti podatke preko udaljenog servisa. Odabirom određene opcije aplikacija pokušava preuzeti podatke sa servisa. Ukoliko je servis dostupan, proslijedit će podatke klijentskom dijelu aplikacije te ih prikazati korisniku. U slučaju kada servis nije dostupan, korisniku će se ispisati informativna poruka da je servis trenutno nedostupan.

2. Tehničke značajke

Web aplikacija je rađena uz pomoć Spring *frameworka* koji pruža između ostalog MVC pristup izrade. Cjelovita web aplikacija se sastoji od tri dijela.

a. Zajednički dio sadrži POJO klase i servise koji se koriste u klijentskom i serverskom dijelu aplikacije.
b. Serverski dio sadrži klase koje definiraju Hibernateu koje podatke je potrebno izvući za određenu metodu, te implementaciju DAO servisa.
c. Klijentski dio pruža korisniku sučelje u kojem uz pomoć *buttona* poziva udaljeni servis, odnosno serverski dio koji dohvaća podatke iz baze podataka

U paketu “silvio.vuk.vestigo.pojos” nalaze se dvije datoteke: “Mjesto” i “Podaci”. Svaka od njih pomoću anotacija opisuje Springu odnosno Hibernateu na koje vrijednosti u bazi podatak se veže određeni atribut.

Mjesto.java

```
@Entity
@Table(name="VESTIGO.MJESTO")
public class Mjesto implements Serializable {
	private static final long serialVersionUID = -8559984336910202859L;

	@Id
	@Column(name = "ID")
	@GeneratedValue
	private int ID;

	@Column(name = "NAZIV")
	private String naziv;

	public Mjesto(String naziv)
	{
		this.naziv = naziv;
	}

	public Mjesto(){}

	public int getID() {
		return ID;
	}
	public void setID(int iD) {
		ID = iD;
	}
	public String getNaziv() {
		return naziv;
	}
	public void setNaziv(String naziv) {
```

```
            this.naziv = naziv;
      }
}
```

U primjeru “Mjesto.java” označeno je da klasa Mjesto predstavlja tablicu MJESTO koja se nalazi u bazi VESTIGO. Privatni atribut ID predstavlja atribut u bazi pod imenom “ID”, anotacije @ID i @GeneratedValue definiraju da se radi o indentifikacijskoj vrijednosti koja se sama generira (pojam *auto-increment* u SQL-u).

ServerVestigo/src/main/webapp/WEB-INF/PodaciServices-servlet.xml

```
<context:component-scan base-package="silvio.vuk.vestigo.services"/>
<bean id="podaciServicesServer"
class="silvio.vuk.vestigo.servicesImpl.PodaciServicesImplementation"></bean>
<bean name="/PodaciServices"
class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service" ref="podaciServicesServer"/>
  <property name="serviceInterface"
  value="silvio.vuk.vestigo.services.PodaciServices"/>
</bean>
<jdbc:embedded-database id="baza" type="H2">
  <jdbc:script location="classpath:schema.sql"/>
  <jdbc:script location="classpath:data.sql"/>
</jdbc:embedded-database>
<bean id="vestigoRepository"
class="silvio.vuk.vestigo.dao.VestigoDaoHibernate">
  <constructor-arg ref="sessionFactory"/>
</bean>
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<tx:annotation-driven transaction-manager="transactionManager"/>
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactor
yBean">
  <property name="dataSource" ref="baza"/>
  <property name="annotatedClasses">
    <list>
      <value>silvio.vuk.vestigo.pojos.Mjesto</value>
      <value>silvio.vuk.vestigo.pojos.Podaci</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <value>
      hibernate.format_sql=true hibernate.show_sql=true
```

```
    hibernate.dialect=org.hibernate.dialect.H2Dialect
   </value>
  </property>
</bean>
```

U gore navedenoj konfiguracijskoj datoteci, definirani su podaci potrebni za uspostavu servisa i dohvata podataka iz baze.

Za testiranje uzeta je *in memory* baza (jdbc:embedded-database). Definiraju se putanje skripta za schemu i punjenje baze podacima. Za to se koristi H2 baza podataka.

Pomoću "HttpInvokerServiceExporter" klase omogućeno je da određeni servis definiran pomoću parametara, bude dostupan kao remote servis.(podržan samo Java to Java *remoting*)

Session Factory povezuje bazu sa klasama koje predstavljaju entitet za tablice. Klase moraju biti navedeni kao parametar "annotatedClasses".

Bean "vestigoRepository" povezuje Bean "sessionFactory" sa klasom koja implementira DAO *interface* uz pomoć Hibernatea.

Bean "transactionManager" omogućuje korištenje transakcija nad određenim klasama odnosno metodama definiranim u Session Factoryu.

U klijentskom dijelu aplikacije nalazi se potreban GUI (za ljepši prikaz korisniku koristi se Twitter Bootstrap). Za pristup određenom *view* dijelu aplikacije brine se "HomeController". Pomoću anotacije "@RequestMapping" određuje se na koji URL da se poziva koja metoda. Zatim ta određena metoda obavlja logički dio te vraća ime JSP datoteke koja će se prikazati korisniku. Kako bi se skratilo pisanje cijelih putanji do ".jsp" datoteka uvedena je konfiguracijska datoteka "servlet-context.xml".

ClientVestigo/src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
  <beans:bean
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/"/>
    <beans:property name="suffix" value=".jsp"/>
  </beans:bean>
  <context:component-scan base-package="silvio.vuk.vestigo"/>
</beans:beans>
```

Dohvat podataka preko *remote* servisa iz serverskog djela aplikacije definiran je u

“app-config.xml” datoteci.

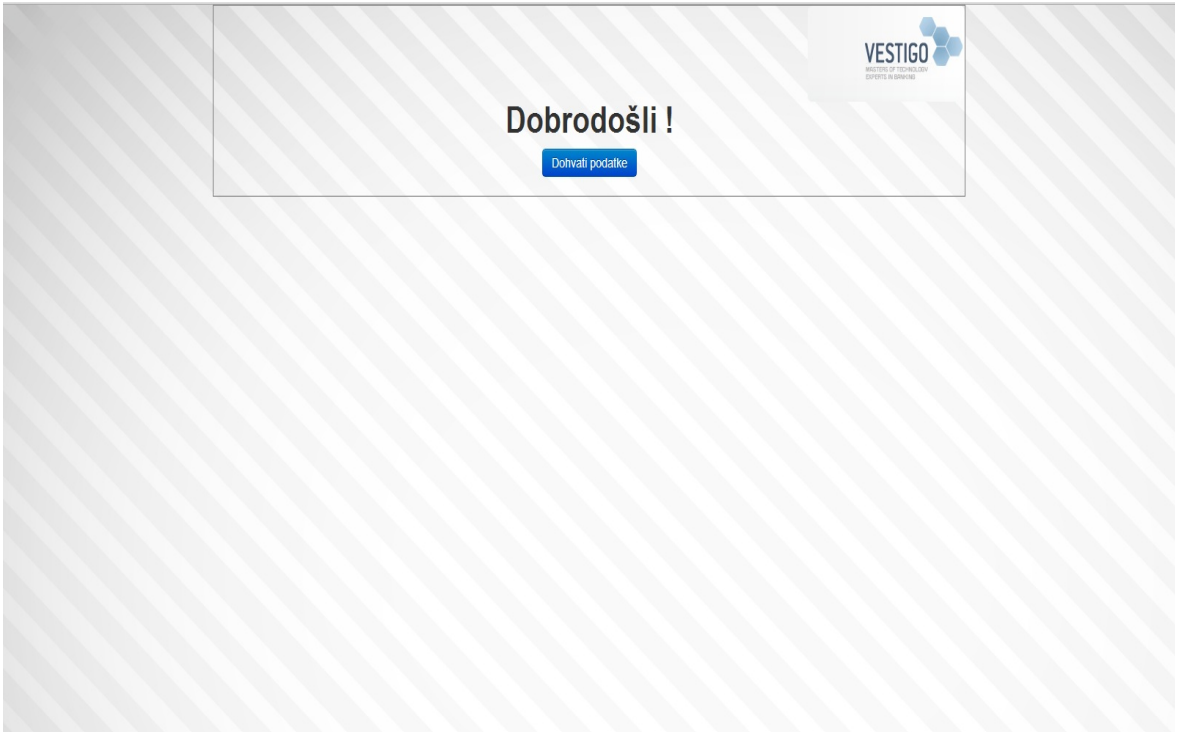
ClientVestigo/src/main/webapp/WEB-INF/app-config.xml

```
  <mvc:annotation-driven/>
  <context:component-scan base-package="silvio.vuk.vestigo.controllers"/>
  <bean id="vestigoServiceClient"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBea
  n">
    <property name="serviceUrl"
    value="http://localhost:8081/vestigo/services/PodaciServices"/>
    <property name="serviceInterface"
    value="silvio.vuk.vestigo.services.PodaciServices"/>
  </bean>
</beans>
```

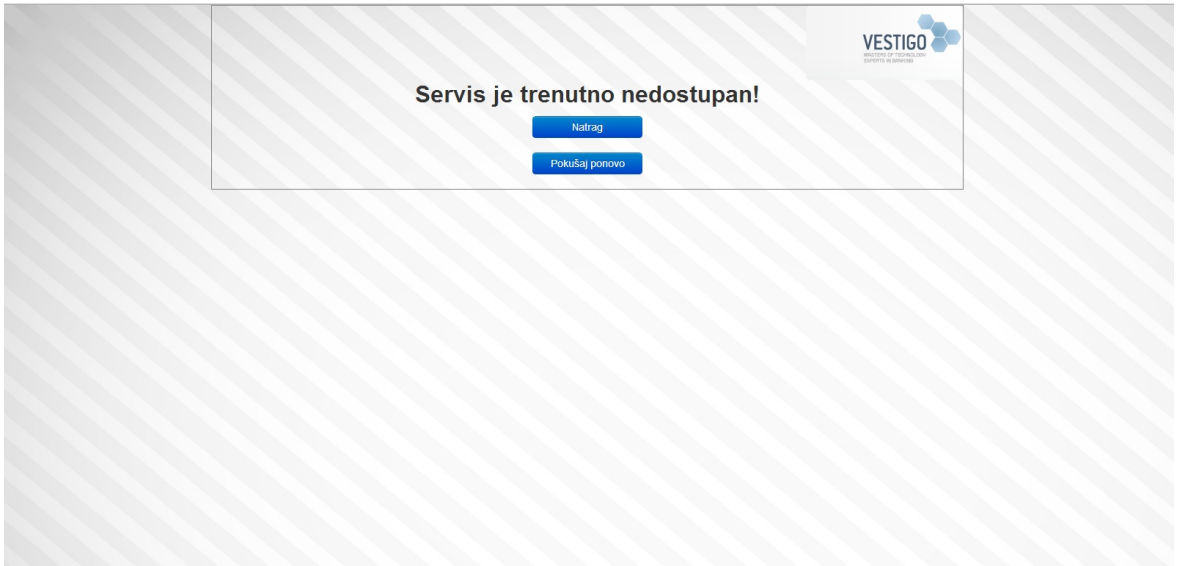
Definira `HttpInvokerProxyFactoryBean` klasu koja prima za parametar *Interface* servisa. Tako u “HomeControlleru” pomoću “@autowire” anotacije poveže *interface* sa Beanom “`vestigoServiceClient`” te pozove udaljeni servis kojim dohvati podatke iz baze.

3. Upute za korištenje

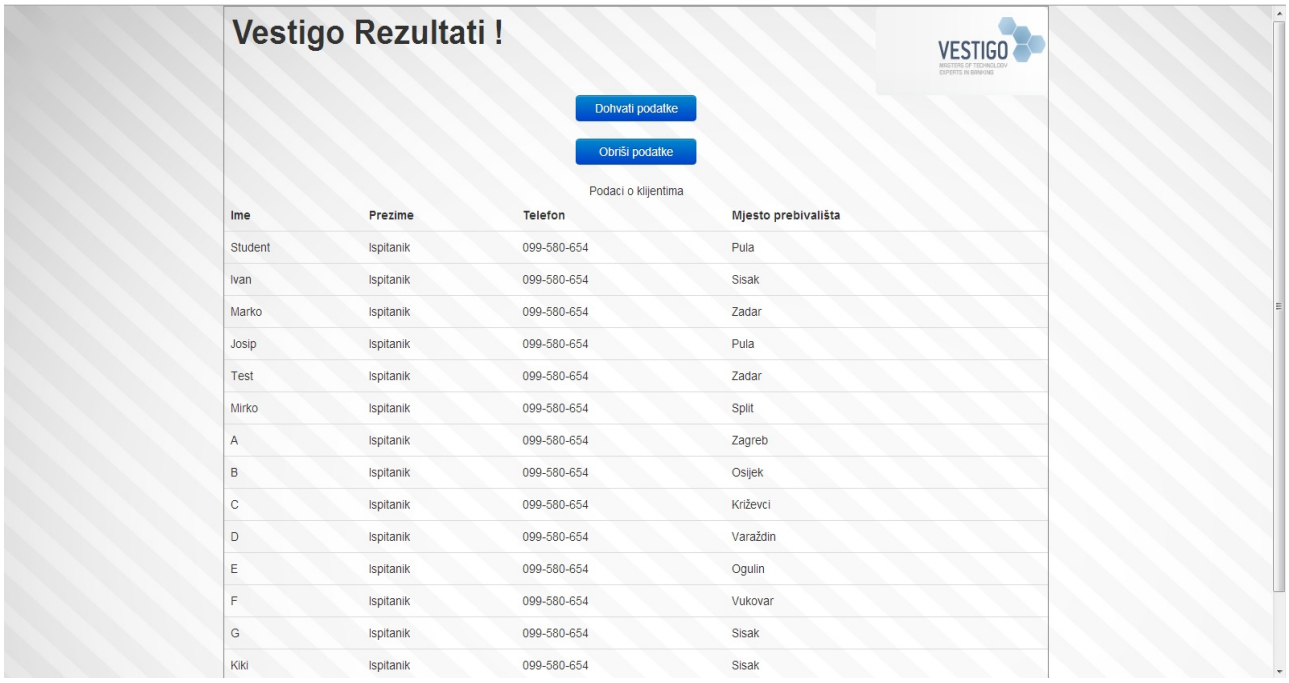
Ova jednostavna web aplikacija ima samo jednu funkciju, a to je dohvat podataka sa remote servisa. To se ostvaruje pritiskom na dugme “Dohvati podatke”.



Ukoliko je servis neaktivan ili je došlo do pogrešake u radu, korisniku se prikazuje informativna poruka.



U suprotnome, dohvačeni podaci se prikazuju korisniku.



4. Tehnologije korištene u projektu

4.1. Spring

Open source framework za Java platformu. Razvijen je s ciljem pojednostavljivanja razvoja Java aplikacija (ne nužno samo J2EE aplikacija) temeljenih na JavaBeansima.
U ovome projektu korištene su neke od njegovih usluga:

- *Inversion of control* - definiranje zavisnosti objekata u konfiguracijkim datotekama
- *Aspect oriented programming* - sam Spring koristi AOP za transaction management, security, remote access, i JMX.
- *MVC* - razdvajanje koda na tri dijela: *model*, view i *controller*
- *Data access* - pruža potporu za sve poznate frameworke u Javi: JDBC, iBatis/MyBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB, and Apache Cayenne
- *Transaction managment*
- Remote access framework

4.2. Hibernate

Omogućava „mapiranje“ POJO objekata na retke u bazi podataka, čime je izbjegnuto “ručno“” popunjavanje objekata prilikom dohvaćanja iz baze ili popunjavanje upita kod spremanja podataka u bazu. Neke od Hibernate anotacija :

- @Entity – služi za definiranje perzistentne klase, definira se na razini klase
- @Table – služi za definiranje tablice za koju se perzistentna klasa veže, definira se na razini klase (parametar „name“ definira naziv tablice u bazi)
- @Id – služi za definiranje property-a kao identifikatora klase
- @Column – služi za definiranje naziva kolone u bazi podataka koja se povezuje s dotičnim property-em
- @JoinColumn, @OneToOne, @ManyToOne, @ManyToMany..., služe za definiranje veza između tablica

Lazy loading omogućava dohvaćanje svih podataka u grafu objekata tek kad zatrebaju, korištenjem tvz. “proxy“ objekata (ponašaju se kao pravi objekti, ali nemaju prave vrijednosti)

4.3. H2

Relacijski DBMS napisan u Javi. U ovome projektu koristi se u demonstracijske svrhe.

Dostupan kao *Open Source* softer. Može biti ugrađena u Java aplikaciju ili pokrenuta u klijent-server modu.

4.4. Apache Maven

Maven je alat za automatizaciju *buildanja* potrebnih projekata. Jedna od glavnih stavki je jednostavno dijeljenje resursa kroz više projekata. *Project Object Model* ili POM je glavna cjelina rada u Mavenu. POM je XML datoteka koja sadrži podatke o projektu i detaljima konfiguracije Mavena.