



## CS230 - DISTRIBUIRANI SISTEMI

Distribuirani servisi – RESTful i  
SOAP veb servisi sa JAX - RS

Lekcija 13

PRIRUČNIK ZA STUDENTE

# CS230 - DISTRIBUIRANI SISTEMI

## Lekcija 13

### *DISTRIBUIRANI SERVISI - RESTFUL I SOAP VEB SERVISI SA JAX - RS*

- ✓ Distribuirani servisi – RESTful i SOAP veb servisi sa JAX - RS
- ✓ Poglavlje 1: Generisanje RESTful servisa iz baze podataka
- ✓ Poglavlje 2: Testiranje RESTful veb servisa
- ✓ Poglavlje 3: Generisanje RESTful Java klijent koda
- ✓ Poglavlje 4: Generisanje JavaSript RESTful klijenta
- ✓ Poglavlje 5: Uvod u SOAP servise
- ✓ Poglavlje 6: Kreiranje jednostavnog veb servisa
- ✓ Poglavlje 7: Testiranje kreiranog veb servisa
- ✓ Poglavlje 8: Razvoj klijenta veb servisa
- ✓ Poglavlje 9: Implementiranje novog veb servisa kao EJB
- ✓ Poglavlje 10: Pokazni primeri - Distribuirani veb servisi
- ✓ Poglavlje 11: Individualna vežba 14 (135 min)
- ✓ Poglavlje 12: Domaći zadatak 13 (180 min)
- ✓ 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

*Veb servisi omogućavaju kreiranje softverskih funkcionalnosti kojima je moguće pristupiti putem mreže na način koji ne zavisi od konkretnog programskog jezika ili platforme.*

Veb servisi omogućavaju kreiranje softverskih funkcionalnosti kojima je moguće pristupiti putem mreže na način koji ne zavisi od konkretnog programskog jezika ili platforme.

Postoje dva različita pristupa, koji se i najčešće koriste, kada je u pitanju razvoj veb servisa:

- prvi pristup je poznat kao *Simple Object Access Protocol* (SOAP);
- drugi pristupe je označen kao *Representational State Transfer* (REST) protocol.

Savremena razvojna okruženja, pa tako i *NetBeans IDE* na kome je fokus u ovim materijalima, podržavaju razvoj veb servisa primenom oba navedena pristupa. U daljem radu, od posebnog značaja će biti analiza i diskusija u vezi sa razvojem veb servisa primenom oba pristupa.

Kada se koristi *SOAP protokol* (SOAP Protocol), operacije veb servisa su definisane u XML dokumentu pod nazivom *Web Services Definition Language* (WSDL). Nakon kreiranja WSDL datoteke, izvodi se implementacija veb servisa primenom pogodnog programskog jezika, kakav je, između ostalih, i programski jezik Java.

### UVOD REST

*Lekcija će se baviti RESTful veb servisima u Java EE platformi.*

REST (*Representational State Transfer*) predstavlja arhitekturni stil u kojem su veb servisi reprezentovani kao resursi i mogu biti identifikovani preko jedinstvenih identifikatora resursa (URI - Uniform Resource Identifiers).

Veb servisi koji su razvijeni primenom REST stila poznati su kao *RESTful veb servisi*. Sa uvođenjem *Java EE 6* platforme predstavljena je i nova Java podrška za *RESTful veb servise* preko novog *Java API za RESTful veb servise* (*Java API for RESTful Web Services*) pod nazivom JAX-RS. U početku JAX-RS je bio dostupan kao samostalni API, da bi nakon izvesnog vremena postao integralni deo *Java EE 6 specifikacije*.

Jedan od opštih načina primene *RESTful veb servisa* jeste njihovo ponašanje kao **frontend** -a za bazu podataka. To praktično znači, klijent *RESTful veb servisa* koristi *RESTful veb servis* za izvođenje **CRUD** (*create, read, update* i *delete*) operacija nad bazom podataka. Posebno značajnu pomoć implementaciji RESTful web servisa, u savremenim Java veb aplikacijama, obezbeđuju savremena razvojna okruženja. Pomoću razvojnog okruženja *NetBeans IDE* obezbeđena je podrška za kreiranje *RESTful veb servisa* u nekoliko jednostavnih koraka.

U navedenom svetu, lekcija će se posebno fokusirati na nekoliko pažljivo odabralih tema:

- Generisanje RESTful veb servisa iz postojeće baze podataka;
- Testiranje RESTful veb servisa pomoću alata koje obezbeđuje razvojno okruženje NetBeans IDE;
- Generisanje RESTful Java klijent koda RESTful veb servisa;
- Generisanje RESTful JavaScript klijenata RESTful veb servisa;

Savladavanjem ove lekcije student će biti u stanju da u potpunosti razume i koristi RESTFUL veb servise u JAVA EE aplikacijama.

## UVOD SOAP

*Distribuirani veb servisi dozvoljavaju razvoj funkcionalnosti kojima se pristupa preko mreže.*

Distribuirani veb servisi dozvoljavaju razvoj funkcionalnosti kojima se pristupa preko mreže. Ono što razlikuje veb servise od sličnih tehnologija, poput EJB (Enterprise Java Beans) ili poziva udaljenih metoda (RMI - Remote Method Invocation), jeste to da veb servisi ne zavise od programskog jezika i / ili platforme. Na primer, veb servisu, koji je razvijen pomoću programskog jezika Java, moguće je pristupiti klijent programima koji su kreirani upotrebom nekog drugog programskog ili skript jezika i obrnuto.

Za analizu i demonstraciju problematike SOAP veb servisa primenom JAX - WS, biće pažljivo izabrane teme koje će ova lekcija obrađivati. Posebno će biti insistirano na konkretnim primerima koji će predstavljati podršku analizi i diskusiji koja će biti vođena u lekciji.

U navedenom svetlu, lekcija će poseban akcenat staviti na sledeće teme:

- Uvod u veb servise;
- Kreiranje jednostavnog veb servisa;
- Testiranje i razvoj klijenta za veb servis;
- Enterprise Java zrno kao veb servis;
- Implementiranje novog veb servisa kao EJB;
- Kreiranje veb servisa iz postojećeg WSDL.

Savladavanjem ove lekcije student će u potpunosti moći da razume, kreira i koristi SOAP veb servise u JAVA EE aplikacijama.

## ▼ Poglavlje 1

# Generisanje RESTful servisa iz baze podataka

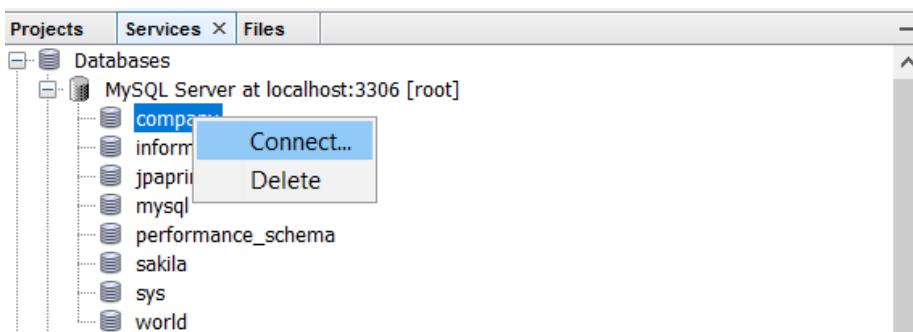
## NETBEANS IDE ČAROBNIJAK ZA GENERISANJE RESTFUL SERVISA

*Razvojno okruženje NetBeans IDE automatizuje generisanje koda RESTful servisa.*

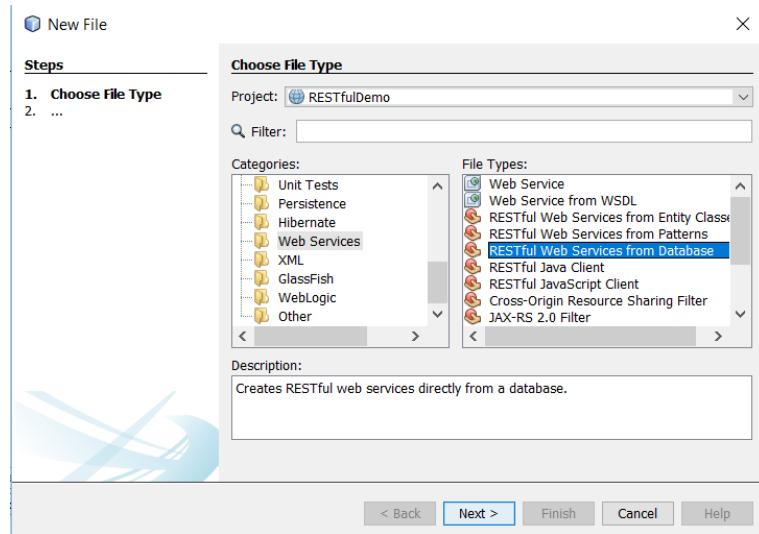
Kao što je napomenuto u uvodnom izlaganju, savremena razvojna okruženja mogu u značajnoj meri da olakšaju izradu Java EE veb aplikacija koje koriste RESTful servise. U narednom izlaganju je cilj da se pokaže kao razvojno okruženje NetBeans IDE automatizuje generisanje RESTful servisa iz postojeće baze podataka. Za početak je neophodno kreirati novi projekat iz kategorije Java Web, tipa Web Application.

Dakle, i u ovom delu lekcije će biti nastavljena praksa da svaka analiza i odgovarajuće izlaganje, budu praćeni pažljivo izabranim primerom. Projekat može da dobije naziv RESTfulDemo. Nakon kreiranja projekta, izbora aplikativnog servera GlassFish i odgovarajućih okvira (npr. JSF framework) moguće je pristupiti generisanju njegovih datoteka. Ovde će, za kreiranje RESTful servisa, biti iskorišćena postojeća MySQL baza podataka pod nazivom company. Navedeno je prikazano Slikom 1.

Za kreiranje RESTful servisa iz postojeće baze podataka neophodno je, koristeći razvojno okruženje NetBeans IDE, izvesti nekoliko jednostavnih koraka. Za početak, za kreirani projekat, neophodno je izabrati opcije File | New. Nakon toga otvara se prozor u okviru kojeg je neophodno izabrati iz kategorije Web Services, tip datoteke RESTful Web Services From Database. Navedeno je prikazano Slikom 2.



Slika 1.1 Povezivanje na postojeću bazu podataka [izvor: autor]

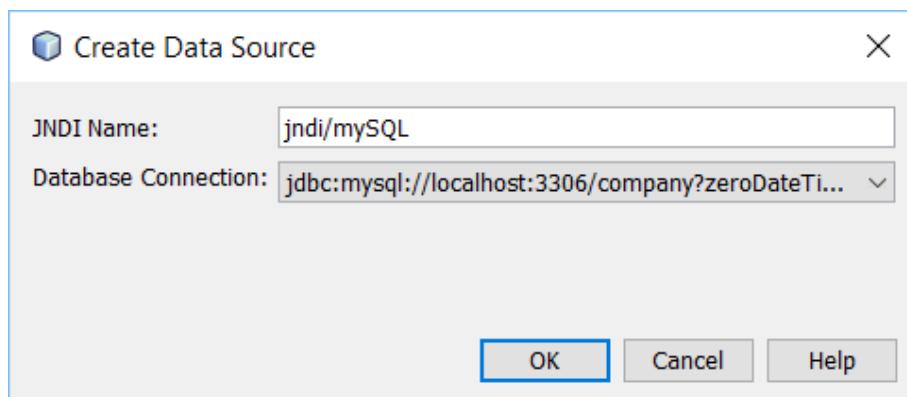


Slika 1.2 Izbor opcije RESTful Web Services From Database [izvor: autor]

## IZBOR TABELA IZ BAZE PODATAKA U RAZVOJNOM OKRUŽENJU

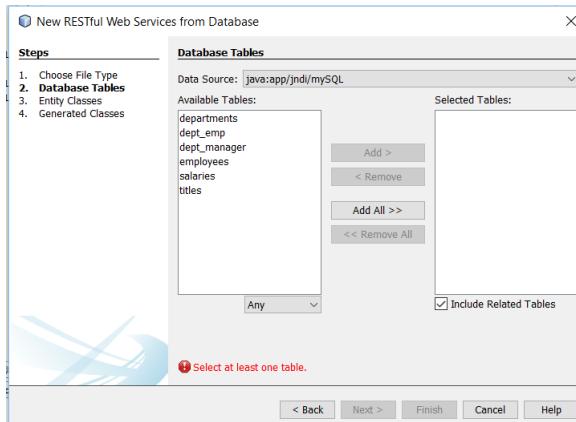
*U nastavku je neophodno izabrati tabele nad kojima će biti kreirane klase RESTful servisa.*

Prethodnom slikom je demonstrirano kako započinje proces kreiranja datoteka tipa *RESTful Web Services From Database*. Klikom na dugme *Next*, ovog prozora, prelazi se na novi prozor *NetBeans IDE* čarobnjaka, u kojem je neophodno izabrati tabele iz baze podataka *company*. U međuvremenu, biće zatraženo da kreiramo izvor podataka, zajedno sa odgovarajućim JNDI nazivom, što može biti demonstrirano sledećom slikom.

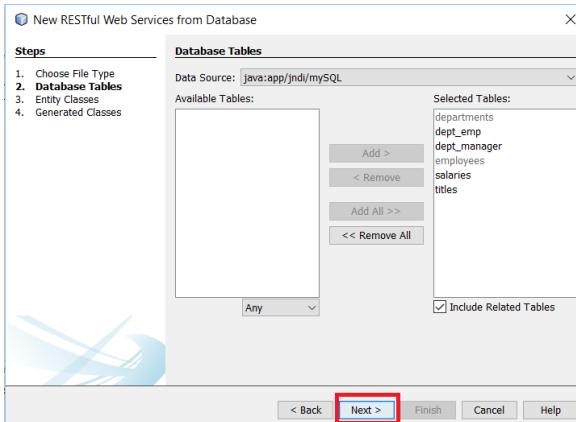


Slika 1.3 Podešavanje baze podataka kao izvora podataka [izvor: autor]

U nastavnu, u navedenom prozoru se otvara lista svih dostupnih tabela baze podataka *company*. Navedeno je prikazano Slikom 4. Klikom na dugme *Add all*, sve tabele bivaju izabrane za generisanje *RESTful* servisa. Lista izabranih tabela za generisanje klasa *RESTful* servisa je prikazana Slikom 5.



Slika 1.4 Lista tabela za RESTful servise [izvor: autor]

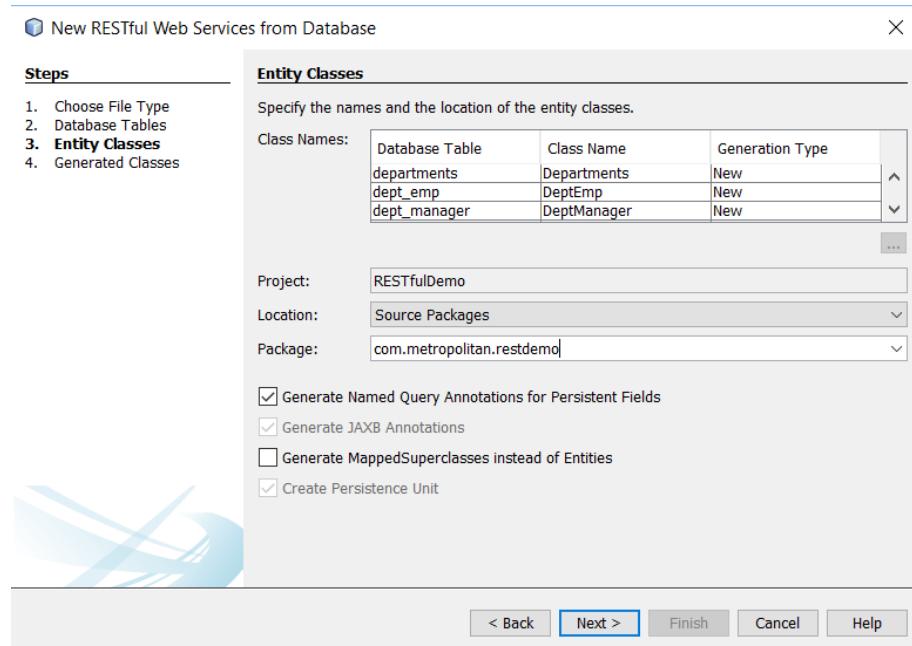


Slika 1.5 Izabrane tabele za RESTful servise [izvor: autor]

## PODEŠAVANJE KLASA RESTFUL SERVISA

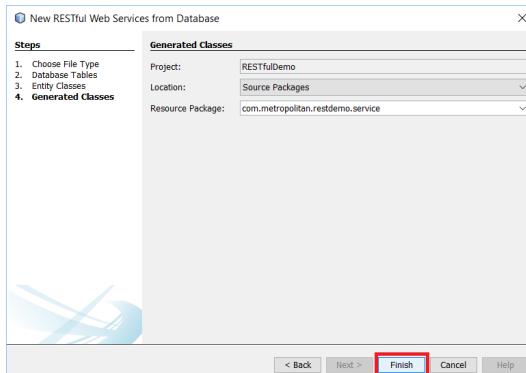
*U NetBeans IDE okruženju je neophodno dodatno podešiti klase koje su u fazi kreiranja.*

Klikom na dugme *Next*, pogledati prethodnu sliku, otvara se novi prozor u kojem je neophodno izabrati paket kojem će pripadati klase koje se generišu. Polje za generisanje *anotacija imenovanih upita* je čekirano po osnovnim podešavanjima i tako bi trebalo i da ostane. Navedeno izlaganje je podržano sledećom slikom.



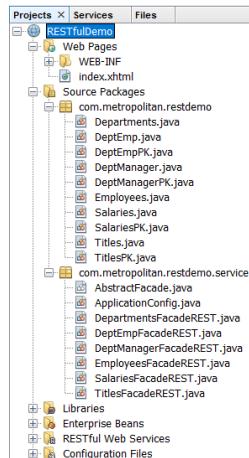
Slika 1.6 Podešavanje novih RESTful servisa [izvor: autor]

Klikom na dugme *Next*, otvara se prozor u kojem se potvrđuju izabrana podešavanja i završava generisanje klasa *RESTful* servisa (Slika 7).



Slika 1.7 Potvrđivanje podešavanja, paketa resursa i kreiranja RESTful servisa [izvor: autor]

Klikom na *Finish*, generisane klase zauzimaju svoje mesto u projektu.



Slika 1.8 Trenutna struktura projekta RESTfulDemo [izvor: autor]

## JPA KLASE ENTITETA

*Generisani kod je neophodno analizirati i diskutovati.*

Kao što je moguće primetiti iz prethodnog izlaganja, generisanje izvesnih Java klasa je završeno i neophodno je izvršiti analizu i diskusiju generisanog koda. Ovde će biti izabранo par reprezentativnih datoteka, dok će ostale biti priložene u sekciji **Vežbe** gde će celokupan primer biti u fokusu.

Posebno je moguće primetiti da je *NetBeans IDE* čarobnjak za tabelu baze podataka generisao odgovarajuću *JPA entitetsku klasu*. Takođe, za svaki *JPA entitet* generisana je fasadna (*Facade*) klasa zajedno sa apstraktnom klasom *AbstractFacade.java*. Generisani kod se oslanja na fasadni dizajn šablon (*Facade design pattern*) koji podrazumeva da su fasadne klase omotači odgovarajućih JPA klasa entiteta.

Sledi kod JPA entitetske klase *Employees.java*. *JPA klase entiteta* su detaljno obrazložene u nekom od prethodnih izlaganja tako da ovde neće biti zadržavanja u diskusiji.

```
package com.metropolitan.restdemo;

import java.io.Serializable;
import java.util.Collection;
import java.util.Date;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;
```

```
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

/**
 *
 * @author Vladimir Milicevic
 */
@Entity
@Table(name = "employees")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Employees.findAll", query = "SELECT e FROM Employees e")
    , @NamedQuery(name = "Employees.findByEmpNo", query = "SELECT e FROM Employees e WHERE e.empNo = :empNo")
    , @NamedQuery(name = "Employees.findByBirthDate", query = "SELECT e FROM Employees e WHERE e.birthDate = :birthDate")
    , @NamedQuery(name = "Employees.findByFirstName", query = "SELECT e FROM Employees e WHERE e.firstName = :firstName")
    , @NamedQuery(name = "Employees.findByLastName", query = "SELECT e FROM Employees e WHERE e.lastName = :lastName")
    , @NamedQuery(name = "Employees.findByGender", query = "SELECT e FROM Employees e WHERE e.gender = :gender")
    , @NamedQuery(name = "Employees.findByHireDate", query = "SELECT e FROM Employees e WHERE e.hireDate = :hireDate")})
public class Employees implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "emp_no")
    private Integer empNo;
    @Basic(optional = false)
    @NotNull
    @Column(name = "birth_date")
    @Temporal(TemporalType.DATE)
    private Date birthDate;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 14)
    @Column(name = "first_name")
    private String firstName;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 16)
    @Column(name = "last_name")
    private String lastName;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 2)
    @Column(name = "gender")
    private String gender;
```

```
@Basic(optional = false)
@NotNull
@Column(name = "hire_date")
@Temporal(TemporalType.DATE)
private Date hireDate;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "employees")
private Collection<Salaries> salariesCollection;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "employees")
private Collection<DeptEmp> deptEmpCollection;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "employees")
private Collection<DeptManager> deptManagerCollection;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "employees")
private Collection<Titles> titlesCollection;

public Employees() {
}

public Employees(Integer empNo) {
    this.empNo = empNo;
}

public Employees(Integer empNo, Date birthDate, String firstName, String
lastName, String gender, Date hireDate) {
    this.empNo = empNo;
    this.birthDate = birthDate;
    this.firstName = firstName;
    this.lastName = lastName;
    this.gender = gender;
    this.hireDate = hireDate;
}

public Integer getEmpNo() {
    return empNo;
}

public void setEmpNo(Integer empNo) {
    this.empNo = empNo;
}

public Date getBirthDate() {
    return birthDate;
}

public void setBirthDate(Date birthDate) {
    this.birthDate = birthDate;
}

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 getGender() {
    return gender;
}

public void setGender(String gender) {
    this.gender = gender;
}

public Date getHireDate() {
    return hireDate;
}

public void setHireDate(Date hireDate) {
    this.hireDate = hireDate;
}

@XmlTransient
public Collection<Salaries> getSalariesCollection() {
    return salariesCollection;
}

public void setSalariesCollection(Collection<Salaries> salariesCollection) {
    this.salariesCollection = salariesCollection;
}

@XmlTransient
public Collection<DeptEmp> getDeptEmpCollection() {
    return deptEmpCollection;
}

public void setDeptEmpCollection(Collection<DeptEmp> deptEmpCollection) {
    this.deptEmpCollection = deptEmpCollection;
}

@XmlTransient
public Collection<DeptManager> getDeptManagerCollection() {
    return deptManagerCollection;
}

public void setDeptManagerCollection(Collection<DeptManager>
deptManagerCollection) {
    this.deptManagerCollection = deptManagerCollection;
}
```

```
@XmlTransient
public Collection<Titles> getTitlesCollection() {
    return titlesCollection;
}

public void setTitlesCollection(Collection<Titles> titlesCollection) {
    this.titlesCollection = titlesCollection;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (empNo != null ? empNo.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not
set
    if (!(object instanceof Employees)) {
        return false;
    }
    Employees other = (Employees) object;
    if ((this.empNo == null && other.empNo != null) || (this.empNo != null &&
!this.empNo.equals(other.empNo))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "com.metropolitan.restdemo.Employees[ empNo=" + empNo + " ]";
}
}
```

## ABSTRACTFAÇADE KLASA

*Nakon JPA entitetskih klasa neophodno je pozabaviti se roditeljskom klasom RESTful servisa.*

Ono što je bilo moguće primetiti iz prethodnog izlaganja jeste da je za svaki *JPA entitet* generisana je fasadna (*Facade*) klasa zajedno sa apstraktnom klasom *AbstractFacade.java*. Fasadna klasa je omotač odgovarajuće klase JPA entiteta, dok svaka fasadna klasa nasleđuje apstraktnu klasu *AbstractFacade.java*. Sledi listing navedene roditeljske klase fasadnih klasa:

```
package com.metropolitan.restdemo.service;

import java.util.List;
import javax.persistence.EntityManager;

/**
 *
 * @author Vladimir Milicevic
 */
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
```

```
        javax.persistence.criteria.CriteriaQuery cq =  
getEntityManager().getCriteriaBuilder().createQuery();  
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);  
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));  
        javax.persistence.Query q = getEntityManager().createQuery(cq);  
        return ((Long) q.getSingleResult()).intValue();  
    }  
  
}
```

Iz priloženog koda je moguće primetiti da klasa *AbstractFacade.java* poseduje varijablu *entityClass* koja se podešava po određenom tipu njenih klasa potomaka primenom generika. Takođe, ova klasa poseduje metode **create**, **edit**, **remove**, **find** i **count** za kreiranje, ažuriranje, uklanjanje, pronalaženje i prebrojavanje entiteta, respektivno. Telo ovih metoda predstavlja dobro poznati *JPA* kod i oko njega u ovom delu lekcije neće biti zadržavanja.

## RESTFUL KLASE

*Konačno, sledi analiza koda RESTful klase.*

Kreirane fasadne kase se angažuju kao RESTful servisi u Java EE veb aplikacijama koje koriste RESTful servise.

Kao što je bilo moguće primetiti, NetBeans IDE čarobnjak je kreirao *fasadu* ili *omotač* za svaki JPA entitet podignut nad odgovarajućom tabelom baze podataka *company*. U ovom delu izlaganja je od značaja fasadna klasa JPA entiteta *employees.java*, podignutog nad tabelom *employees* baze podataka *company*. Sledi listing datoteke *EmployeesFacadeREST.java*.

```
package com.metropolitan.restdemo.service;  
  
import com.metropolitan.restdemo.Employees;  
import java.util.List;  
import javax.ejb.Stateless;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import javax.ws.rs.Consumes;  
import javax.ws.rs.DELETE;  
import javax.ws.rs.GET;  
import javax.ws.rs.POST;  
import javax.ws.rs.PUT;  
import javax.ws.rs.Path;  
import javax.ws.rs.PathParam;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;  
  
/**  
 *  
 * @author Vladimir Milicevic
```

```
/*
*@Stateless
@Path("com.metropolitan.restdemo.employees")
public class EmployeesFacadeREST extends AbstractFacade<Employees> {

    @PersistenceContext(unitName = "RESTfulDemoPU")
    private EntityManager em;

    public EmployeesFacadeREST() {
        super(Employees.class);
    }

    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void create(Employees entity) {
        super.create(entity);
    }

    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Integer id, Employees entity) {
        super.edit(entity);
    }

    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") Integer id) {
        super.remove(super.find(id));
    }

    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Employees find(@PathParam("id") Integer id) {
        return super.find(id);
    }

    @GET
    @Override
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Employees> findAll() {
        return super.findAll();
    }

    @GET
    @Path("{from}/{to}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Employees> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
        return super.findRange(new int[]{from, to});
    }
}
```

```
@GET  
@Path("count")  
@Produces(MediaType.TEXT_PLAIN)  
public String countREST() {  
    return String.valueOf(super.count());  
}  
  
@Override  
protected EntityManager getEntityManager() {  
    return em;  
}  
}
```

Ono što je moguće uočiti na prvom mestu jeste da je klasa *EmployeesFacadeREST.java* obeležena anotacijom **@Stateless**. To znači da klasa predstavlja zrno sesije bez stanja.

Anotacija **@Path** je upotrebljena sa ciljem identifikovanja **URI** (**Uniform Resource Identifier**) identifikatora za koji će kreirana klasa slati zahteve. Kao što je moguće primetiti, iz priloženog listinga, nekoliko metoda je obeleženo anotacijama **@POST**, **@PUT**, **@DELETE** i **@GET**. Ove metode će automatski biti pozivane kada veb servis odgovori na odgovarajući HTTP zahtev.

Takođe, nekoliko metoda je obeleženo anotacijom **@Path**. To znači da neke od metoda zahtevaju parametre. Na primer, ukoliko je neophodno izvršiti brisanje nekog unosa iz tabele *employees*, neophodno je proslediti primarni ključ odgovarajuće vrste kao parametar metode **remove()**. Format odgovarajuće vrednosti, obuhvaćene anotacijom **@Path**, odgovara obliku "**{varName}**" pri čemu vrednost je između velikih zagrada označena kao **parametar putanje** (**path parameter**). Na kraju je moguće primetiti da su za ovakve metode odgovarajući parametri obeleženi anotacijom **@PathParam**.

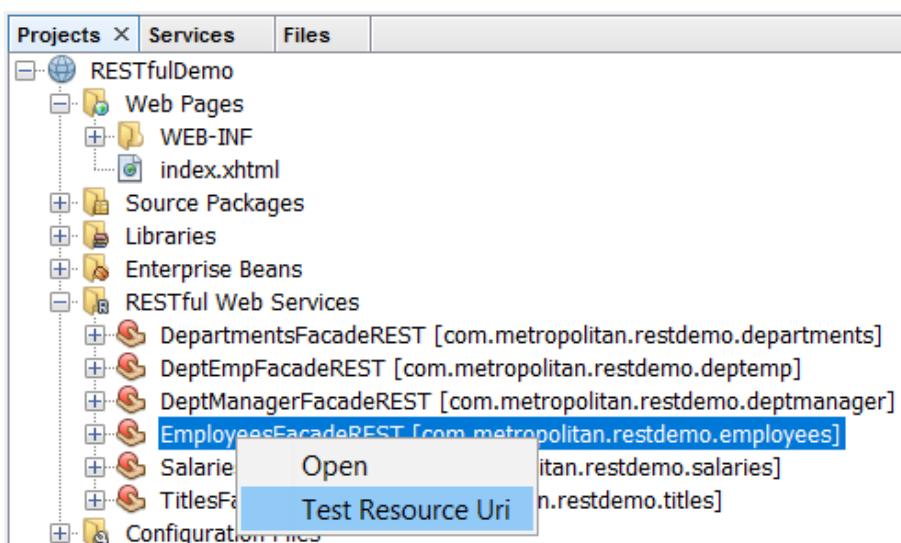
## ▼ Poglavlje 2

# Testiranje RESTful veb servisa

## TEST RESOURCE URI

*Neophodno je pokazati NetBeans IDE mehanizme za testiranje RESTful veb servisa.*

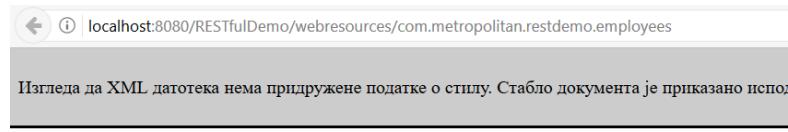
Nakon što je razvijen veb projekat primera, moguće je izvršiti proveru da li su i *RESTful* veb servisi kreirani na adekvatan način. Procedura provere započinje tako što se izvrši desni klik mišem na odgovarajući *RESTful veb servis*, u folderu *RESTful Web Services*, a zatim izabere opciju *Test Resource Uri*. Klasa *RESTful veb servisa* nad kojom će biti obavljeno *testiranje* je dobro poznata klasa *EmployeesFacadeREST.java*. Opisana procedura je prikazana sledećom slikom.



Slika 2.1 Izbor opcije Test Resource Uri za RESTful veb servis [izvor: autor]

Pre navedene radnje neophodno je izvršiti desni klik na koren projekta i izabrati opciju deploy, ukoliko projekat nije angažovan.

Prikazana akcija će pozvati metodu *findAll()* u posmatranom servisu iz razloga što ona predstavlja jedinu metodu koja ne zahteva nijedan parametar. Tada dolazi do generisanja XML odgovora koji će automatski biti prikazan u veb pregledaču na način prikazan sledećom slikom. XML prikazuje trenutni sadržaj odgovarajuće tabele.



The screenshot shows a browser window with the URL `localhost:8080/RESTfulDemo/webresources/com.metropolitan.restdemo.employees`. A message in the center of the page reads: "Изгледа да XML датотека нема придружене податке о стилу. Стабло документа је приказано испод." Below this, the XML document structure is displayed:

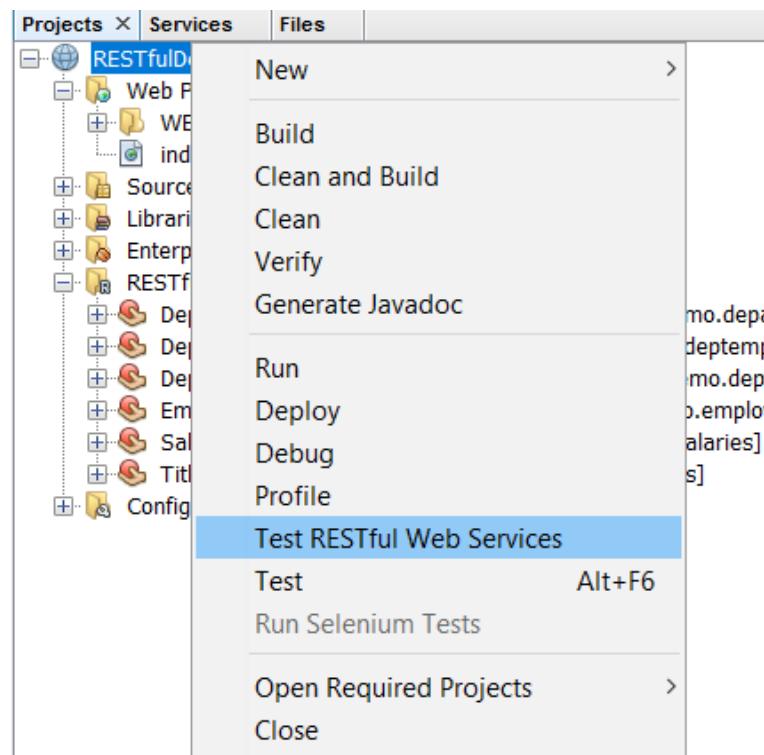
```
- <employees>
  - <employees>
    <birthDate>1974-09-20T00:00:00+01:00</birthDate>
    <empNo>1</empNo>
    <firstName>Vladimir</firstName>
    <gender>M</gender>
    <hireDate>2020-09-20T00:00:00+02:00</hireDate>
    <lastName>Milicevic</lastName>
  </employees>
  - <employees>
    <birthDate>1980-07-21T00:00:00+02:00</birthDate>
    <empNo>2</empNo>
    <firstName>Petar</firstName>
    <gender>M</gender>
    <hireDate>2012-10-12T00:00:00+02:00</hireDate>
    <lastName>Nikolic</lastName>
  </employees>
  - <employees>
    <birthDate>1978-05-18T00:00:00+01:00</birthDate>
    <empNo>3</empNo>
    <firstName>Jovana</firstName>
    <gender>M</gender>
    <hireDate>2013-11-11T00:00:00+01:00</hireDate>
    <lastName>Markovic</lastName>
  </employees>
</employees>
```

Slika 2.2 Generisani XML kao odgovor na metodu findAll() [izvor: autor]

## TEST RESTFUL WEB SERVICES

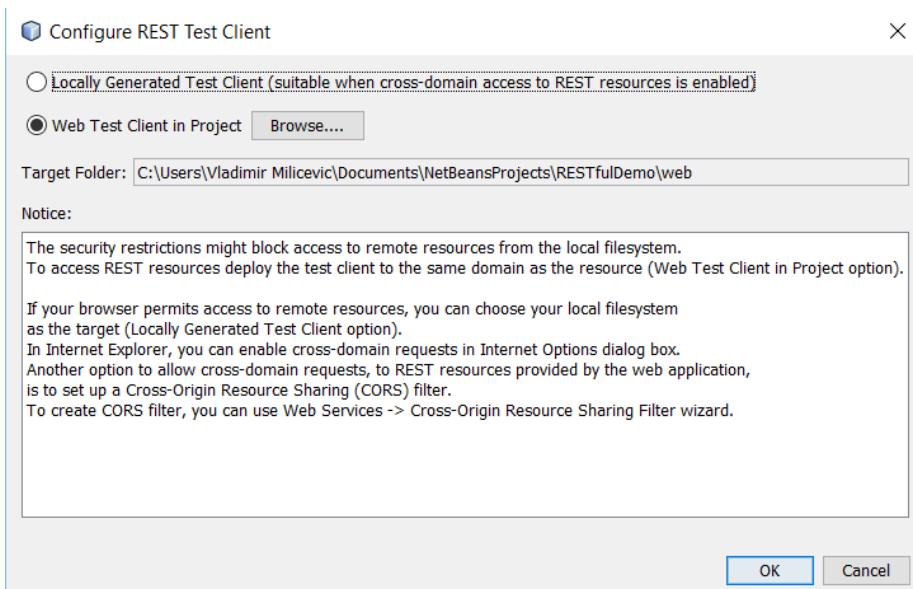
*Sledeće testiranje započinje izborom opcije Test RESTful Web Services.*

Takođe, razvojno okruženje *NetBeans IDE* dozvoljava da se na veoma jednostavan način testiraju i druge metode aktuelnog veb servisa. Neophodno je izvršiti desni klik na koren projekta i izabratи opciju **Test RESTful Web Services**. Navedeno je moguće ilustrovati sledećom slikom.



Slika 2.3 Izbor opcije za testiranje Test RESTful Web Services [izvor: autor]

Nakon obavljenе akcije, prikazane prethodnom slikom, pojavljuje se sledeći prozor.



Slika 2.4 Podešavanje RESTful test klijenta [izvor: autor]

U velikoj većini slučajeva dovoljno je prihvati podrazumevani web test klijent (**Web Test Client**) iz opcije **Project** budući da je podržan od strane većine web pregledača i operativnih sistema.

## STRANICA TEST RESTFUL WEB SERVICES

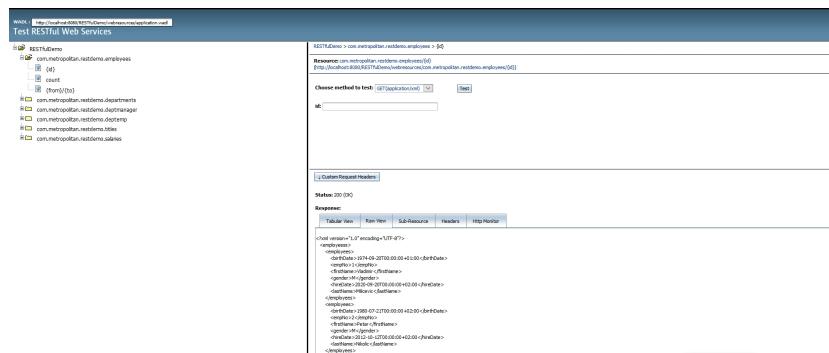
*Nakon izbora web test klijenta otvorice se u web pregledaču stranica Test RESTful Web Services.*

Nakon izbora web test klijenta otvorice se u web pregledaču stranica *Test RESTful Web Services*. Navedena stranica prikazana je sledećom slikom.



Slika 2.5 Stranica Test RESTful Web Services [izvor: autor]

Proširenjem bilo kojeg čvora *RESTful* servisa, sa leve strane prethodne slike, i izborom *GET(application/xml)* ili *GET(application/json)*, u padajućoj listi sa desne strane koja je obeležena labelom *Choose method to test*, *HTTP GET* zahtev se šalje *RESTful web servisu* i vraća se *XML* ili *JSON* odgovor, respektivno.



Slika 2.6 Vraćanje XML odgovora [izvor: autor]

Ukoliko se iz prethodno navedenog menija izabere *PUT* metoda, moguće je proslediti *XML* ili *JSON* formatirane podatke. Na ovaj način je moguće dodati jedan zapis u bazu podataka. Sledećom slikom je demonstrirano testiranje dodavanja zapisa u bazu podataka (klik na Test izvršava *PUT* metodu).

Slika 2.7 Test dodavanja novog zapisa [izvor: autor]

Sada je neophodno ponovo izabrati *GET(application/xml)* za proveru korektnosti dodavanja novog zapisa *PUT(application/xml)* metodom.

```

<status>200 (OK)
<headers>
<header>Content-Type: application/xml</header>
<header>Date: Mon, 20 Sep 2021 10:45:45 GMT</header>
<header>Content-Length: 313</header>
<body><employees>
<employee>
<empNo>1</empNo>
<firstName>Vladimir</firstName>
<lastName>Milicevic</lastName>
<gender>M</gender>
<hireDate>2020-09-20T00:00:00+02:00</hireDate>
<empNo>2</empNo>
<firstName>Petar</firstName>
<lastName>Nikolic</lastName>
<gender>M</gender>
<hireDate>2012-10-12T00:00:00+02:00</hireDate>
<empNo>3</empNo>
<firstName>Jovana</firstName>
<lastName>Markovic</lastName>
<gender>M</gender>
<hireDate>2013-11-11T00:00:00+01:00</hireDate>
<empNo>4</empNo>
<firstName>Marija</firstName>
<lastName>Jovanovic</lastName>
<gender>M</gender>
<hireDate>2021-09-20T00:00:00+02:00</hireDate>
</employees>
</body>

```

Slika 2.8 Test nakon unetog novog zapisa [izvor: autor]

## KLASA APPLICATIONCONFIG.JAVA

*Posebnu pažnju je nophodno obratiti na klasu ApplicationConfig.java.*

Poslednji test je pokazao da su *RESTful* web servisi konkretnog primera razvijeni na pravi način. To može biti provereno i u *MySQL Workbench* okruženju, izvršavanjem SQL upita:

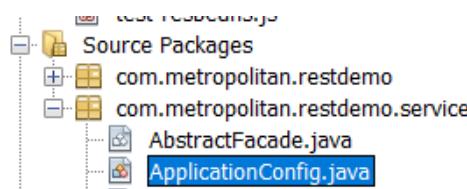
```
select * from employees
```

za proveru da li je poslednji zapis zaista se našao u bazi podataka. Da je sve urađeno na pravi način prikazano sledećom slikom.

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	1	1974-09-20	Vladimir	Milicevic	M	2020-09-20
	2	1980-07-21	Petar	Nikolic	M	2012-10-12
	3	1978-05-18	Jovana	Markovic	M	2013-11-11
	4	1984-06-20	Marija	Jovanovic	M	2021-09-20
*	NULL	NULL	NULL	NULL	NULL	NULL

Slika 2.9 Dodatna provera u MySQL Workbench okruženju [izvor: autor]

Pošto je sve urađeno kako treba, u sledećem koraku je neophodno pristupiti razvoju klijent aplikacije koja koristi *RESTful* servise. Pre toga je neophodno pregledati generisanu klasu *ApplicationConfig.java*. Klasu je tokom prikazanih podešavanja generisalo razvojno okruženje *NetBeans IDE*. Sledi slika koja prikazuje položaj ove klase u hijerarhiji projekta, a odmah nakon slike sledi listing koda ove klase.



Slika 2.10 Položaj klase ApplicationConfig.java u hijerarhiji projekta [izvor: autor]

```
package com.metropolitan.restdemo.service;

import java.util.Set;
import javax.ws.rs.core.Application;

/**
 *
 * @author Vladimir Milicevic
 */
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new java.util.HashSet<?>();
        addRestResourceClasses(resources);
        return resources;
    }

    /**
     * Do not modify addRestResourceClasses() method.
     * It is automatically populated with
     * all resources defined in the project.
     * If required, comment out calling this method in getClasses().
     */
    private void addRestResourceClasses(Set<Class<?>> resources) {

        resources.add(com.metropolitan.restdemo.service.DepartmentsFacadeREST.class);
        resources.add(com.metropolitan.restdemo.service.DeptEmpFacadeREST.class);

        resources.add(com.metropolitan.restdemo.service.DeptManagerFacadeREST.class);
        resources.add(com.metropolitan.restdemo.service.EmployeesFacadeREST.class);
        resources.add(com.metropolitan.restdemo.service.SalariesFacadeREST.class);
        resources.add(com.metropolitan.restdemo.service.TitlesFacadeREST.class);
    }
}
```

## KLASA APPLICATIONCONFIG.JAVA - DODATNA RAZMATRANJA

*Sledi objašnjenje priloženog koda klase `ApplicationConfig.java`.*

Nakon priloženog listinga klase `ApplicationConfig.java` potrebno je dati dodatna objašnjenja u vezi sa kodom kojim je snabdevena ova klasa. Osnovna namena ove klase je da konfiguriše JAX-RS. Jedini zahtev koji klasa `ApplicationConfig.java` mora da ispuni jeste nasleđivanje osnovne klase `javax.ws.rs.core.Application` i njeno obeležavanje anotacijom `@javax.ws.rs.ApplicationPath("webresources")`. Navedena anotacija se koristi da specificira bazični `URI` svih putanja specificiranih `@Path` anotacijom u konkretnim klasama

*RESTful web servisa*. Po osnovnim podešavanjima, razvojno okruženje *NetBeans IDE* koristi putanju pod nazivom **webresources** za sve *RESTful servise*.

Dalje, razvojno okruženje redefiniše metodu **getClasses()** klase **javax.ww.rs.core.Application** i obezbeđuje vraćanje skupa klasa koje sadrže sve *RESTful servise* u konkretnoj aplikaciji (klase obeležene **@Path** anotacijom). Razvojno okruženje *NetBeans IDE* automatski dodaje sve dostupne *RESTful servise* u metodu **addRestResourceClasses()** koja se poziva u okviru poziva metode **getClasses()**.

Sledi izolovan kod metode **getClasses()** klase *ApplicationConfig.java*.

```
@Override
public Set<Class<?>> getClasses() {
    Set<Class<?>> resources = new java.util.HashSet<?>();
    addRestResourceClasses(resources);
    return resources;
}
```

Takođe, biće priložen i izolovan kod metode **addRestResourceClasses()** koja se poziva unutar priložene metode **getClasses()**.

```
private void addRestResourceClasses(Set<Class<?>> resources) {

    resources.add(com.metropolitan.restdemo.service.DepartmentsFacadeREST.class);
    resources.add(com.metropolitan.restdemo.service.DeptEmpFacadeREST.class);

    resources.add(com.metropolitan.restdemo.service.DeptManagerFacadeREST.class);
    resources.add(com.metropolitan.restdemo.service.EmployeesFacadeREST.class);
    resources.add(com.metropolitan.restdemo.service.SalariesFacadeREST.class);
    resources.add(com.metropolitan.restdemo.service.TitlesFacadeREST.class);
}
```

## ▼ Poglavlje 3

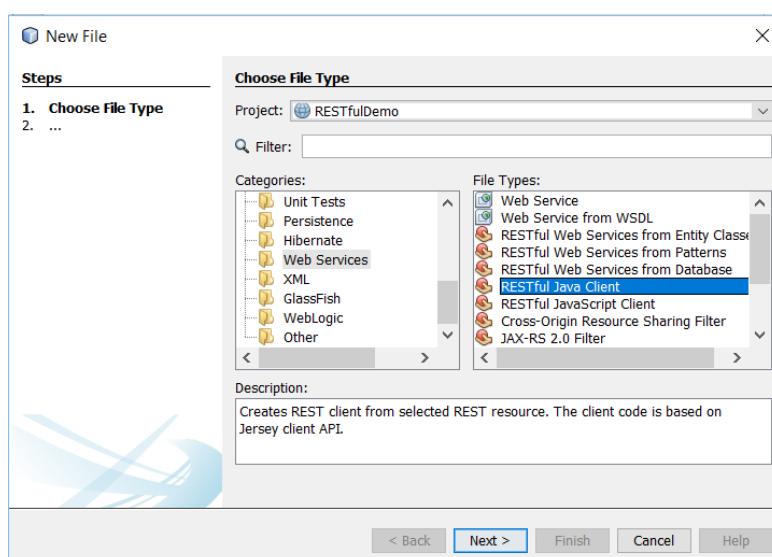
# Generisanje RESTful Java klijent koda

## GENERISANJE DATOTEKE RESTFUL JAVA KLIJENT KODA

*NetBeans IDE obezbeđuje čarobnjak za automatsko generisanje Java klijent koda RESTful servisa.*

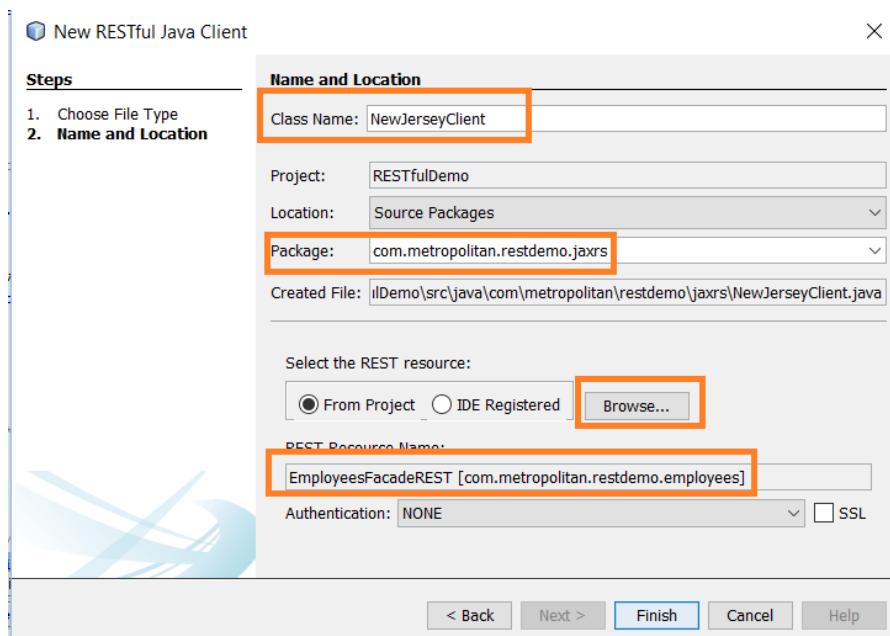
Razvojno okruženje *NetBeans IDE* obezbeđuje čarobnjak za automatsko generisanje Java klijent koda *RESTful servisa*. Ovaj kod ima zadatku da poziva metode *RESTful servisa* na osnovu odgovarajućeg *HTTP zahteva*.

Za generisanje klijent koda *RESTful servisa*, u konkretnom projektu, neophodno je, primenom razvojnog okruženja NetBeans IDE, izabrati opciju *File | New File*, a zatim u prozoru *New File* je neophodno iz kategorije *Web Services* izabrati tip datoteke *RESTful Java Client*. Navedeno je prikazano sledećom slikom.



Slika 3.1 Kreiranje datoteke tipa RESTful Java Client [izvor: autor]

U sledećem koraku čarobnjaka, neophodno je obezbediti naziv klasi, i njenom paketu, *JAX-RS* klijenta. U ovom prozoru, klikom na dugme *Browse*, bira se *RESTful servis* za koji se generiše klijent kod. U konkretnom slučaju, budući da je sve vreme bio u fokusu, biće izabrana datoteka *RESTful servisa* pod nazivom *EmployeesFacadeREST*. Navedeno je prikazano sledećom slikom.



Slika 3.2 Podešavanje datoteke RESTful Java klijenta [izvor: autor]

Klikom na *Finish*, željeni kod počinje sa generisanjem.

## JAVA KOD RESTFUL KLIJENTA

*Sledi analiza automatski generisanog Java koda RESTful klijenta.*

Posmatra se podrazumevani naziv klase RESTful klijenta. Jersey je JAX-RS implementacija povezana sa GlassFish serverom. Pošto se u razvoju koristi navedeni aplikativni server, uključen u NetBeans IDE, ovo razvojno okruženje, po podrazumevanim podešanjima, koristi naziv NewJerseyClient za polje Class Name čarobnjaka za generisanje koda RESTful klijenta.

Za RESTful veb servis *EmployeesFacadeREST.java*, praćenjem prethodno opisanih i prikazanih koraka NetBeans IDE čarobnjaka, generisan je sledeći Java kod RESTful klijenta.

Generisani Java kod koristi JAX-RS client API predstavljen u JAX-RS 2.0.

Ono što je moguće primetiti, iz priloženog koda, sve metode konkretnog RESTful servisa dobine su metode omotače u klijent kodu. Postoje dve verzije ovih metoda - koje produkuju ili troše XML ili JSON podatke. Svaka od metoda koristi generike tako da se povratni tip metoda podešava tokom vremena izvršavanja.

```
package com.metropolitan.restdemo.service;  
  
import com.metropolitan.restdemo.Employees;
```

```
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
@Path("com.metropolitan.restdemo.employees")
public class EmployeesFacadeREST extends AbstractFacade<Employees> {

    @PersistenceContext(unitName = "RESTfulDemoPU")
    private EntityManager em;

    public EmployeesFacadeREST() {
        super(Employees.class);
    }

    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void create(Employees entity) {
        super.create(entity);
    }

    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Integer id, Employees entity) {
        super.edit(entity);
    }

    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") Integer id) {
        super.remove(super.find(id));
    }

    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Employees find(@PathParam("id") Integer id) {
```

```
        return super.find(id);
    }

    @GET
    @Override
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Employees> findAll() {
        return super.findAll();
    }

    @GET
    @Path("{from}/{to}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Employees> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
        return super.findRange(new int[]{from, to});
    }

    @GET
    @Path("count")
    @Produces(MediaType.TEXT_PLAIN)
    public String countREST() {
        return String.valueOf(super.count());
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }
}
```

## PRIKAZIVANJE ZAPISA

*Kao i za prethodne datoteke, neophodno je obaviti testiranje kreiranog koda klijenta.*

Najlakši način da se testira generisani kod *klijenta RESTful servisa* jeste da se koriste stringovi. Na primer, moguće je upotrebiti sledeću metodu: *find\_XML(Class<T> responseType, String id)* i to na sledeći način:

```
package com.metropolitan.restdemo.klijenttest;

import com.metropolitan.restdemo.jaxrs.NewJerseyClient;

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

```
public static void main(String[] args) {
    NewJerseyClient newJerseyClient = new NewJerseyClient();
    String response = newJerseyClient.find_XML(
        String.class, "1");
    System.out.println("Odgovor je: " + response);
    newJerseyClient.close();
}
```

Pokretanjem ove klase, u *Output* monitoru razvojnog okruženja *NetBeans IDE*, moguće je primetiti izlaz prikazan sledećom slikom:

Slika 3.3 Testiranje metode findXML() koda klijenta

Za bolju preglednost generisanog izlaza, kreiran je sledeći XML listing koji je ekvivalentan odgovoru sa Slike 3.

```
Odgovor je: <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employees>
    <birthDate>1974-09-0T00:00:00+01:00</birthDate>
    <empNo>1</empNo>
    <firstName>Vladimir</firstName>
    <gender>M</gender>
    <hireDate>2020-09-0T00:00:00+02:00</hireDate>
    <lastName>Milicevic</lastName>
</employees>
```

Dalje je moguće manipulisati XML odgovorom, ili JSON da je bila testirana metoda `findJSON()` koda klijenta, na dobro poznate načine.

## DODAVANJE NOVOG ZAPISA

*Moguće je izvršiti i testiranje dodavanja zapisa u bazu podataka.*

Kao dodatak testiranju kreiranog koda klijenta *RESTful servisa* moguće je izvršiti i testiranje dodavanja zapisa u bazu podataka. Neka je kreirana nova klasa pod nazivom *Main1.java* koja implementira metodu `create_XML()` čiji će zadatak biti dodavanje novog zapisa u tabelu *employees* baze podataka *company*. Sledećim listingom je priložen kod ove klase:

```
package com.metropolitan.restdemo.klijenttest;

import com.metropolitan.restdemo.jaxrs.NewJerseyClient;

/**
 *
 * @author Vladimir Milicevic
```

```

/*
public class Main1 {
    public static void main(String[] args) {
String xml = "<employees>\n" +
"            <birthDate>1988-09-22</birthDate>\n" +
"            <empNo>5</empNo>\n" +
"            <firstName>Petar</firstName>\n" +
"            <gender>M</gender>\n" +
"            <hireDate>2018-09-27</hireDate>\n" +
"            <lastName>Markovic</lastName>\n" +
"        </employees> ";
NewJerseyClient newJerseyClient = new NewJerseyClient();
newJerseyClient.create_XML(xml);
newJerseyClient.close();
}
}

```

Priloženim klijent kodom, generisan je podatak u *XML* formatu, koji *RESTful* servis *EmployeesFacadeREST* razume, i prosleđen je metodi *create\_XML()* generisane klase *klijenta RESTful servisa*. Ova klasa je pozvala konkretni veb servis i izvršila umetanje novog reda u tabelu *employees* baze podataka *company*. To može biti provereno i u *MySQL Workbench* okruženju, izvršavanjem SQL upita:

```
select * from employees
```

za proveru da li je poslednji zapis zaista se našao u bazi podataka. Da je sve urađeno na pravi način prikazano sledećom slikom.

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	1	1974-09-20	Vladimir	Milicevic	M	2020-09-20
	2	1980-07-21	Petar	Nikolic	M	2012-10-12
	3	1978-05-18	Jovana	Markovic	M	2013-11-11
	4	1984-06-20	Marija	Jovanovic	M	2021-09-20
	5	1988-09-22	Petar	Markovic	M	2018-09-27

Slika 3.4 Dodatna provera u MySQL Workbench okruženju [izvor: autor]

## PRIKAZIVANJE SVIH ZAPISA

*Moguće je izvršiti još neka testiranja generisanog koda klijenta RESTful servisa.*

Takođe, korektnost prethodno urađenog testa može se proveriti na još jedan način, a to je primenom novog testa. Ovaj put zadatak će biti kreiranje nove klase koja će implementirati metodu *findAll\_XML()* čijim pozivom bi trebalo da budu izlistani, u XML formatu, svi zapisi iz tabele *employees* baze podataka *company*. Nova Java klasa, koja će služiti kao test klasa za ovaj slučaj, nazvana je *Main2.java* i sledi njen listing:

```

package com.metropolitan.restdemo.klijenttest;

import com.metropolitan.restdemo.jaxrs.NewJerseyClient;

/**
 *
 * @author Vladimir Milicevic
 */
public class Main2 {
    public static void main(String[] args) {
        NewJerseyClient newJerseyClient = new NewJerseyClient();
        String response = newJerseyClient.findAll_XML(
                String.class);
        System.out.println("Odgovor je: " + response);
        newJerseyClient.close();
    }
}

```

Pokretanjem ove klase, u *Output* monitoru razvojnog okruženja *NetBeans IDE*, moguće je primetiti izlaz prikazan sledećom slikom u kojem poslednji zapis predstavlja onaj koji je u prethodnom izlaganju dodat u tabelu *employees* baze podataka *company*.



```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><employees><employee><birthDate>1974-09-20T00:00:00+01:00</birthDate><empNo>1</empNo><firstName>Vladimir</firstName><lastName>Milicevic</lastName><gender>M</gender><hireDate>2020-09-20T00:00:00+02:00</hireDate><employees><employee><birthDate>1980-07-21T00:00:00+02:00</birthDate><empNo>2</empNo><firstName>Petar</firstName><lastName>Nikolic</lastName><gender>M</gender><hireDate>2012-10-12T00:00:00+02:00</hireDate><employees><employee><birthDate>1978-05-18T00:00:00+01:00</birthDate><empNo>3</empNo><firstName>Jovan</firstName><lastName>Markovic</lastName><gender>M</gender><hireDate>2013-11-11T00:00:00+01:00</hireDate><employees><employee><birthDate>1984-06-20T00:00:00+02:00</birthDate><empNo>4</empNo><firstName>Marija</firstName><lastName>Marinkovic</lastName><gender>F</gender><hireDate>2021-05-20T00:00:00+02:00</hireDate><employees><employee><birthDate>1998-03-22T00:00:00+02:00</birthDate><empNo>5</empNo><firstName>Petar</firstName><lastName>Markovic</lastName><gender>M</gender><hireDate>2018-09-17T00:00:00+02:00</hireDate><employees>
BUILD SUCCESSFUL (total time: 2 seconds)

```

Slika 3.5 Testiranje metode findAll\_XML() koda klijenta [izvor: autor]

Za bolju preglednost generisanog izlaza, kreiran je sledeći XML listing koji je ekvivalentan odgovoru sa Slike 5.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employees>
    <employees>
        <birthDate>1974-09-20T00:00:00+01:00</birthDate>
        <empNo>1</empNo>
        <firstName>Vladimir</firstName>
        <gender>M</gender>
        <hireDate>2020-09-20T00:00:00+02:00</hireDate>
        <lastName>Milicevic</lastName>
    </employees>
    <employees>
        <birthDate>1980-07-21T00:00:00+02:00</birthDate>
        <empNo>2</empNo>
        <firstName>Petar</firstName>
        <lastName>Nikolic</lastName>
        <gender>M</gender>
        <hireDate>2012-10-12T00:00:00+02:00</hireDate>
    </employees>
    <employees>
        <birthDate>1978-05-18T00:00:00+01:00</birthDate>
        <empNo>3</empNo>

```

```
<firstName>Jovana</firstName>
<gender>M</gender>
<hireDate>2013-11-11T00:00:00+01:00</hireDate>
<lastName>Markovic</lastName>
</employees>
<employees>
    <birthDate>1984-06-20T00:00:00+02:00</birthDate>
    <empNo>4</empNo>
    <firstName>Marija</firstName>
    <gender>M</gender>
    <hireDate>2021-09-20T00:00:00+02:00</hireDate>
    <lastName>Jovanovic</lastName>
</employees>
<employees>
    <birthDate>1988-09-22T00:00:00+02:00</birthDate>
    <empNo>5</empNo>
    <firstName>Petar</firstName>
    <gender>M</gender>
    <hireDate>2018-09-27T00:00:00+02:00</hireDate>
    <lastName>Markovic</lastName>
</employees>
</employees>
```

## ▼ Poglavlje 4

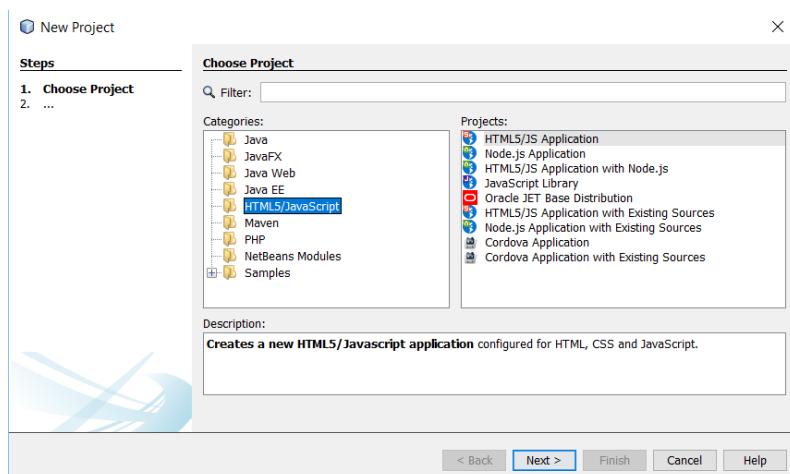
# Generisanje JavaScript RESTful klijenta

## KREIRANJE PROJEKTA JAVASRCIPT RESTFUL KLIJENTA

*Poželjno je JavaSript RESTful klijente kreirati u posebnom projektu.*

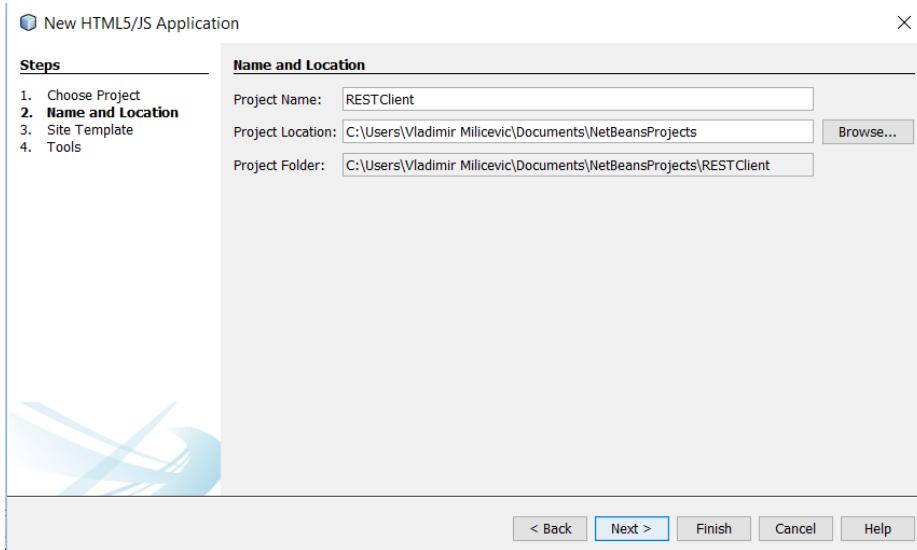
U prethodnom izlaganju je pokazano kako se u razvojnog okruženju NetBeans IDE kreiraju *Java klijenti RESTful servisa*. U opštem slučaju, u savremenim aplikacijama, uporedno se razvijaju *RESTful servisi*, *Java klijenti RESTful servisa* i *JavaScript klijenti RESTful servisa* koji se izvršavaju u veb pregledaču. Kao što na pojednostavljen i elegantan način razvojno okruženje NetBeans IDE omogućava razvoj *Java klijenata RESTful servisa*, tako omogućava i razvoj *JavaScript klijenata RESTful servisa*.

Za generisanje JavaScript klijenata RESTful servisa, prvo će biti kreiran projekat iz kategorije *HTML5 / JavaScript*.



Slika 4.1 Kreiranje projekta iz kategorije HTML5 / JavaScript [izvor: autor]

Za sada je dovoljno dodeliti naziv projektu i prihvati ponuđena podešavanja za ovaj projekat. Projekat će nositi naziv *RESTClient*, a to je moguće videti iz sledeće slike.



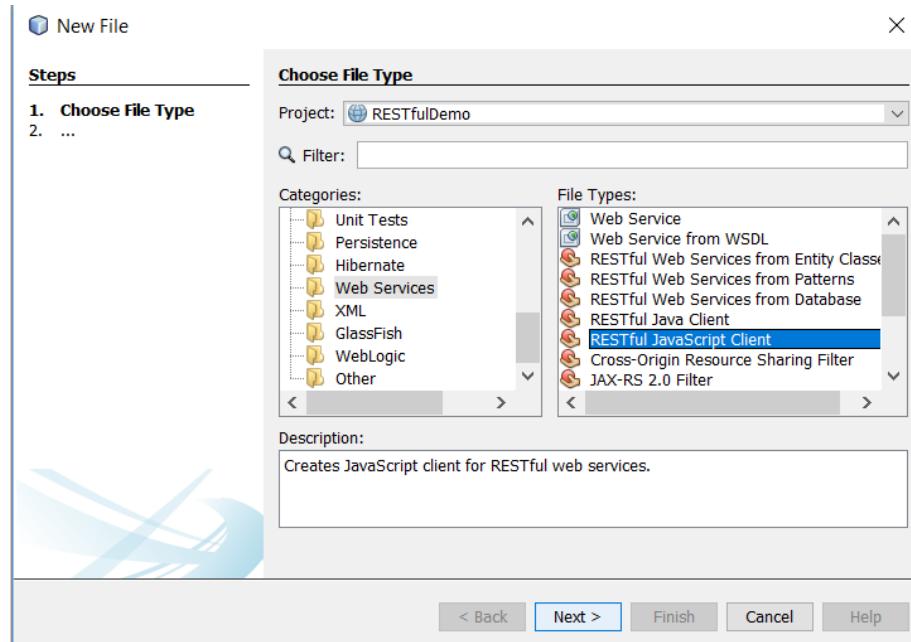
Slika 4.2 Dodela naziva projektu iz kategorije HTML5 / JavaScript [izvor: autor]

Klikom na *Finish*, projekat je kreiran i moguće je sada pristupiti kreiranju datoteka koje odgovaraju *JavaScript klijentima RESTful servisa*. Na ovome će se insistirati u sledećem izlaganju.

## KREIRANJE DATOTEKA JAVASRCIPT RESTFUL KLIJENATA

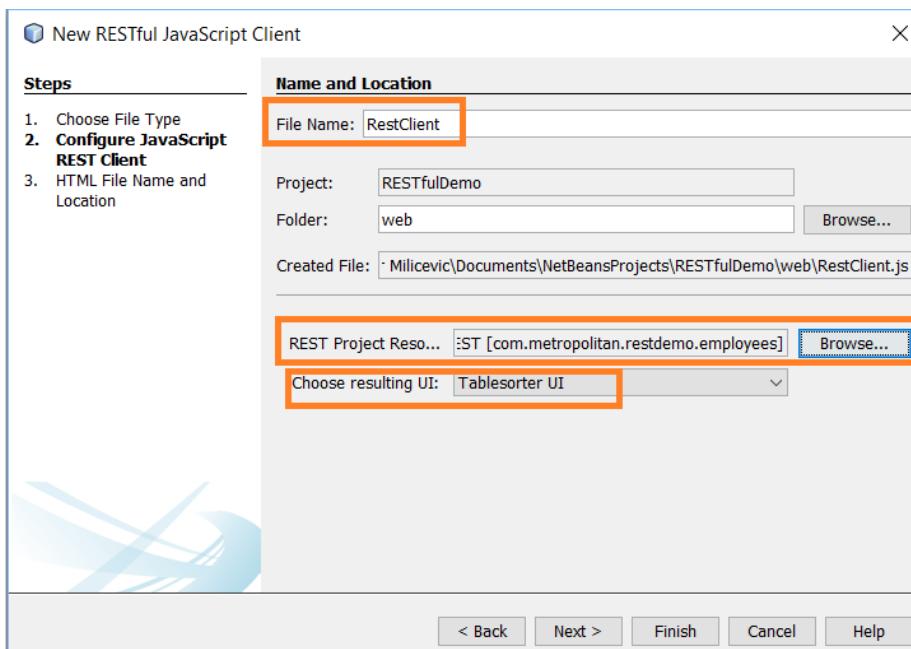
*U nastavku, u kreiranom projektu je neophodno kreirati datoteke JavaScript RESTFul klijenata.*

Pošto je nov projekat uspešno kreiran, moguće je pristupiti kreiranju datoteka *JavaScript RESTFul klijenata*. Procedura započinje izborom opcija *File | New*, za kreirani projekat *RESTClient*, nakon čega se u prozoru *New File*, iz kategorije *Web Services*, bira tip datoteke *RESTFul JavaScript Client*. Navedeno je prikazano sledećom slikom.



Slika 4.3 Izbor kreiranja datoteke tipa RESTful JavaScript Client [izvor: autor]

Klikom na dugme *Next*, otvara se prozor pod nazivom *New RESTful JavaScript Client* koji je prikazan sledećom slikom.



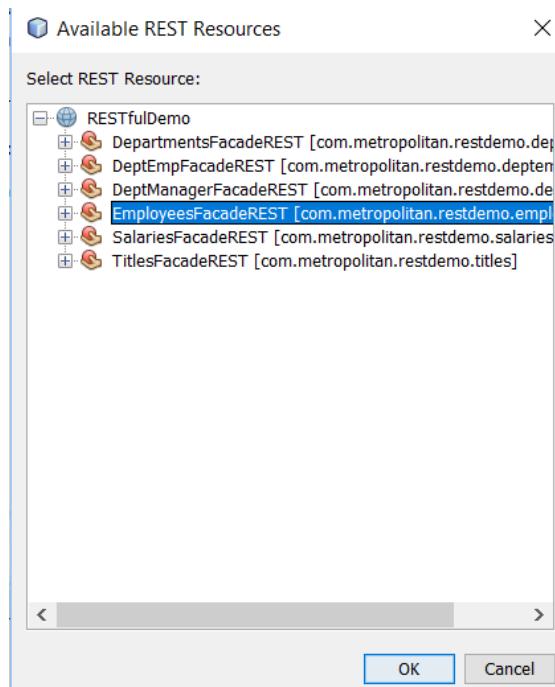
Slika 4.4 Izbor naziva, REST resursa i predefinisanog korisničkog interfejsa [izvor: autor]

U ovom prozoru vrše se podešavanja datoteke tipa *RESTful JavaScript Client* koja podrazumevaju dodelu naziva, izbor *REST* resursa, kao i nekog od predefinisanih korisničkih interfejsa. Naziv *JavaScript* datoteke će da glasi *RestClient.js*, za *REST* resurs će bit izabrana dobro poznata klasa *RESTful servisa EmployeesFacadeREST.java*, a predefinisani korisnički interfejs odgovara opciji *Tablesorter UI*.

## IZBOR REST RESURSA I KREIRANJE HTML STRANICE JAVASCRIPT KLIJENTA

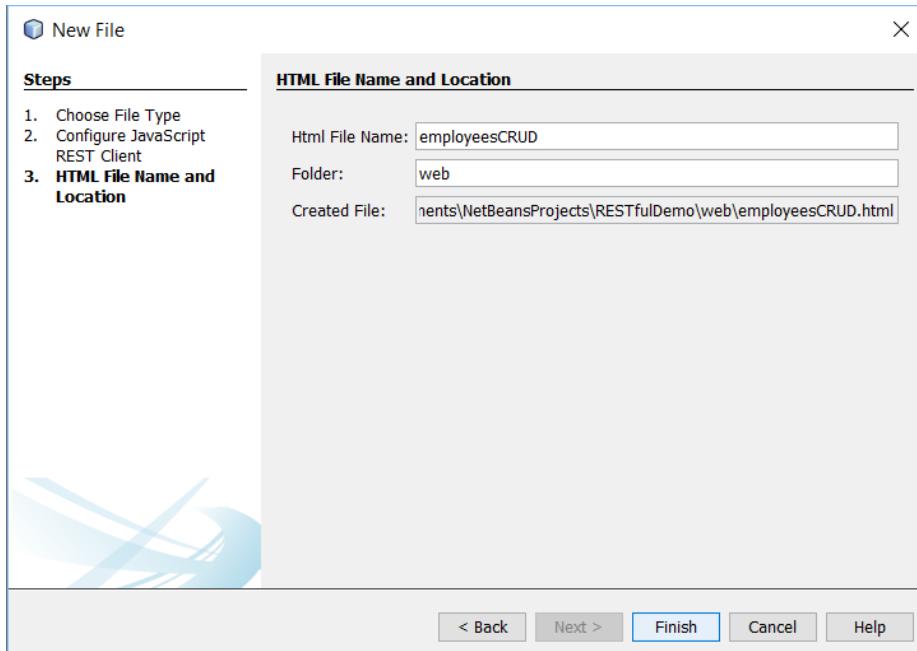
*Neophodno je kreirati i HTML stranu koja učitava odgovarajuću JS datoteku.*

Na prethodnoj slici moguće je primetiti opciju za dodavanje *REST resursa za JavaScript klijent* koji se kreira. Proces je veoma jednostavan, klikom na dugme *Browse* otvorice se prozor koji je prikazan sledećom slikom. U prozoru je prikazana lista svih dostupnih *REST resursa*. Kao što je istaknuto u prethodnom izlaganju, biće izabran dobro poznati *RESTful servis EmployeesFacadeREST.java*.



Slika 4.5 Izbor REST resursa [izvor: autor]

Klikom na *OK*, kontrola se vraća u prozor prikazan Slikom 4. Klik na dugme *Next* otvara prozor koji je prikazan Slikom 6. U ovom prozoru definiše se naziv *HTML* stranice, koja će takođe automatski biti generisana, koja učitava generisani *RestClient.js* i koja će omogućiti korisniku da izvodi *CRUD* operacije, koristeći *RESTful servis EmployeesFacadeREST.java*, nad tabelom *employees*, baze podataka *company*.



Slika 4.6 Kreiranje HTML stranice JavaScript klijenta [izvor: autor]

## GENERISANI KODOVI DATOTEKA PROJEKTA

*Kada su sva podešavanja gotova, neophodno je priložiti automatski generisane kodove.*

Kada su sva podešavanja gotova, neophodno je priložiti automatski generisane kodove za *JavaScript klijent RESTFul servisa* i njemu odgovarajuću *HTML* stranicu.

Sledi kod datoteke *RestClient.js*:

```
var app = {
    // Create this closure to contain the cached modules
    module: function () {
        // Internal module cache.
        var modules = {};

        // Create a new module reference scaffold or load an
        // existing module.
        return function (name) {
            // If this module has already been created, return it.
            if (modules[name]) {
                return modules[name];
            }

            // Create a module and save it under this name
            return modules[name] = {Views: {}};
        };
    }()
};
```

```
(function (models) {

    // Model for Employees entity
    models.Employees = Backbone.Model.extend({
        urlRoot: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.employees/",
        idAttribute: 'empNo',
        defaults: {
            firstName: "",
            lastName: "",
            gender: ""
        },
        toViewJson: function () {
            var result = this.toJSON(); // displayName property is used to render
item in the list
            result.displayName = this.get('empNo');
            return result;
        },
        isNew: function () {
            // default isNew() method implementation is
            // based on the 'id' initialization which
            // sometimes is required to be initialized.
            // So isNew() is redifined here
            return this.notSynced;
        },
        sync: function (method, model, options) {
            options || (options = {});
            var errorHandler = {
                error: function (jqXHR, textStatus, errorThrown) {
                    // TODO: put your error handling code here
                    // If you use the JS client from the different domain
                    // (f.e. locally) then Cross-origin resource sharing
                    // headers has to be set on the REST server side.
                    // Otherwise the JS client has to be copied into the
                    // some (f.e. the same) Web project on the same domain
                    alert('Unable to fulfil the request');
                }
            };
            if (method === 'create') {
                options.url = 'http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.employees/';
            }
            var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
            return result;
        }
    });

});
```

```

// Collection class for Employees entities
models.EmployeesCollection = Backbone.Collection.extend({
    model: models.Employees,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.employees/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        };
        var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',
        initialize: function (options) {
            this.options = options || {};
            this.model.bind("reset", this.render, this);
            var self = this;
            this.model.bind("add", function (modelName) {
                var row = new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el;
                $(self.el).append($(row));
                $(self.el).parent().trigger('addRows', [$(row)]);
            });
        },
        render: function (eventName) {
            var self = this;
            _.each(this.model.models, function (modelName) {
                $(this.el).append(new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el);
            });
        }
    });
})

```

```
        },
        return this;
    }
});

views.ListItemView = Backbone.View.extend({
    tagName: 'tr',

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    close: function () {
        var table = $(this.el).parent().parent();
        table.trigger('disable.pager');
        $(this.el).unbind();
        $(this.el).remove();
        table.trigger('enable.pager');
    }
};

views.ModelView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         */
```

```
        * $(this.options.templateName) is element access to the element
        * using jQuery
        */
    return _.template($(this.options.templateName).html())(json);
},

/*
 * Classes "save" and "delete" are used on the HTML controls to listen
events.
 * So it is supposed that HTML has controls with these classes.
*/
events: {
    "change input": "change",
    "click .save": "save",
    "click .delete": "drop"
},

change: function (event) {
    var target = event.target;
    console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
},

save: function () {
    // TODO : put save code here
    var hash = this.options.getHashObject();
    this.model.set(hash);
    if (this.model.isNew() && this.collection) {
        var self = this;
        this.collection.create(this.model, {
            success: function () {
                // see isNew() method implementation in the model
                self.model.notSynced = false;
                self.options.navigate(self.model.id);
            }
        });
    } else {
        this.model.save();
        this.model.el.parent().parent().trigger("update");
    }
    return false;
},

drop: function () {
    this.model.destroy({
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
            */
            window.history.back();
        }
    });
}
```

```
        return false;
    },

    close: function () {
        $(this.el).unbind();
        $(this.el).empty();
    }
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },

    render: function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    /*
     *  Class "new" is used on the control to listen events.
     *  So it is supposed that HTML has a control with "new" class.
     */
    events: {
        "click .new": "create"
    },

    create: function (event) {
        this.options.navigate();
        return false;
    }
});

})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
```

```

routes: {
    '' : 'list',
    'new' : 'create'

    ,
    ':id' : 'details'
},
initialize: function () {
    var self = this;
    $('#create').html(new views.CreateView({
        // tpl-create is template identifier for 'create' block
        templateName: '#tpl-create',
        navigate: function () {
            self.navigate('new', true);
        }
    }).render().el);
},
list: function () {
    this.collection = new models.EmployeesCollection();
    var self = this;
    this.collection.fetch({
        success: function () {
            self.listView = new views.ListView({
                model: self.collection,
                // tpl-employees-list-item is template identifier for item
                templateName: '#tpl-employees-list-item'
            });
        }
    });

    $('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()));
    if (self.requestedId) {
        self.details(self.requestedId);
    }
    var pagerOptions = {
        // target the pager markup
        container: $('.pager'),
        // output string - default is '{page}/{totalPages}'
possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
        output: '{startRow} to {endRow} ({totalRows})',
        // starting page of the pager (zero based index)
        page: 0,
        // Number of visible rows - default is 10
        size: 10
    };
    $('#datatable').tablesorter({widthFixed: true,
        widgets: ['zebra']}).
        tablesorterPager(pagerOptions);
    }
}),
details: function (id) {
    if (this.collection) {
        this.employees = this.collection.get(id);
        if (this.view) {

```

```
        this.view.close();
    }
    var self = this;
    this.view = new views.ModelView({
        model: this.employees,
        // tpl-employees-details is template identifier for chosen
model element
        templateName: '#tpl-employees-details',
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
} else {
    this.requestedId = id;
    this.list();
}
},
create: function () {
    if (this.view) {
        this.view.close();
    }
    var self = this;
    var dataModel = new models.Employees();
    // see isNew() method implementation in the model
    dataModel.notSynced = true;
    this.view = new views.ModelView({
        model: dataModel,
        collection: this.collection,
        // tpl-employees-details is a template identifier for chosen model
element
        templateName: '#tpl-employees-details',
        navigate: function (id) {
            self.navigate(id, false);
        },
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
},
getData: function () {
    return {
        empNo: $('#empNo').val(),
        firstName: $('#firstName').val(),
        lastName: $('#lastName').val(),
        gender: $('#gender').val()
    };
}
});
new AppRouter();
```

```
Backbone.history.start();
});
```

Ovu datoteku učitava *HTML* stranica pod nazivom *EmployeesCRUD.html* čiji je kod priložen sledećim listingom:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/addons/
pager/jquery.tablesorter.pager.css'>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>
        <script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
        <script src='http://mottie.github.com/tablesorter/addons/pager/
jquery.tablesorter.pager.js'></script>
        <script src='RestClient.js'></script>
    </head>
    <body>
        <div id='create'></div>

        <table id='datatable' class='tablesorter-blue'>
        </table>
        <div class='pager' id='pager'>
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
first.png' class='first' alt='First' />
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
prev.png' class='prev' alt='Prev' />
            <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
next.png' class='next' alt='Next' />
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
last.png' class='last' alt='Last' />
            <select class='pagesize'>
                <option selected='selected' value='10'>10</option>
                <option value='20'>20</option>
                <option value='30'>30</option>
                <option value='40'>40</option>
            </select>
        </div>
        <br/>
```

```
<div id='details'></div>

<!-- Templates -->
<script type='text/template' id='tpl-create'>
    <!--
        Put your controls to create new entity here.

        Class 'new' is used to listen on events in JS code.
    -->
    <button class='new'>Create</button>
</script>

<script type='text/template' id='thead'>
    <thead>
        <tr>
            <th>empNo</th>
            <th>firstName</th>
            <th>lastName</th>
            <th>gender</th>
        </tr>
    </thead>
</script>
<script type='text/template' id='tpl-employees-list-item'>
    <td><a href='#<%= empNo %>'><%= empNo %></a></td>
    <td><%= firstName %></td>
    <td><%= lastName %></td>
    <td><%= gender %></td>
</script>

<script type='text/template' id='tpl-employees-details'>
    <div>
        <table>
            <tbody>
                <tr><td>Id</td>
                <td>
                    <input type='text' id='empNo' name='id' value='<%= typeof(empNo) !==
"undefined" ? empNo : "" %>' />
                </td>
                </tr>
                <tr>
                    <td>firstName</td><td><input type='text' id='firstName' name='firstName' value='<%= firstName %>' /></td></tr>
                <tr>
                    <td>lastName</td><td><input type='text' id='lastName' name='lastName' value='<%= lastName %>' /></td></tr>
                <tr>
                    <td>gender</td><td><input type='text' id='gender' name='gender' value='<%= gender %>' /></td></tr>
            </tbody>
        </table>
    <!--
        Put your controls to create new entity here.
    -->
</script>
```

```
Classes 'save' and 'delete' are used to listen on events in JS code.  
-->  
<button class='save'>Save</button>  
<button class='delete'>Delete</button>  
</div>  
</script>  
  
</body>  
</html>
```

## TESTIRANJE KREIRANOG JAVASCRIPT KLIJENTA RESTFUL SERVISA

*Nakon kreiranja odgovarajućih datoteka JavaScript klijenta RESTFul servisa, sledi testiranje.*

Zbog problema tokom testiranja koji su se javili koristeći veb pregledač Mozilla Firefox, primer je uspešno testiran u veb pregledaču Edge. Za testiranje pomoću pregledača Mozilla Firefox i Google Chrome, neophodno je dodavanje dodatka (plugin) za HTTP access control (CORS). Više o ovome moguće je pogledati na linku: [https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)

Prevođenjem oba projekta, i izborom opcije *Deploy* za projekat *RESTFulDemo*, moguće je pristupiti testiranju kreiranog *JavaScript klijenta RESTFul servisa*. Navođenjem linka <http://localhost:8383/RESTClient/EmployeesCRUD.html> u veb pregledaču, ili desnim klikom da datoteku *EmployeesCRUD.html* u projektu i izborom opcije *Run file*, veb pregledač učitava navedenu stranicu, a zajedno sa njom i kreirani *JavaScript* kod klijenta *RESTFul servisa*. Navedeno je prikazano sledećom stranicom.

Create				
empNo	firstName	lastName	gender	
1	Vladimir	Milicevic	M	
2	Petar	Nikolic	M	
3	Jovana	Markovic	F	
4	Marija	Ješanovic	F	
5	Petar	Markovic	M	

Slika 4.7 Stranica EmployeesCRUD sa odgovarajućim UI [izvor: autor]

Kao što je moguće primetiti stranica je učitala sve raspoložive zapise koji postoje u tabeli *employees*, baze podataka *company*. Dalje, klikom na dugme *Create*, omogućeno je dodavanje novog zapisa u navedenu tabelu. Ovo je prikazano sledećom slikom.

empNo	firstName	lastName	gender
1	Vladimir	Milicevic	M
2	Petar	Nikolic	M
3	Jovana	Markovic	F
4	Marija	Jovanovic	F
5	Petar	Markovic	M
6	Zorana	Nikolic	F

(1) (2) (3) (4) [10 ▾]  
 Id: 6  
 firstName: Zorana  
 lastName: Nikolic  
 gender: F  
 Save | Delete

Slika 4.8 Dodavanje novog zapisa u tabelu [izvor: autor]

Nov zapis, nakon popunjavanja forme sa slike, klikom na dugme **Save** prosledjuje se u tabelu *employees*, baze podataka *company*. Zapise je moguće ažurirati i brisati na ovoj stranici. Klikom na ID (kolona *empNo*) u formu se učitavaju podaci izabranog radnika, nakon izmene u formi, klikom na **Save** oni se ažuriraju. Na isti način se bira i zaposleni čije podatke je neophodno izbrisati iz baze podataka (klik na dugme **Delete**).

empNo	firstName	lastName	gender
1	Vladimir	Milicevic	M
2	Petar	Nikolic	M
3	Jovana	Markovic	F
4	Marija	Jovanovic	F
5	Petar	Markovic	M
6	Zorana	Nikolic	F

(1) (2) (3) (4) [10 ▾]  
 Id: 6  
 firstName: Zorana  
 lastName: Nikolic  
 gender: F  
 Save | Delete

Slika 4.9 RESTFul poziv delete [izvor: autor]

## VIDEO MATERIJALI

*Izlaganje problematike RESTFul veb servisi sa JAX - RS biće zaokruženo odgovarajućim video materijalima.*

A Little REST with JAX-RS 2.0 and Java EE 7

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

Beautiful REST + JSON APIs with JAX-RS and Jersey

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

## ▼ Poglavlje 5

### Uvod u SOAP servise

#### PRISTUP KREIRANJU VEB SERVISA

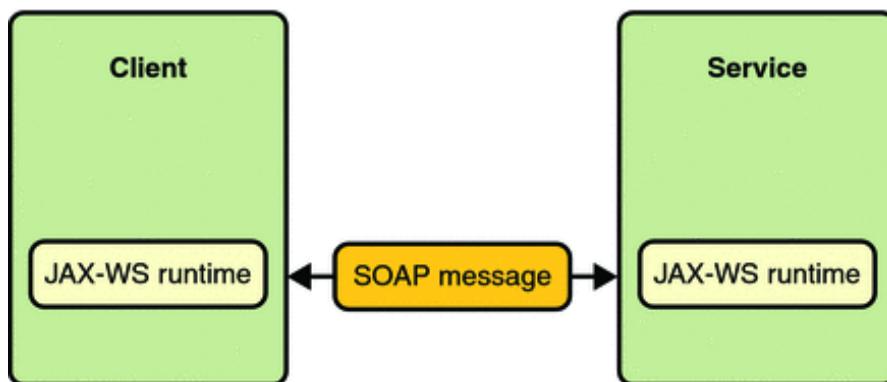
*Postoje dva različita pristupa, koji se i najčešće koriste, kada je u pitanju razvoj veb servisa.*

U daljem radu, od posebnog značaja će biti analiza i diskusija u vezi sa razvojem veb servisa primenom pristupa **SOAP**. Kada se koristi **SOAP protokol (SOAP Protocol)**, operacije veb servisa su definisane u XML dokumentu pod nazivom **Web Services Definition Language (WSDL)**. Nakon kreiranja **WSDL** datoteke, izvodi se implementacija veb servisa primenom pogodnog programskog jezika, kakav je, između ostalih, i programski jezik Java.

Proces kreiranja **WSDL** datoteke je veoma zahtevan zadatak podložan brojnim greškama. Kao velika pomoć, u obavljanju navedenog zadatka, javlja se upotreba **Java EE 7** platforme. Primenom **Java EE 7** platforme, kreiranje **WSDL** datoteke može biti automatski generisano iz veb servisa napisanog Java programskim jezikom kada je ovaj servis pridružen aplikativnom serveru.

Ukoliko postoji **WSDL** datoteka i potreba za razvojem operacija veb servisa u programskom jeziku Java, razvojno okruženje NetBeans IDE može automatski da generiše većinu implementacionog koda, kreiranjem klase sa metodama koje odgovaraju pojedinačnim operacijama veb servisa.

Kao što je moguće primetiti, i u ovom slučaju programer je pošteđen obavljanja zamornih pratećih zadataka i neophodno je da se fokusira isključivo na implementaciju konkretnе logike navedenih metoda veb servisa.



Slika 5.1 Komunikacija servisa i klijenta preko SOAP poruka [izvor: Oracle]

## ▼ Poglavlje 6

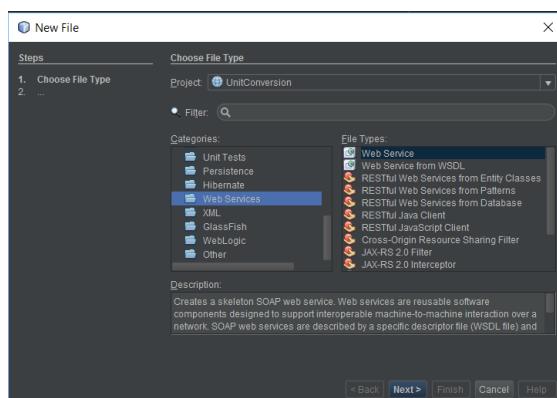
# Kreiranje jednostavnog veb servisa

## PROJEKAT VEB SERVISA

*Akcenat je na primeru koji je izabran za analizu i demonstraciju postupka kreiranja veb servisa*

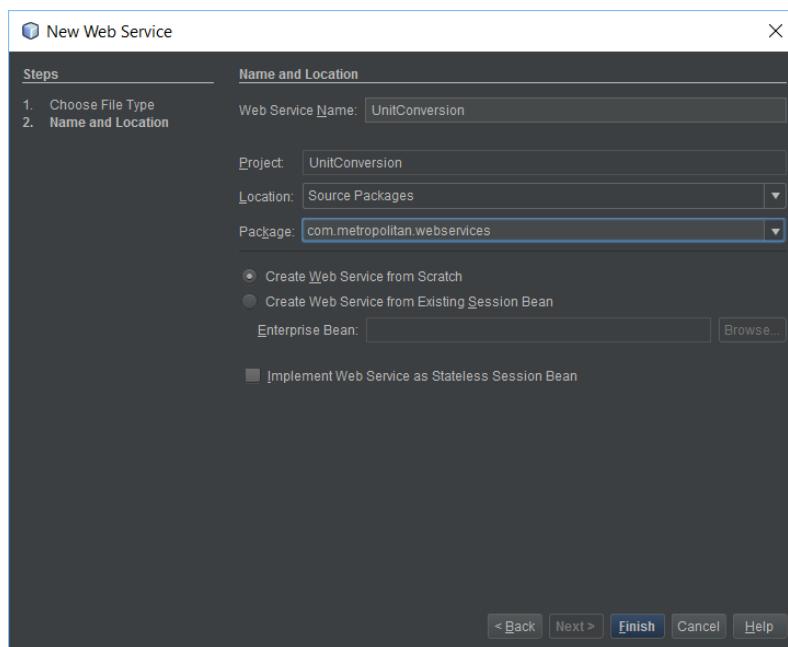
U ovom delu lekcije akcenat će biti na primeru koji je izabran za analizu i demonstraciju postupka kreiranja veb servisa. Projekat će nositi naziv *UnitConversion* i imaće jednostavan zadatak da konvertuje vrednosti jedinica za merenje dužine. U konkretnom slučaju program će moći da konvertuje inče u centimetre i postojaće mogućnost obrnute konverzije, centimetara u inče.

Nakon kreiranja Java veb projekta, pod nazivom *UnitConversion*, pristupa se kreiranju konkretnih datoteka ovog projekta. Za početak biće kreiran WEB servis, na primer desnim klikom na projekat, izborom opcija *File | New File*. U nastavku se otvara poznati prozor *New File*, u kojem se bira kategorija *Web Services* i u okviru nje tip datoteke *Web Service*. Navedeno je prikazano sledećom slikom.



Slika 6.1 Kreiranje datoteke veb servisa [izvor: autor]

U nastavku, klikom na dugme **Next**, otvara se prozor u kojem je neophodno dodeliti naziv datoteci veb servisa koji se kreira, kao i odgovarajući paket. Navedena aktivnost je ilustrovana sledećom slikom.



Slika 6.2 Podešavanje naziva i paketa veb servisa [izvor: autor]

Klikom na dugme Finish, datoteka veb servisa sa inicijalnim kodom je generisana i ovome će posebno biti govora u narednom izlaganju.

## GENERISANI KODA VEB SERVISA

*Neophodno je posvetiti pažnju generisanom kodu veb servisa.*

Kao što je istaknuto u prethodnom izlaganju, klikom na dugme *Finish* (Slika 2) generiše se datoteka sa inicijalnim kodom veb servisa koji je priložen sledećim listingom:

```
package com.metropolitan.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "UnitConversion")
public class UnitConversion {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

```
}
```

Kao što je moguće primetiti, razvojno okruženje *NetBeans IDE* je automatski generisalo jednostavan "Hello World" veb servis. Na nivou klase primenjena je anotacija **@WebService**. To direktno znači da je kreirana klasa označena kao veb servis.

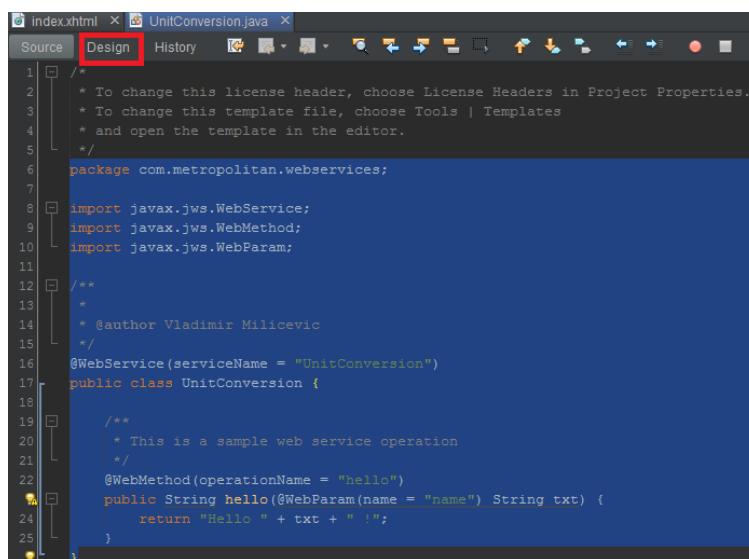
Na nivou metode, koristi se anotacija **@WebMethod** čiji je zadatak da označi metodu kao operaciju veb servisa. Atribut **operationName**, anotacije **@WebMethod**, definiše naziv operacije veb servisa. Upravo ovaj naziv će koristiti klijenti veb servisa. Anotacijom **@WebParam** biće definisane osobine parametara operacija veb servisa. U generisanom veb servisu, atribut **name** je upotrebljen da specificira naziv parametra u WSDL koji se generiše kada se veb servis angažuje.

## GRAFIČKI DIZAJNER VEB SERVISA

*Razvojno okruženje omogućava modifikovanje generisanih veb servisa na jednostavan način.*

Razvojno okruženje *NetBeans IDE* omogućava modifikovanje generisanih veb servisa na jednostavan način, primenom grafičkog interfejsa. Primenom ove pomoći na jednostavan način je moguće dodavati i / ili uklanjati operacije veb servisa, tako i upravljati parametrima, odgovarajućim metodama i anotacijama koje su automatski dodata u kod na već prikazani način.

Za pristup grafičkom dizajneru veb servisa, neophodno je jednostavno kliknuti na dugme *Design* koje je prikazano sledećom slikom.



Slika 6.3 Pristup grafičkom dizajneru veb servisa [izvor: autor]

Klikom na navedeno dugme otvara se **grafički dizajner veb servisa** koji je prikazan sledećom slikom.



Slika 6.4 Grafički dizajner web servisa [izvor: autor]

## MODIFIKOVANJE GENERISANIH VEB SERVISA

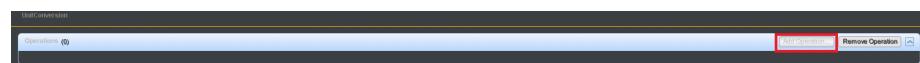
*Sledi demonstracija modifikacije generisanih veb servisa.*

Nakon otvaranja grafičkog dizajnera generisanih veb servisa sledi njihovo modifikovanje. Prvi korak koji je neophodno izvesti jeste uklanjanje automatski implementiranih operacija. Izborom operacije *hello* i opcije *Remove* (Slika 5) vrši se brisanje navedene operacije.

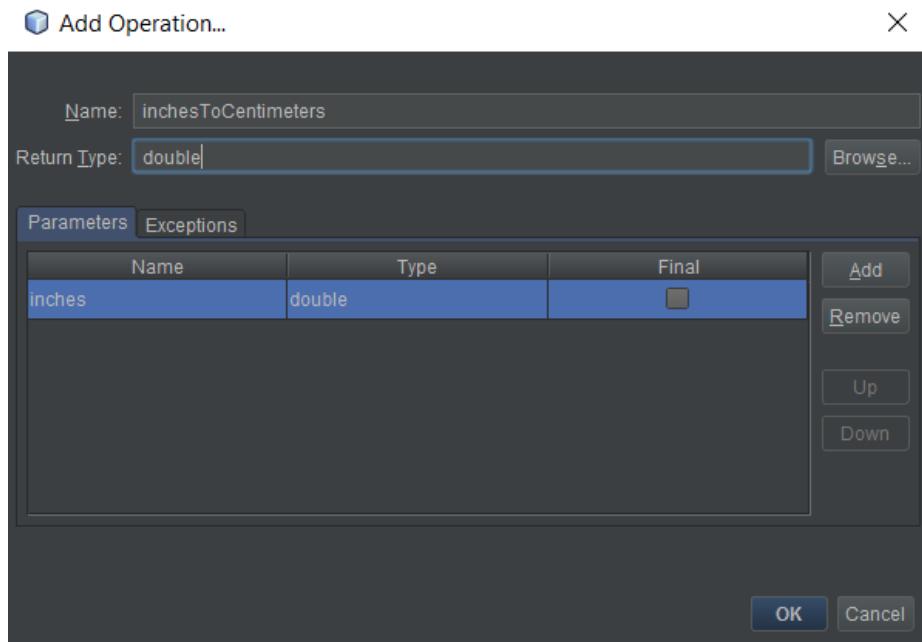


Slika 6.5 Uklanjanje automatski generisane operacije web servisa [izvor: autor]

Sada je moguće pristupiti dodavanju novih operacija veb servisa. Neophodno je izvršiti klik na dugme *Add Operation* (Slika 6) i popuniti odgovarajuće podatke u prozoru koji se ubrzo nakon klika otvara (Slika 7).



Slika 6.6 Dodavanje novih operacija web servisa [izvor: autor]



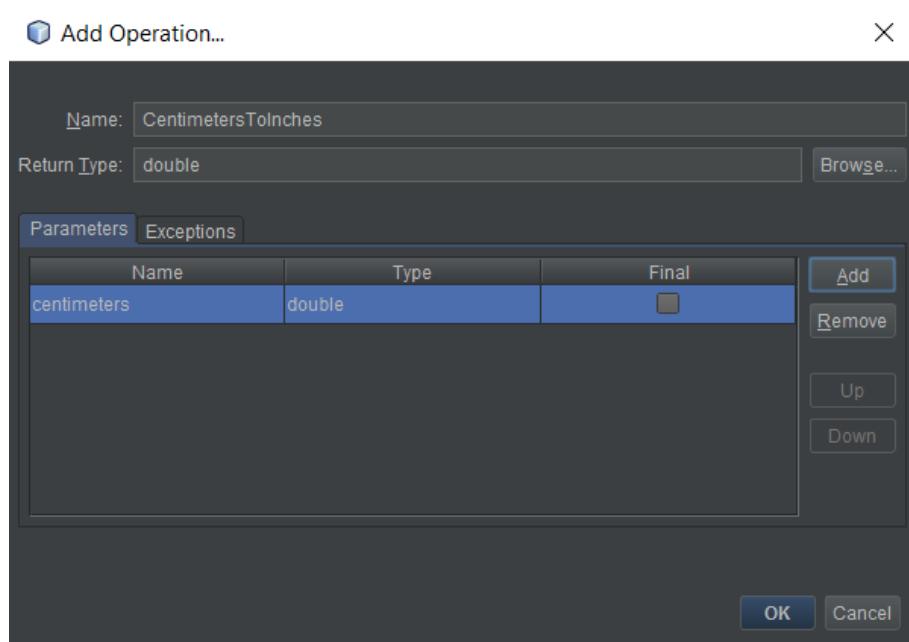
Slika 6.7 Prozor za dodavanje operacija web servisa [izvor: autor]

Na ovaj način je kreirana operacija `inchesToCentimeters()`. Klikom na dugme `Add` kreiran je i parametar `inches` koji će biti tipa `double`. Pored naziva operacije, definisan je i povratni tip odgovarajuće metode u polju `Return Type`.

## O METODAMA VEB SERVISA

*Neophodno je generisati dve metode veb servisa za konverziju veličina.*

U nastavku je moguće omogućiti i servis koji omogućava obrnutu konverziju, iz centimetara u inče. Kreiranje ove metode je prikazano sledećom slikom.

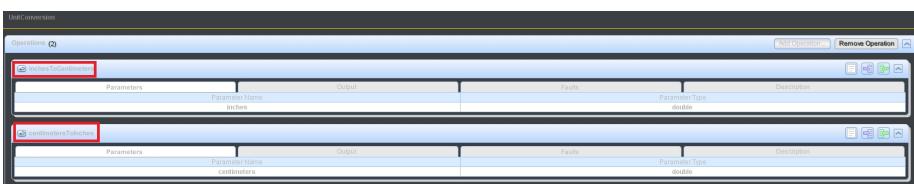


Slika 6.8 Kreiranje druge metode web servisa [izvor: autor]

Na ovaj način je kreiranja operacija *CentimetersToInches()*. Klikom na dugme *Add* kreiran je i parametar *centimeters* koji će biti tipa **double**. Pored naziva operacije, definisan je i povratni tip odgovarajuće metode u polju *Return Type*.

Kao dodatak kreiranju operacija web servisa, razvojno okruženje *NetBeans IDE* daje mogućnost kontrole kvaliteta podešavanja servisa jednostavnim selektovanjem ili deselektovanjem odgovarajuće **CheckBox** kontrole u *Design* prozoru. Web servisi šalju podatke kao *XML* tekstualne poruke između web servisa i njegovog klijenta. Ponekad je neophodno poslati i binarne podatke, kao što su na primer slike. Binarni podaci su normalno u vezi sa SOAP porukama pomoću mehanizma **MTOM** (*Message Transmission Optimization Mechanism*). Ovim mehanizmom se binarni podatak šalje kao dodatak (*attachment*) uz poruku. Na ovaj način je prenos binarnih podataka efikasniji. Primenom *NetBeans IDE* razvojnog okruženja upotreba MTOM mehanizma se jednostavno omogućava izborom opcije *Optimize Transfer Of Binary Data (MTOM)* u *Design* prozoru.

Nakon kreiranja obeju operacija web servisa, moguće je primetiti u prozoru grafičkog dizajnera web servisa da su ove metode zauzele svoje mesto. Navedeno je prikazano sledećom slikom.



Slika 6.9 Kreirane metode web servisa [izvor: autor]

Sada je moguće izvršiti, po potrebi, dodatna podešavanja ovih metoda i o tome će biti više govora u narednom izlaganju.

## DODATNA PODEŠAVANJA ZA VEB SERVISE

*Na kreirane operacije veb servisa moguće je primeniti dodatna podešavanja.*

Na kreirane operacije veb servisa moguće je primeniti dodatna podešavanja koja su prikazana sledećom slikom.



Slika 6.10 Dodatna podešavanja metoda veb servisa [izvor: autor]

Kao dodatak kreiranju operacija veb servisa, razvojno okruženje *NetBeans IDE* daje mogućnost kontrole kvaliteta podešavanja servisa jednostavnim selektovanjem ili deselektovanjem odgovarajuće **CheckBox** kontrole u *Design* prozoru.

Veb servisi šalju podatke kao *XML* tekstualne poruke između veb servisa i njegovog klijenta. Ponekad je neophodno poslati i binarne podatke, kao što su na primer slike. Binarni podaci su normalno u vezi sa SOAP porukama pomoću mehanizma *MTOM* (*Message Transmission Optimization Mechanism*). Ovim mehanizmom se binarni podatak šalje kao dodatak (*attachment*) uz poruku. Na ovaj način je prenos binarnih podataka efikasniji. Primenom *NetBeans IDE* razvojnog okruženja upotreba MTOM mehanizma se jednostavno omogućava izborom opcije *Optimize Transfer Of Binary Data (MTOM)* u *Design* prozoru.

Selektovanjem opcije *Reliable Message Delivery* ukazuje se da je cilj obezbeđivanje slanja poruke najmanje jedanput ali ne više od jedanput. Na ovaj način aplikacija omogućava oporavak u situaciji kada je poruka izgubljena negde u prenosu.

Izborom opcije *Secure Service* omogućavaju se bezbednosni alati, kao što je enkripcija poruka između klijenta i servera i / ili zahtev za omogućavanjem autentifikacije klijenta za veb servis.

Sledi trenutni listing datoteke veb servisa sa prethodno kreiranim metodama:

```
package com.metropolitan.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "UnitConversion")
public class UnitConversion {

    /**
     * Web service operation
     */
}
```

```
@WebMethod(operationName = "inchesToCentimeters")
public double inchesToCentimeters(@WebParam(name = "inches") double inches) {
    //TODO write your implementation code here:
    return 0.0;
}

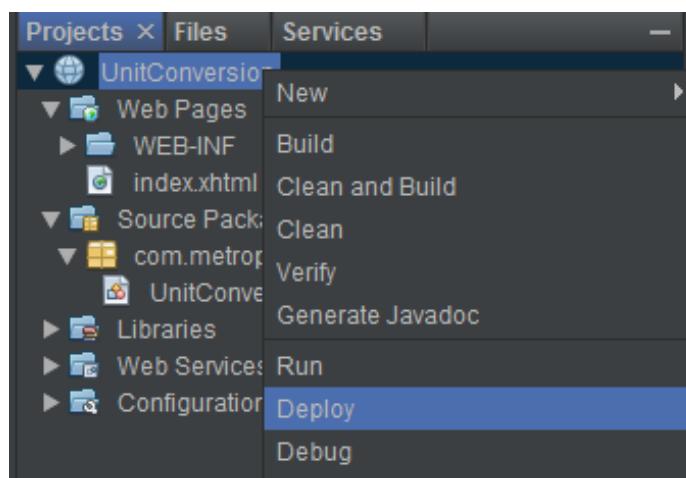
/**
 * Web service operation
 */
@WebMethod(operationName = "centimetersToInches")
public double centimetersToInches(@WebParam(name = "centimeters") double
centimeters) {
    //TODO write your implementation code here:
    return 0.0;
}
}
```

## REDEFINISANJE KREIRANIH METODA VEB SERVISA

*Sada je moguće redefinisati tela generisanih metoda klase veb servisa.*

Konačno, sada je moguće redefinisati tela generisanih metoda klase veb servisa ugrađivanjem željenog koda. Konkretno, metode moraju da obezbede obavljanje konverzije iz centimetara u inče i obrnuto. To znači, da metoda koja konvertuje centimetre u inče, broj centimetara mora da pomnoži sa 2,54 i da vrati rezultat. Obrnuto, metoda koja konvertuje inče u centimetre, broj inča mora da podeli sa 2,54 i da vrati rezultat.

Kada se izvrši redefinisanje metoda implementiranjem željenih funkcionalnosti, sve je spremno za angažovanje i testiranje kreiranog veb servisa. Ovo je omogućeno desnim klikom na projekat i izborom opcije *Deploy* (pogledati Sliku 11).



Slika 6.11 Angažovanje kreiranog servisa [izvor: autor]

Konačna verzija koda kreirane klase veb servisa *UnitConversion.java* priložena je sledećim listingom.

```
package com.metropolitan.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "UnitConversion")
public class UnitConversion {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "inchesToCentimeters")
    public double inchesToCentimeters(@WebParam(name = "inches") double inches) {
        //TODO write your implementation code here:
        return inches / 2.54;
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "centimetersToInches")
    public double centimetersToInches(@WebParam(name = "centimeters") double centimeters) {
        //TODO write your implementation code here:
        return centimeters * 2.54;
    }
}
```

## VIDEO MATERIJAL : HOW TO CREATE, DEPLOY AND TEST JAX-WS SOAP BASED WEB SERVICES IN NETBEANS ?

*Sledi video materijal za podršu tekućeg izlaganja u lekciji.*

How to create, deploy and test [JAX-WS SOAP](#) based Web Services in NetBeans ?

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

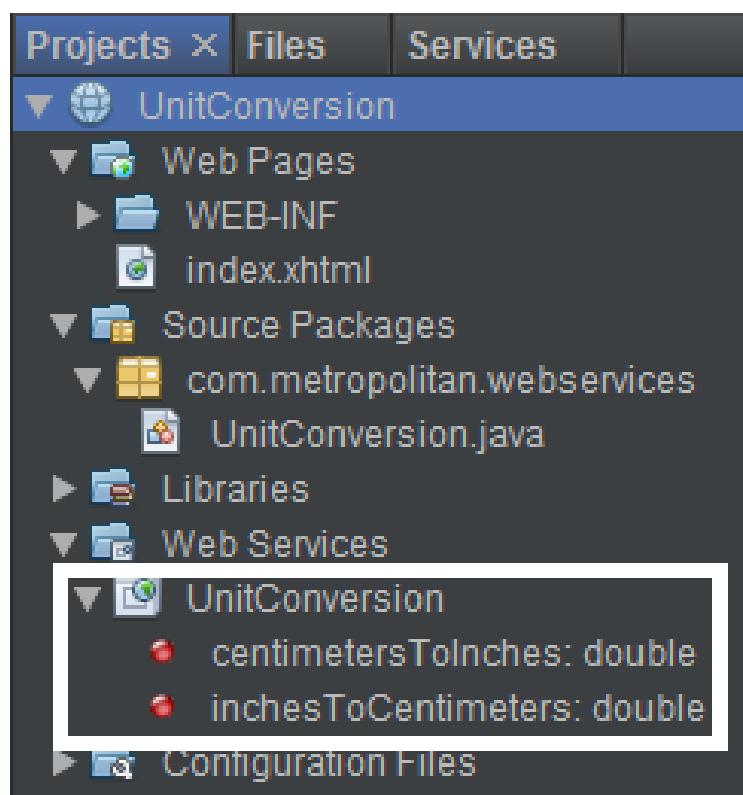
## ▼ Poglavlje 7

# Testiranje kreiranog veb servisa

## IZBOR OPCIJE ZA TESTIRANJE VEB SERVISA

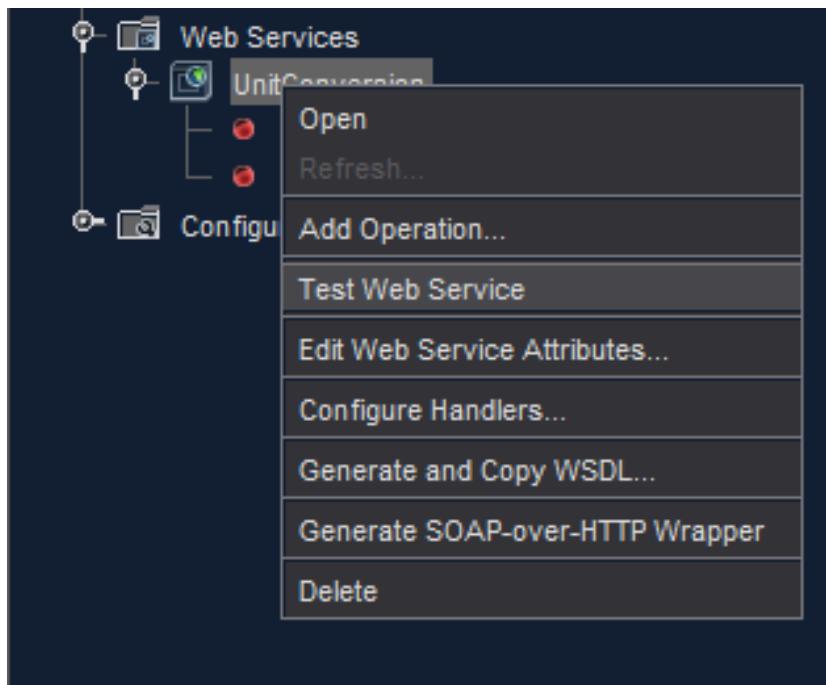
*U nastavku je neophodno pristupiti testiranju funkcionalnosti operacija kreiranog veb servisa.*

U nastavku je neophodno pristupiti testiranju funkcionalnosti operacija kreiranog veb servisa. Za početak je potrebno fokusirati se na folder Web Services iz hijerarhije kreiranog projekta koji je prikazan sledećom slikom.



Slika 7.1 Čvor Web Services u hijerarhiji projekta [izvor: autor]

Razvojno okruženje *NetBeans IDE*, uz asistenciju *GlassFish* aplikativnog servera, opet nudi značajno olakšanje prilikom izvođenja zadataka testiranja kreiranih veb servisa. Dovoljno je izvršiti desni klik u čvoru *Web Services* na podčvor *UnitConversion* i izabrati opciju *Test Web Service* (videti sledeću sliku). Ubrzo u veb pregledaču se otvara stranica za testiranje operacija kreiranog veb servisa.



Slika 7.2 Izbor opcije za testiranje veb servisa [izvor: autor]

## STRANICA ZA TESTIRANJE VEB SERVISA

*U veb pregledaču je neophodno posetiti stranicu za testiranje veb servisa.*

Ukoliko se prilikom testiranja u GlassFish logu javi greška "Failed to read schema document 'xjc.xsd', because 'bundle' access is not allowed due to restriction set by the accessExternalSchema property", neophodno je kreirati datoteku jaxp.properties i u nju dodati sledeći sadržaj: javax.xml.accessExternalSchema=all . Datoteku je nophodno sačuvati u folderu /jre/lib JDK foldera.

Kao što je pomenuto u prethodnom izlaganju, nakon izbora opcije *Test Web Service*, na prethodno opisani način, u veb pregledaču se otvara stranica za testiranje kreiranog veb servisa. Navedena stranica je ilustrovana sledećom slikom.



This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

**Methods :**

```
public abstract double com.metropolitan.webservices.UnitConversion.inchesToCentimeters(double)
inchesToCentimeters ( )
```

```
public abstract double com.metropolitan.webservices.UnitConversion.centimetersToInches(double)
centimetersToInches ( )
```

Slika 7.3 Stranica za testiranje veb servisa [izvor: autor]

Na ovoj stranici je moguće izvršiti testiranje kreiranih operacija veb servisa jednostavnim unošenjem **double** vrednosti u odgovarajuća polja za unos teksta i klikom na odgovarajuće dugme, kao na sledećoj slici.

#### Methods :

```
public abstract double com.metropolitan.webservices.UnitConversion.inchesToCentimeters(double)
inchesToCentimeters (5 )
```

Slika 7.4 Testiranje operacije inchesToCentimeters veb servisa [izvor: autor]

Klikom na dugme sa slike, otvara se nova stranica koja prikazuje rezultat konverzije jedinica, kao i "sirovi" **SOAP** zahtev i odgovor. Navedeno je prikazano sledećom slikom.



Method invocation

Method parameter(s)

Type	Value
double	5

Method returned

double : 1.968503937007874

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:inchesToCentimeters xmlns:ns2="http://webservices.metropolitan.com/">
<inches>5.0</inches>
</ns2:inchesToCentimeters>
</S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:inchesToCentimetersResponse xmlns:ns2="http://webservices.metropolitan.com/">
<return>1.968503937007874</return>
</ns2:inchesToCentimetersResponse>
</S:Body>
</S:Envelope>
```

Slika 7.5 Rezultat testiranja operacije veb servisa [izvor: autor]

## ▼ Poglavlje 8

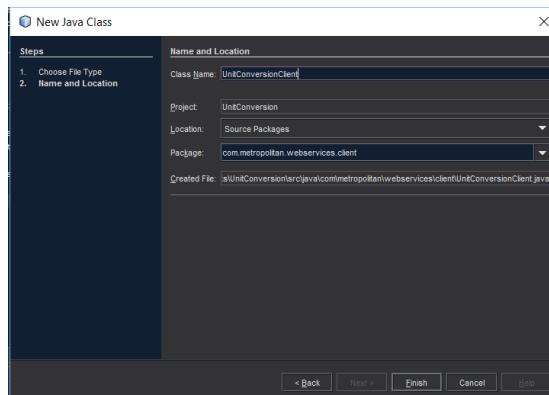
# Razvoj klijenta veb servisa

## KREIRANJE KLASE KLIJENTA VEB SERVISA

*Klijent veb servisa može biti bilo koji tip Java projekta.*

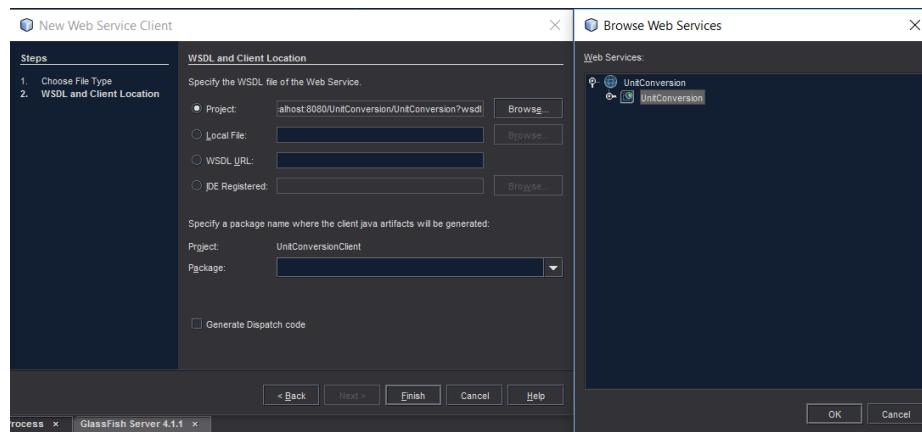
Nakon što je kreiran veb servis, sa odgovarajućim operacijama, i pošto je uspešno obavljeno testiranje funkcionalnosti operacija veb servisa, moguće je pristupiti kreiranju njegovih klijenata. U konkretnom slučaju, za potrebe analize i diskusije, biće kreiran jednostavan klijent koji će pozivati kreirani WEB servis.

Klijent veb servisa može biti bilo koji tip Java projekta, kao što je standardna Java aplikacija, Java ME aplikacija, veb aplikacija ili EE projekat. Budući da je u ovo delu izlaganja cilj pokazivanje pozivanja veb servisa, fokus će biti na jednostavnom klijentu koji će bit standardna Java aplikacija. Izborom opcija *File | NewProject* biće izvršen izbor kreiranja projekta standardne Java aplikacije. Naziv projekta će glasiti *UnitConverterClient*.



Slika 8.1 Kreiranje klase klijenta veb servisa [izvor: autor]

Nakon što je kreiran projekat klijenta veb servisa, moguće je pristupiti kreiranju klase klijenta veb servisa. Za prethodno kreirani projekat, bira se opcija *File | New File*, a potom se iz kategorije *Web Services* bira tip datoteke *Web Service Client*. Ubrzo se, u čarobnjaku, otvara prozor pod nazivom *New Web Service Client* u kojem se, izborom odgovarajućeg, radio dugmeta, i opcije *Browse*, bira veb servis, iz liste dostupnih servisa, kojeg će klijent pozivati. *URL* za generisani *WSDL* fajl će biti automatski dodat u odgovarajuće polje nakon obavljenog izbora servisa. Izborom paketa i klikom na dugme *Finish* vrši se kreiranje klijenata veb servisa. Navedeno je prikazano sledećom slikom.



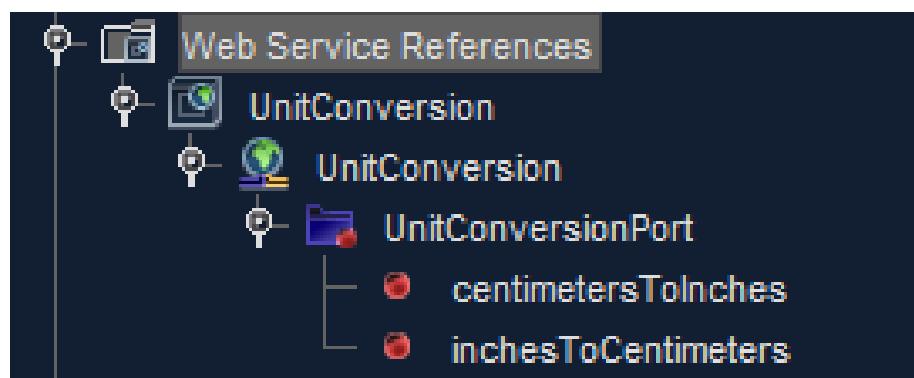
Slika 8.2 Podešavanje i kreiranje klase klijenta veb servisa [izvor: autor]

## GENERISANJE KODA KLIJENTA

*Moguće je primetiti da je razvojno okruženje automatski generisalo klijente veb servisa.*

Nakon obavljenog podešavanja, prikazanog prethodnom slikom, Moguće je primetiti da je razvojno okruženje automatski generisalo brojne datoteke veb servisa. Takođe, moguće je koristiti i klijente veb servisa u čijem kreiranju je učestvovao neko drugi, ili su prethodno kreirani, izborom opcije *Local File* za izbor *WSDL* datoteke sa hard diska. Takođe, moguće je koristiti opciju *WSDL URL* za izbor *WSDL* datoteke koja je objavljena na vebu. *NetBeans IDE* daje mogućnost primene i nekoliko javno objavljenih veb servisa. Za razvoj klijenata ovih servisa, u prozoru sa prethodne slike, neophodno je izabrati opciju *IDE Registered*.

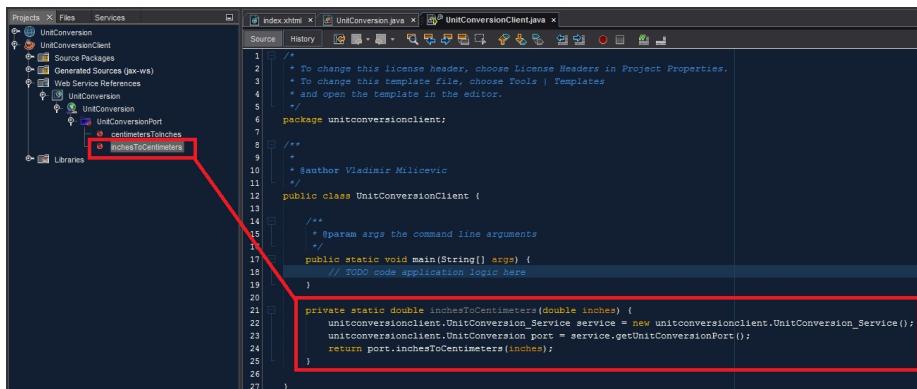
Sada je moguće posvetiti više pažnje kreiranim klijentima. U hijerarhiji projekta klijenata pojavio se nov čvor pod nazivom *Web Service References*. Navedeno je prikazano sledećom slikom.



Slika 8.3 Web Service References u projektu klijenta veb servisa [izvor: autor]

Kao što je već pomenuto, pisanje veb servisa, kao i odgovarajućih klijenata, praćeno je kreiranjem koda koji je veoma sklon greškama. Savremena razvojna okruženja, među njima i *NetBeans IDE*, značajno olakšavaju ovaj posao i dozvoljavaju da se jednostavno "prevuku" operacije veb servisa koje je neophodno imati u kodu. Na ovaj način će biti generisan sav potreban kod i posao se svodi na specifikaciju parametara koji će biti prosleđeni u veb servis.

Prevlačenjem operacije `inchesToCentimeters` iz prozora projekta u klasu `UnitConversionClient.java` (videti sledeću sliku) dolazi do automatskog generisanja koda u klasi koja je pre toga imala samo praznu metodu `main()`.



Slika 8.4 Pravljenje operacija u glavnu klasu projekta klijenta

## GENERISANI KOD KLIJENTA

*Neophodno je izvršiti analizu generisanog koda klase klijenta web servisa.*

Nakon prevlačenja operacije `inchesToCentimeters` iz prozora projekta u klasu `UnitConversionClient.java` (videti Sliku 4) dolazi do automatskog generisanja koda u klasi koja je pre toga imala samo praznu metodu `main()`. Sledećim listingom je priložen trenutni kod ove klase:

```

package unitconversionclient;

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

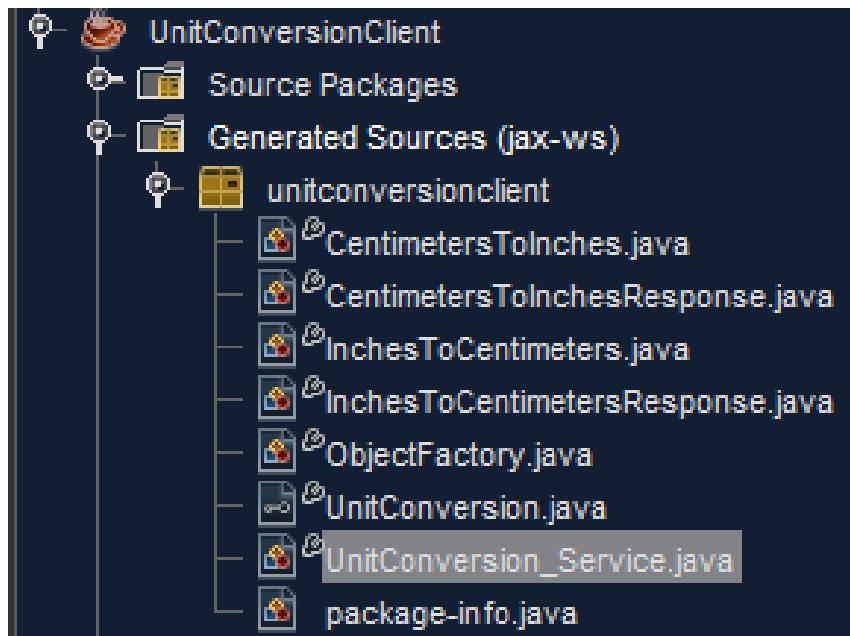
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

    private static double inchesToCentimeters(double inches) {
        unitconversionclient.UnitConversion_Service service = new
        unitconversionclient.UnitConversion_Service();
        unitconversionclient.UnitConversion port = service.getUnitConversionPort();
        return port.inchesToCentimeters(inches);
    }
}

```

U nastavku je neophodno izvršiti dodatnu analizu automatski generisanog koda metode *inchesToCentimeters()* klase *UnitConversionClient*.

Kao što je moguće primetiti iz priloženog primera klase klijenta veb servisa, generisan metoda, koja odgovara operaciji veb servisa *inchesToCentimeters()* poziva nekoliko metoda u klasi *UnitConversion\_Service.java*. Ova klasa, zajedno sa nekolicinom ostalih, automatski je generisana kada je operacija veb servisa "prevučena" u kod klase. Sve generisane klase je moguće pronaći kada se proširi čvor *Generated Sources* (jax-ws) u prozoru projekta (sledeća slika).



Slika 8.5 Generisane klase za prevučenu operaciju inchesToCentimeters() [izvor: autor]

## DOPUNA I TESTIRANJE

*Sledi dalja analiza generisano koda klase UnitConversionClient.java.*

Sledi dalja analiza generisano koda klase *UnitConversionClient.java*. Još jedna instrukcija je automatski generisana. Metoda *getUnitConversionPort()* klase *UnitConversion\_Service* vraća instancu klase *UnitConversion* koja je kreirana iz *WSDL* i slična je identično nazvanoj klasi koja je kreirana u okviru postojećeg projekta veb servisa. Metoda koja je generisana kada je prevučena konkretna operacija veb servisa u kod klase klijenta, poziva ovu metodu, a zatim poziva i metodu *inchesToCentimeters()* vraćene instance klase *UnitConversion*.

Za proveru uspešnosti kreiranog klijent koda neophodno je samo izvršiti još malu modifikaciju **main()** metode klase klijenta i snabdeti je pozivom generisane metode *inchesToCentimeters()* kao u sledećem listingu:

```
package unitconversionclient;

/**
 *
 * @author Vladimir Milicevic
```

```
/*
public class UnitConversionClient {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

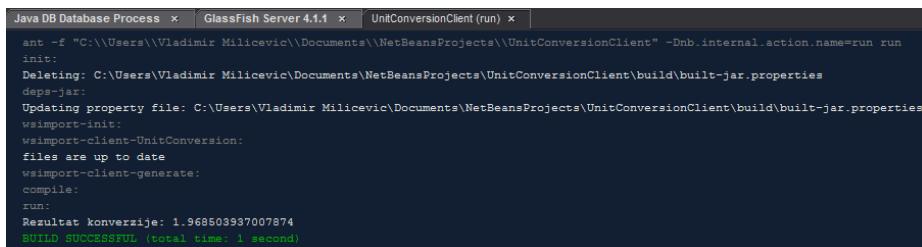
        System.out.println("Rezultat konverzije: " + inchesToCentimeters(5));

    }

    private static double inchesToCentimeters(double inches) {
        unitconversionclient.UnitConversion_Service service = new
unitconversionclient.UnitConversion_Service();
        unitconversionclient.UnitConversion port = service.getUnitConversionPort();
        return port.inchesToCentimeters(inches);
    }

}
```

Angažovanjem servisa na veb serveru i pokretanjem klijenta veb servisa, u razvojnog okruženju NetBeans IDE, na konzoli je moguće uočiti izlaz prikazan sledećom slikom.



```
Java DB Database Process x GlassFish Server 4.1.1 x UnitConversionClient(run) x
ant -f "C:\Users\Vladimir Milicevic\Documents\NetBeansProjects\UnitConversionClient" -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\Vladimir Milicevic\Documents\NetBeansProjects\UnitConversionClient\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\Vladimir Milicevic\Documents\NetBeansProjects\UnitConversionClient\build\built-jar.properties
wsimport-init:
wsimport-client-UnitConversion:
files are up to date
wsimport-client-generate:
compile:
run:
Rezultat konverzije: 12.700000
BUILD SUCCESSFUL (total time: 1 second)
```

Slika 8.6 Izvršavanje klijenta veb servisa [izvor: autor]

Na ovaj način je provereno da je sav prethodni posao obavljen na pravi način.

## DOPUNSKO RAZMATRANJE I GENERISANE POMOĆNE KLASE

*Neophodno je pozabaviti se automatski generisanom klasom [UnitConversion\\_Service](#).*

Kao što je istaknuto u prethodnom izlaganju, klasa [UnitConversion\\_Service.java](#), zajedno sa nekolicinom ostalih, automatski je generisana kada je operacija veb servisa "prevučena" u kod klase. Takođe je pokazano, da je sve generisane klase je moguće pronaći kada se proširi čvor [Generated Sources \(jax-ws\)](#) u prozoru projekta. Za ilustrovanje koliko je bila velika pomoć razvojnog okruženja [NetBeans IDE](#) prilikom razvoja ovog projekta biće priložen kompleksan kod ove kase koji je automatski generisan. Čak je i upravljanje anotacijama krajnjih tačaka veb servisa [@WebEndpoint\(name = "UnitConversionPort"\)](#) automatski

odrađeno kada je prevučena operacija *inchesToCentimeters()* web servisa u kod klase klijenta web servisa.

Sledećim listingom je priložen kod automatski generisane klase *UnitConversion\_Service.java*.

```
package unitconversionclient;

import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;
import javax.xml.ws.WebServiceException;
import javax.xml.ws.WebServiceFeature;

/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.6-1b01
 * Generated source version: 2.2
 *
 */
@WebServiceClient(name = "UnitConversion", targetNamespace =
"http://webservices.metropolitan.com/", wsdlLocation = "http://localhost:8080/
UnitConversion/UnitConversion?wsdl")
public class UnitConversion_Service
    extends Service
{

    private final static URL UNITCONVERSION_WSDL_LOCATION;
    private final static WebServiceException UNITCONVERSION_EXCEPTION;
    private final static QName UNITCONVERSION_QNAME = new
QName("http://webservices.metropolitan.com/", "UnitConversion");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:8080/UnitConversion/
UnitConversion?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        UNITCONVERSION_WSDL_LOCATION = url;
        UNITCONVERSION_EXCEPTION = e;
    }

    public UnitConversion_Service() {
        super(__getWsdlLocation(), UNITCONVERSION_QNAME);
    }

    public UnitConversion_Service(WebServiceFeature... features) {
```

```
        super(__getWsdlLocation(), UNITCONVERSION_QNAME, features);  
    }  
  
    public UnitConversion_Service(URL wsdlLocation) {  
        super(wsdlLocation, UNITCONVERSION_QNAME);  
    }  
  
    public UnitConversion_Service(URL wsdlLocation, WebServiceFeature... features) {  
        super(wsdlLocation, UNITCONVERSION_QNAME, features);  
    }  
  
    public UnitConversion_Service(URL wsdlLocation, QName serviceName) {  
        super(wsdlLocation, serviceName);  
    }  
  
    public UnitConversion_Service(URL wsdlLocation, QName serviceName,  
WebServiceFeature... features) {  
        super(wsdlLocation, serviceName, features);  
    }  
  
    /**  
     *  
     * @return  
     *      returns UnitConversion  
     */  
    @WebEndpoint(name = "UnitConversionPort")  
    public UnitConversion getUnitConversionPort() {  
        return super.getPort(new QName("http://webservices.metropolitan.com/",  
"UnitConversionPort"), UnitConversion.class);  
    }  
  
    /**  
     *  
     * @param features  
     *      A list of {@link javax.xml.ws.WebServiceFeature} to configure on the  
proxy. Supported features not in the <code>features</code> parameter will have  
their default values.  
     * @return  
     *      returns UnitConversion  
     */  
    @WebEndpoint(name = "UnitConversionPort")  
    public UnitConversion getUnitConversionPort(WebServiceFeature... features) {  
        return super.getPort(new QName("http://webservices.metropolitan.com/",  
"UnitConversionPort"), UnitConversion.class, features);  
    }  
  
    private static URL __getWsdlLocation() {  
        if (UNITCONVERSION_EXCEPTION!= null) {  
            throw UNITCONVERSION_EXCEPTION;  
        }  
        return UNITCONVERSION_WSDL_LOCATION;  
    }
```

}

## GENERATED SOURCES ČVOR

*Generisane klase je moguće pronaći kada se proširi čvor Generated Sources projekta klijenta.*

Kao što je već istaknuto, sve automatski generisane klase je moguće pronaći kada se proširi čvor *Generated Sources* (*jax-ws*) u prozoru projekta. Za ilustrovanje koliko je bila velika pomoć razvojnog okruženja NetBeans IDE prilikom razvoja ovog projekta biće priložen kompleksan kod još par klasa koje je automatski generisalo razvojno okruženje prilikom kreiranja projekta klijenta web servisa.

Sledi listing pomoćne klase *CentimetersToInches*:

```
package unitconversionclient;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>Java class for centimetersToInches complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * <complexType name="centimetersToInches">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         <element name="centimeters" type="{http://www.w3.org/2001/
XMLSchema}double"/>
 *       </sequence>
 *     </restriction>
 *   </complexContent>
 * </complexType>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "centimetersToInches", propOrder = {
    "centimeters"
})
public class CentimetersToInches {
```

```
protected double centimeters;

/**
 * Gets the value of the centimeters property.
 *
 */
public double getCentimeters() {
    return centimeters;
}

/**
 * Sets the value of the centimeters property.
 *
 */
public void setCentimeters(double value) {
    this.centimeters = value;
}

}
```

Sledi listing pomoćne klase *InchesToCentimeters*:

```
package unitconversionclient;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>Java class for inchesToCentimeters complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
*
* <pre>
* <complexType name="inchesToCentimeters">
*   <complexContent>
*     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
*       <sequence>
*         <element name="inches" type="{http://www.w3.org/2001/XMLSchema}double"/>
*       </sequence>
*     </restriction>
*   </complexContent>
* </complexType>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "inchesToCentimeters", propOrder = {
    "inches"
})
```

```
public class InchesToCentimeters {

    protected double inches;

    /**
     * Gets the value of the inches property.
     *
     */
    public double getInches() {
        return inches;
    }

    /**
     * Sets the value of the inches property.
     *
     */
    public void setInches(double value) {
        this.inches = value;
    }

}
```

## RESPONSE DATOTEKE

*Posebno je moguće uočiti generisane klase sa sufiksom Response.*

Među generisanim klasama ističu se i dve klase koje poseduju u nazivu sufiks *Response*.

Sledi listing klase *InchesToCentimetersResponse*:

```
package unitconversionclient;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>Java class for inchesToCentimetersResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
 * this class.
 *
 * <pre>
 * <complexType name="inchesToCentimetersResponse">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         <element name="return" type="{http://www.w3.org/2001/XMLSchema}double"/>
 *       </sequence>
 *     </restriction>
 *   </complexContent>
 * </complexType>
 * </pre>
 * 
 * This class was generated by JAX-B (JAXB) 2.3.2-b6-b07f2d43.
 * See http://java.sun.com/xml/jaxb
 * 
 */
```

```
*      </restriction>
*    </complexContent>
* </complexType>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "inchesToCentimetersResponse", propOrder = {
    "_return"
})
public class InchesToCentimetersResponse {

    @XmlElement(name = "return")
    protected double _return;

    /**
     * Gets the value of the return property.
     *
     */
    public double getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     */
    public void setReturn(double value) {
        this._return = value;
    }

}
```

Sledi listing klase *CentimetersToInchesResponse*:

```
package unitconversionclient;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>Java class for centimetersToInchesResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
*
* <pre>
* <complexType name="centimetersToInchesResponse">
*   <complexContent>
```

```
*      <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
*          <sequence>
*              <element name="return" type="{http://www.w3.org/2001/XMLSchema}double"/>
*          </sequence>
*      </restriction>
*  </complexContent>
* </complexType>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "centimetersToInchesResponse", propOrder = {
    "_return"
})
public class CentimetersToInchesResponse {

    @XmlElement(name = "return")
    protected double _return;

    /**
     * Gets the value of the return property.
     *
     */
    public double getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     */
    public void setReturn(double value) {
        this._return = value;
    }

}
```

## VIDEO MATERIJAL : CREATE SOAP WEB SERVICE AND ITS CLIENT

*Sledi video materijal za podršku analizi i diskusiji u vezi sa klijentima SOAP servisa.*

Create *SOAP web service* and its client

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

## ▼ Poglavlje 9

# Implementiranje novog veb servisa kao EJB

## EJB KAO VEB SERVIS

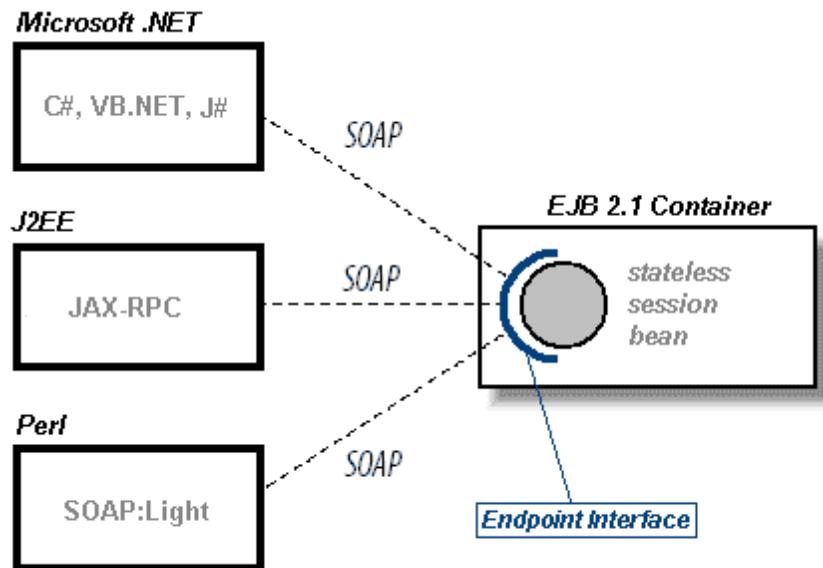
*Moguće je iskoristiti zrna sesije bez stanja (stateless beans) kao veb servise.*

U prethodnom izlaganju kao primer upotrebe veb servisa istaknut je primer za konverziju jedinica dužine u kojem je iskorišćen **POJO** (**Plain Old Java Object**) objekat kao veb servis. **Veb servis** je spakovan u veb aplikaciju i dodeljeno mu je par odgovarajućih anotacija.

Kada se radi sa **EJB** modulom, kao projektom, moguće je iskoristiti **zrna sesije bezstanja (stateless session beans)** kao veb servise. Na ovaj način je moguće pristupiti kreiranim veb servisima pomoću klijenata koji su kreirani u programskim jezicima koji su različiti od programskog jezika Java (pogledati Sliku 1). Uvođenjem zrna sesije bez stanja, kao veb servisa, omogućava kreiranim veb servisima da u potpunosti iskoriste sve prednosti primene EJB zrna, kao što su: *upravljanje transakcijama i aspektno - orijentisano programiranje*.

Postoje dva načina da pomoći kojih se zrna sesije definišu kao veb servisi. Kada se kreira nov veb servis u projektu EJB modula, veb servis će automatski biti implementiran kao zrno sesije bez stanja (**stateless session beans**). Takođe, i postojeća zrna sesije u postojećem projektu EJB modula, mogu biti deklarisana kao veb servisi.

Sledećom slikom je prikazan scenario po kojem se EJB zrna sesije koriste kao veb servisi.



Slika 9.1 EJB zrna sesije kao veb servisi. [izvor: <https://www.theserverside.com>]

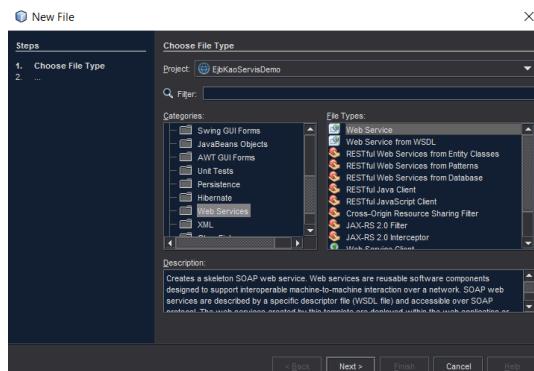
## KREIRANJE NOVOG VEB SERVISA

*Kreira se nov veb servis kao zrno sesije bez stanja.*

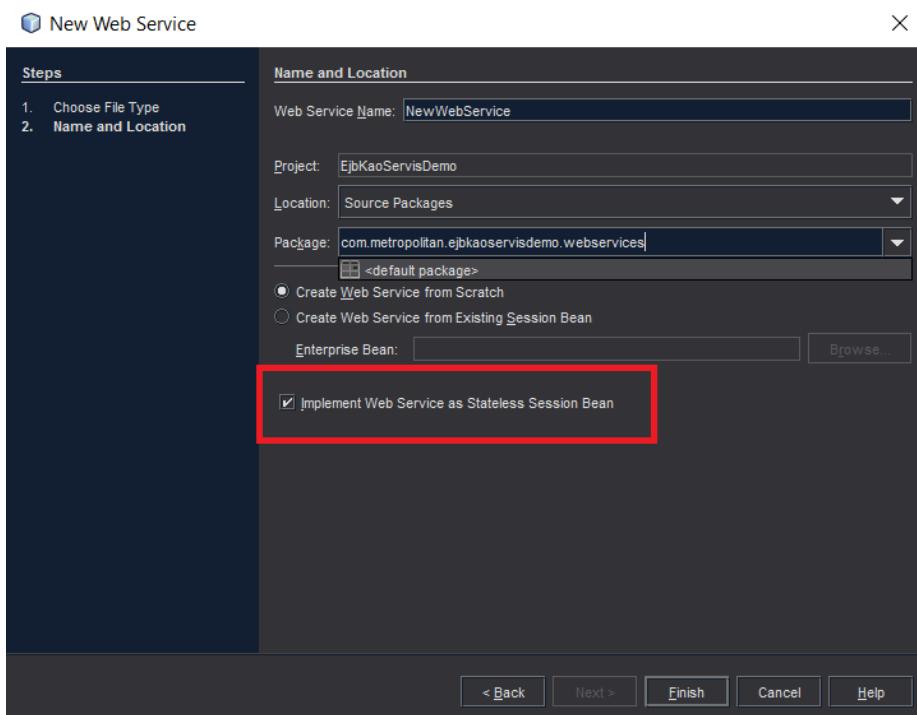
Nakon uvodnog razmatranja moguće je pristupiti analizi i diskusiji problematike koja je u vezi sa kreiranjem veb servisa u formi **EJB** zrna. Veb servis se jednostavno kreira u projektu **EJB** modula ili projektu veb aplikacije.

Kada se koristi projekat veb aplikacije za kreiranje SOAP veb servisa, data je mogućnost da se veb servis kreira kao POJO objekat ili zrno sesije bez stanja. Kada se koristi EJB modul za kreiranje SOAP veb servisa, veb servis je moguće kreirati isključivo kao zrno sesije bez stanja.

Izborom, u kreiranom projektu, opcije **File | New File**, a potom iz kategorije **Web Services** bira tip datoteke **Web Service** (Slika 2), otvara se prozor pod nazivom **New Web Service** u kojem se unose detalji podešavanja veb servisa koji se kreira (Slika3).



Slika 9.2 Kreiranje datoteke veb servisa [izvor: autor]



Slika 9.3 Inicijalno podešavanja datoteke veb servisa [izvor: autor]

Ako se koristi EJB zrno bez stanja kao veb servis u projektu Java veb aplikacije, u prozoru prikazanom na Slici 2, neophodno je čekirati opciju Implement Web Service as Stateless Session Bean.

## KREIRANJE KLASE VEB SERVISA

*Nakon kreiranja projekta za EJB veb servise, neophodno je kreirati klasu veb servisa.*

Nakon kreiranja projekta za EJB veb servise, neophodno je kreirati klasu veb servisa. Klasa veb servisa će nositi naziv `VolumeUnitConversion.java` i moraće da bude obeležena odgovarajućom anotacijom `@WebService(serviceName = "VolumeUnitConversion")`. Inicijalni listing generisane klase `VolumeUnitConversion` priložen je u narednom izlaganju:

```
package com.metropolitan.ejbkaoservisdemo.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
```

```

 * @author Vladimir Milicevic
 */
@WebService(serviceName = "VolumeUnitConversion")
@Stateless()
public class VolumeUnitConversion {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}

```

Ono što je moguće primetiti, u nastavku, jeste to da je klasa obeležena i anotacijom `@Stateless()` koja ukazuje da klasa predstavlja *zrno sesije bez stanja*. Klasa ima predefinisanu jednu metodu koja je obeležena anotacijom `@WebMethod` čiji parametri moraju biti obeleženi anotacijama `@WebParam`. Ono što je lako uočiti jeste razlika između ove klase veb servisa i one koja je analizirana i diskutovana u prethodnom izlaganju. Jednina razlika između navedenih klasa je ta da je klasa `VolumeUnitConversion` implementirana kao zrno sesije bez stanja i da, kao takva, može u potpunosti da koristi prednosti koje su obezbeđene primenom *EJB* zrna. Ove prednosti mogu biti upravljanje transakcijama, aspektno - orijentisano programiranje, kao i brojne druge EJB pogodnosti.

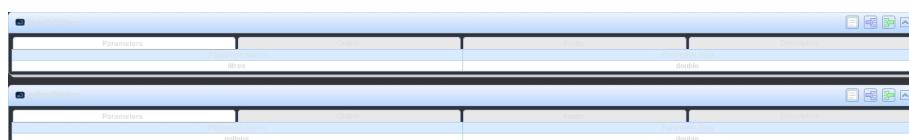
## KREIRANJE OPERACIJA EJB VEB SERVISA

*Predefinisane operacije klase EJB veb servisa neophodno je zameniti konkretnim.*

Kao i kod regularni veb servisa, operacije veb servisa kreiranih kao *zrna sesije* bez stanja mogu biti generisane primenom vizuelnog *dizajnera veb servisa* koji je obezbeđen razvojnim okruženjem *NetBeans IDE*. Upravo u navedenom svetu biće pokrenut *dizajner veb servisa* u kojem će biti obrisana predefinisana operacija kreiranog veb servisa. Izborom opcije *Design* (opisano u prethodnom izlaganju) otvara se prozor koji odgovara navedenom dizajneru veb servisa. Za predefinisanu operaciju bira se opcija *Remove Operation* i ona se briše iz veb servisa. U nastavku će biti kreirane željene operacije *EJB veb servisa*. Bira se opcija *Add Operation*. Na ova način će biti kreirane dve nove operacije:

- *gallonsToLitres()* - konvertuje vrednost u galonima u vrednost u litrima;
- *litresToGallons()* - konvertuje vrednosti u galonima u vrednost u litrima.

Obe metode će preuzeti parametre tipa double, gallons i litres, respektivno.



Slika 9.4 Kreirane nove operacije EJB veb servisa [izvor: autor]

Klikom na dugme *Source*, prelazi se iz pogleda *dizajnera veb servisa* u pogled editora koda i moguće je primetiti da je kod koji odgovara novim operacijama *EJB veb servisa* generisan u kreiranoj klasi veb servisa. Sledi novi listing kreirane klase veb servisa.

```
package com.metropolitan.ejbkaoservisdemo.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "VolumeUnitConversion")
@Stateless()
public class VolumeUnitConversion {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "litresToGallons")
    public double litresToGallons(@WebParam(name = "litres") double litres) {
        //TODO write your implementation code here:
        return 0.0;
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "gallonsToLitres")
    public double gallonsToLitres(@WebParam(name = "gallons") double gallons) {
        //TODO write your implementation code here:
        return 0.0;
    }
}
```

## REDEFINISANJE KREIRANIH METODA EJB VEB SERVISA

*Kreirane metode EJB veb servisa je neophodno redefinisati tako da vrate željene vrednosti.*

U prethodnom izlaganju je pokazano kao se kreiraju željene operacije za konkretni veb servis izgrađen kao *zrno sesije bez stanja*. U nastavku je neophodno dodati, u generisane metode, konkretnе izraze kojima se vrši konverzija između navedenih veličina.

Sledećim listingom je priložen konačan kod klase *EJB veb servisa* pod nazivom *VolumeUnitConversion.java*.

```
package com.metropolitan.ejbkaoservisdemo.webservices;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "VolumeUnitConversion")
@Stateless()
public class VolumeUnitConversion {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "litresToGallons")
    public double litresToGallons(@WebParam(name = "litres") double litres) {
        //TODO write your implementation code here:
        return (litres * 0.26417);
    }

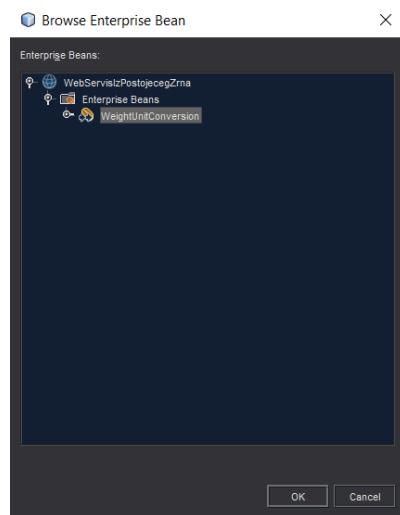
    /**
     * Web service operation
     */
    @WebMethod(operationName = "gallonsToLitres")
    public double gallonsToLitres(@WebParam(name = "gallons") double gallons) {
        //TODO write your implementation code here:
        return gallons * 3.7854;
    }
}
```

## IMPLEMENTIRANJE POSTOJEĆEG EJB KAO VEB SERVISA

*U nastavku je neophodno pokazati drugačiji pristup - uvođenje postojećeg EJB zrna kao veb servis*

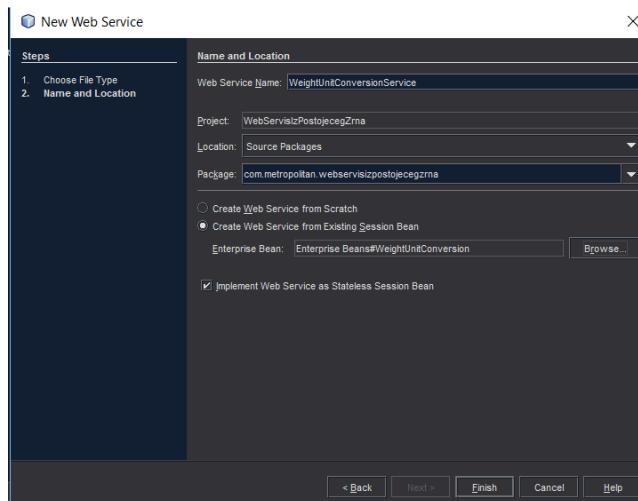
U nastavku je neophodno pokazati drugačiji pristup - uvođenje postojećeg *EJB zrna* kao veb servisa. Za potrebe analize i dizajna biće kreiran nov projekat pod nazivom *WebServisIzPostojećegZrna*. Zatim će prateći izbor *File | New | Web Service* biti kreiran veb servis na način koji je već prikazan u prethodnom izlaganju. U odgovarajućem prozoru (sledeća slika) biće izabrana opcija *Create Web Service from Existing Session Bean*, kao i obavezna opcija *Implement Web Service As Stateless Session Bean*. Neophodno je definisati

naziv i paket za klasu EJB web servisa koji se kreira, a potom i izabratи odgovarajuћe, postojeће zrno, iz kojeg se kreira web servis (izborom opcije *Browse* - slika 5).

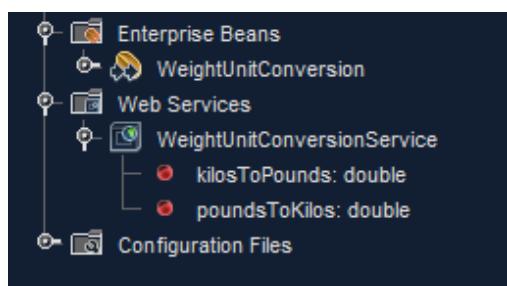


Slika 9.5 Izbor postojećeg EJB zrna [izvor: autor]

Nakon toga je potrebno dovršiti definiciju web servisa koji se kreira iz postojećeg *EJB* zrna klikom na *Finish* (videti sledeći prozor).



Slika 9.6 Dovršavanje definicije servisa iz postojećeg EJB zrna [izvor: autor]



Slika 9.7 Kreirani servis u strukturi projekta. [izvor: autor]

## LISTING KLASE KREIRANOOG VEB SERVISA

*Prezentacija generisanog koda veb servisa kreiranog iz postojećeg EJB zrna.*

U prethodnom izlaganju je pokazano kao se kreiraju željene operacije za konkretni veb servis izgrađen kao *zrno sesije bez stanja* iz postojećeg EJB zrna. U nastavku je neophodno pokazati da su operacije kreiranog veb servisa, u potpunosti odgovarajuće metodama EJB zrna iz kojeg je kreiran veb servis.

Sledećim listingom je priložen konačan kod klase *EJB veb servisa* pod nazivom *WeigthUnitConversion.java*.

```
package com.metropolitan.webservisizpostojecegznra;

import com.ensode.ejb.WeightUnitConversion;
import javax.ejb.EJB;
import javax.jws.WebService;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "WeightUnitConversionService")
@Stateless()
public class WeightUnitConversionService {

    @EJB
    private WeightUnitConversion ejbRef;// Add business logic below. (Right-click
in editor and choose
    // "Insert Code > Add Web Service Operation")

    @WebMethod(operationName = "kilosToPounds")
    public double kilosToPounds(@WebParam(name = "kilos") double kilos) {
        return ejbRef.kilosToPounds(kilos);
    }

    @WebMethod(operationName = "poundsToKilos")
    public double poundsToKilos(@WebParam(name = "pounds") double pounds) {
        return ejbRef.poundsToKilos(pounds);
    }
}
```

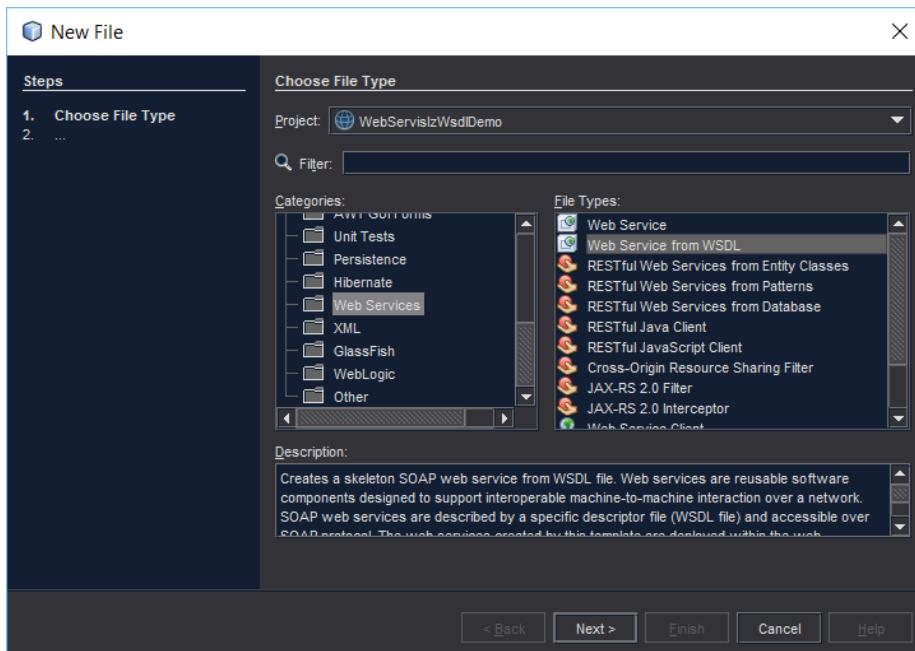
# KREIRANJE VEB SERVISA IZ POSTOJEĆEG WSDL

## *Kreiranje SOAP veb servisa zahteva kreiranje WSDL datoteke.*

Kao što je već više puta istaknuto, u prethodnim izlaganjima, kreiranje *SOAP veb servisa* zahteva kreiranje *WSDL* datoteke. Proces kreiranja *WSDL* datoteke je veoma kompleksan i podložan greškama. Međutim, savremena razvojna okruženja, posebno *NetBeans IDE*, u kombinaciji sa *JavaEE 7* platformom, daju značajnu podršku kojom je ovaj proces značajno olakšan i automatizovan.

Ako se zamisli scenario po kojem je dostupan *WSDL* fajl i neophodno je generisati njegove operacije primenom Java koda. Za ovaj scenario, *NetBeans IDE* daje veoma efikasan i jednostavan čarobnjak pomoću kojeg je moguće izvršiti kreiranje Java klasa sa metodama koje odgovaraju *WSDL* datoteci.

Za analizu i diskusiju ovakvog vida problematike biće kreiran nov Java veb projekat pod nazivom *WebServisIzWsdlDemo*. Za ovaj projekat će biti kreiran konkretni fajl iz kategorije *Web Services*, tipa *Web Service from WSDL*. Navedeno je prikazano sledećom slikom.



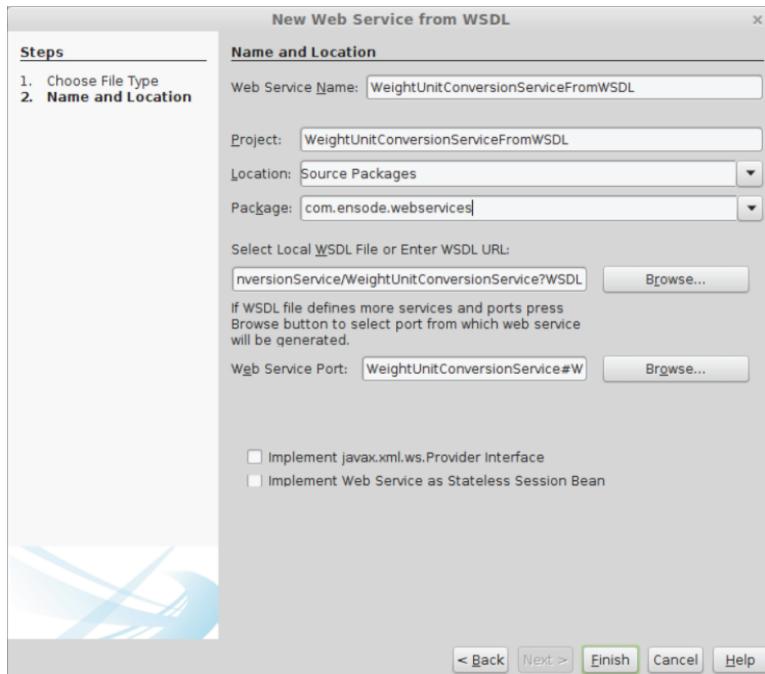
Slika 9.8 Kreiranje veb servisa iz postojećeg WSDL [izvor: autor]

U nastavku je neophodno izvršiti podešavanje datoteke koja se kreira primenom sledećeg prozora NetBeans IDE čarobnjaka.

# PODEŠAVANJE KLASE VEB SERVISA IZ POSTOJEĆEG WSDL

*U nastavku je potrebno izvršiti podešavanje klase veb servisa generisane iz postojećeg WSDL.*

Kao što je istaknuto u prethodnom izlaganju, u nastavku je neophodno izvršiti podešavanje datoteke koja se kreira primenom sledećeg prozora *NetBeans IDE* čarobnjaka. Klikom na dugme **Next**, u prozoru koji je prikazan Slikom 9, otvara se prozor u kojem se unosi: naziv web servisa i paket kojem će pripadati i, konačno, bira se lokacija WSDL datoteke iz koje se kreira web servis. Navedeno je prikazano sledećom slikom.



Slika 9.9 Podešavanje klase web servisa iz postojećeg WSDL [izvor: autor]

Klikom na dugme *Finish*, automatski se generiše kod klase web servisa iz postojećeg *WSDL* zajedno sa pripadajućim metodama. Kod generisane klase je priložen sledećim listingom.

```
package com.ensode.webservices;

import javax.jws.WebService;

/**
 *
 * @author Vladimir Milicevic
 */
@WebService(serviceName = "WeightUnitConversionService",
        portName = "WeightUnitConversionServicePort",
        endpointInterface = "com.ensode.webservices.WeightUnitConversionService",
        targetNamespace = "http://webservices.ensode.com/",
        wsdlLocation
        = "WEB-INF/wsdl/WeightUnitConversionServiceFromWSDL/localhost_8080/
WeightUnitConversionService/WeightUnitConversionService.wsdl")
public class WeightUnitConversionServiceFromWSDL {

    public double kilosToPounds(double kilos) {
        //TODO implement this method
        throw new UnsupportedOperationException("Not implemented yet.");
    }
}
```

```
public double poundsToKilos(double pounds) {  
    //TODO implement this method  
    throw new UnsupportedOperationException("Not implemented yet.");  
}  
}
```

## VIDEO MATERIJAL : INTRODUCING SOAP AND JAX-WS

*Sledi rezime teorijskog dela lekcije lekcije izabranim video materijalom.*

Introducing [SOAP](#) and [JAX-WS](#):

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

## ▼ Poglavlje 10

# Pokazni primeri - Distribuirani veb servisi

## SADRŽAJ

*Cilj vežbi je praktično utvrđivanje znanja stečenog na teorijskim OU.*

Vežbe će biti podeljene na izučavanje konkretnih primera iz oblasti:

- RESTful veb servisi sa JAX - RS
- SOAP veb servisi sa JAX - RS.

## ▼ 10.1 RESTful veb servisi sa JAX - RS (25 min)

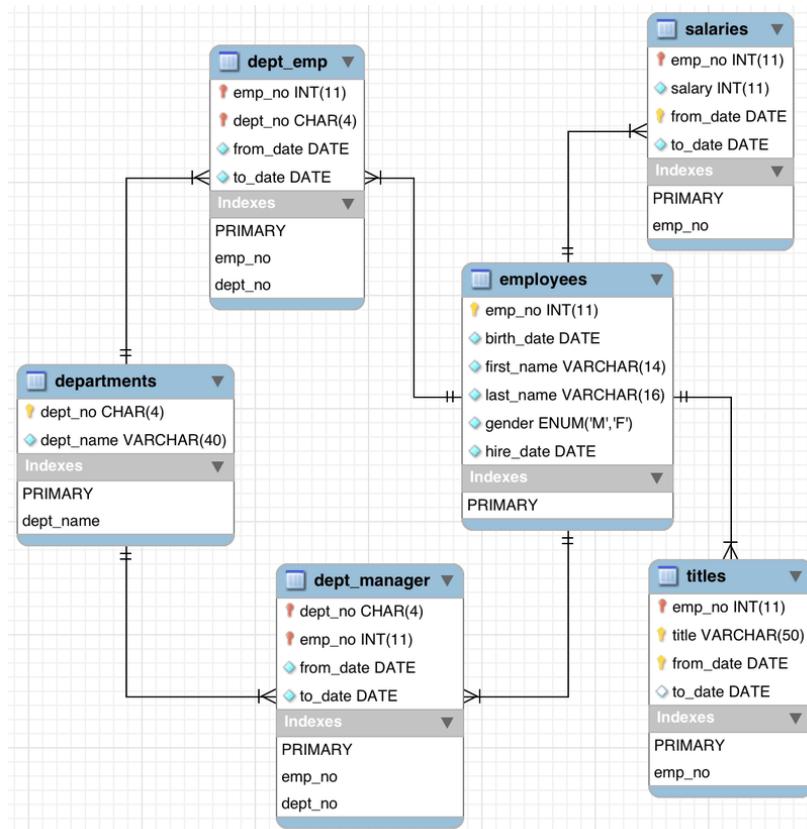
## POSTAVKA

*Vežbanje izrade kompletne distribuirane aplikacije sa RESTful veb servisima sa JAX - RS*

*Zadatak:*

Dovršiti primer sa predavanja. Nad svim tabelama:

1. kreirati RESTful servise;
2. kreirati Java klijente RESTful servisa;
3. kreirati JavaScript klijente RESTful servisa;
4. za tabele za koje postoji unosi izvršiti testiranje;
5. Šema baze podataka aplikacije je data Slikom 1;



Slika 10.1.1 Šema baze podataka [izvor: autor]

## TABELA "DEPARTMENTS"

*Kreiraju se tražene datoteke nad tabelom "departments".*

Sledi listing RESTful servisa DepartmentsFacadeREST.

```

package com.metropolitan.restdemo.service;

import com.metropolitan.restdemo.Departments;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

/**

```

```
*  
* @author Vladimir Milicevic  
*/  
@Stateless  
@Path("com.metropolitan.restdemo.departments")  
public class DepartmentsFacadeREST extends AbstractFacade<Departments> {  
  
    @PersistenceContext(unitName = "RESTfulDemoPU")  
    private EntityManager em;  
  
    public DepartmentsFacadeREST() {  
        super(Departments.class);  
    }  
  
    @POST  
    @Override  
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public void create(Departments entity) {  
        super.create(entity);  
    }  
  
    @PUT  
    @Path("{id}")  
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public void edit(@PathParam("id") String id, Departments entity) {  
        super.edit(entity);  
    }  
  
    @DELETE  
    @Path("{id}")  
    public void remove(@PathParam("id") String id) {  
        super.remove(super.find(id));  
    }  
  
    @GET  
    @Path("{id}")  
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public Departments find(@PathParam("id") String id) {  
        return super.find(id);  
    }  
  
    @GET  
    @Override  
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public List<Departments> findAll() {  
        return super.findAll();  
    }  
  
    @GET  
    @Path("{from}/{to}")  
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public List<Departments> findRange(@PathParam("from") Integer from,  
    @PathParam("to") Integer to) {
```

```
        return super.findRange(new int[]{from, to});
    }

    @GET
    @Path("count")
    @Produces(MediaType.TEXT_PLAIN)
    public String countREST() {
        return String.valueOf(super.count());
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

}
```

Sledi listing Java klijenta RESTful servisa .

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.metropolitan.restdemo.jaxrs;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;

/**
 * Jersey REST client generated for REST resource:DepartmentsFacadeREST
 * [com.metropolitan.restdemo.departments]<br/>
 * USAGE:
 * <pre>
 *     DepartmetsClient client = new DepartmetsClient();
 *     Object response = client.XXX(...);
 *     // do whatever with response
 *     client.close();
 *
 *
 *     @author Vladimir Milicevic
 * /
public class DepartmetsClient {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/RESTfulDemo/
webresources";

    public DepartmetsClient() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget =

```

```
client.target(BASE_URI).path("com.metropolitan.restdemo.departments");
}

public String countREST() throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("count");
    return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
}

public void edit_XML(Object requestEntity, String id) throws
ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_XML).put(javax.ws.rs.c
lient.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
}

public void edit_JSON(Object requestEntity, String id) throws
ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(javax.ws.rs.s
client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
}

public <T> T find_XML(Class<T> responseType, String id) throws
ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

public <T> T find_JSON(Class<T> responseType, String id) throws
ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public <T> T findRange_XML(Class<T> responseType, String from, String to)
throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

public <T> T findRange_JSON(Class<T> responseType, String from, String to)
throws ClientErrorException {
```

```

        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
    }

    public void create_XML(Object requestEntity) throws ClientErrorException {

webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
}

    public void create_JSON(Object requestEntity) throws ClientErrorException {

webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
}

    public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {
        WebTarget resource = webTarget;
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

    public <T> T findAll_JSON(Class<T> responseType) throws ClientErrorException {
        WebTarget resource = webTarget;
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

    public void remove(String id) throws ClientErrorException {
        webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request().delete();
}

    public void close() {
        client.close();
}
}

```

Sledi listing JavaScript klijenta RESTful servisa .

```

var app = {
    // Create this closure to contain the cached modules
    module: function () {
        // Internal module cache.
        var modules = {};

        // Create a new module reference scaffold or load an
        // existing module.
        return function (name) {

```

```
// If this module has already been created, return it.  
if (modules[name]) {  
    return modules[name];  
}  
  
// Create a module and save it under this name  
return modules[name] = {Views: {}};  
};  
}()  
};  
  
(function (models) {  
  
// Model for Departments entity  
models.Departments = Backbone.Model.extend({  
    urlRoot: "http://localhost:8080/RESTfulDemo/webresources/  
com.metropolitan.restdemo.departments/",  
    idAttribute: 'deptNo',  
    defaults: {  
        deptName: ""  
    },  
    toViewJson: function () {  
        var result = this.toJSON(); // displayName property is used to render  
item in the list  
        result.displayName = this.get('deptNo');  
        return result;  
    },  
    isNew: function () {  
        // default isNew() method implementation is  
        // based on the 'id' initialization which  
        // sometimes is required to be initialized.  
        // So isNew() is rediefined here  
        return this.notSynced;  
    },  
    sync: function (method, model, options) {  
        options || (options = {});  
        var errorHandler = {  
            error: function (jqXHR, textStatus, errorThrown) {  
                // TODO: put your error handling code here  
                // If you use the JS client from the different domain  
                // (f.e. locally) then Cross-origin resource sharing  
                // headers has to be set on the REST server side.  
                // Otherwise the JS client has to be copied into the  
                // some (f.e. the same) Web project on the same domain  
                alert('Unable to fulfil the request');  
            }  
        };  
  
        if (method === 'create') {  
            options.url = 'http://localhost:8080/RESTfulDemo/webresources/  
com.metropolitan.restdemo.departments/';  
        }  
        var result = Backbone.sync(method, model, _.extend(options,
```

```
errorHandler));
        return result;
    }

});

// Collection class for Departments entities
models.DepartmentsCollection = Backbone.Collection.extend({
    model: models.Departments,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.departments/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        };
        var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',
        initialize: function (options) {
            this.options = options || {};
            this.model.bind("reset", this.render, this);
            var self = this;
            this.model.bind("add", function (modelName) {
                var row = new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el;
                $(self.el).append($(row));
                $(self.el).parent().trigger('addRows', [$(row)]);
            });
        },
    },
}
```

```
render: function (eventName) {
    var self = this;
    _.each(this.model.models, function (modelName) {
        $(this.el).append(new views.ListItemView({
            model: modelName,
            templateName: self.options.templateName
        }).render().el);
    }, this);
    return this;
}

views.ListItemView = Backbone.View.extend({
    tagName: 'tr',

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    close: function () {
        var table = $(this.el).parent().parent();
        table.trigger('disable.pager');
        $(this.el).unbind();
        $(this.el).remove();
        table.trigger('enable.pager');
    }
});

views.ModelView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
    },
}
```

```
render: function (eventName) {
    $(this.el).html(this.template(this.model.toJSON()));
    return this;
},

template: function (json) {
    /*
     *  templateName is element identifier in HTML
     *  $(this.options.templateName) is element access to the element
     *  using jQuery
     */
    return _.template($(this.options.templateName).html())(json);
},

/*
 * Classes "save" and "delete" are used on the HTML controls to listen
events.
 * So it is supposed that HTML has controls with these classes.
*/
events: {
    "change input": "change",
    "click .save": "save",
    "click .delete": "drop"
},

change: function (event) {
    var target = event.target;
    console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
},

save: function () {
    // TODO : put save code here
    var hash = this.options.getHashObject();
    this.model.set(hash);
    if (this.model.isNew() && this.collection) {
        var self = this;
        this.collection.create(this.model, {
            success: function () {
                // see isNew() method implementation in the model
                self.model.notSynced = false;
                self.options.navigate(self.model.id);
            }
        });
    } else {
        this.model.save();
        this.model.el.parent().parent().trigger("update");
    }
    return false;
},

drop: function () {
    this.model.destroy({
```

```
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
             */
            window.history.back();
        }
    });
    return false;
},

close: function () {
    $(this.el).unbind();
    $(this.el).empty();
}
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },

    render: function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    /*
     * Class "new" is used on the control to listen events.
     * So it is supposed that HTML has a control with "new" class.
     */
    events: {
        "click .new": "create"
    },

    create: function (event) {
        this.options.navigate();
        return false;
    }
});
```

```
)})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
        routes: {
            '' : 'list',
            'new' : 'create'
            ,
            ':id' : 'details'
        },
        initialize: function () {
            var self = this;
            $('#create').html(new views.CreateView({
                // tpl-create is template identifier for 'create' block
                templateName: '#tpl-create',
                navigate: function () {
                    self.navigate('new', true);
                }
            }).render().el);
        },
        list: function () {
            this.collection = new models.DepartmentsCollection();
            var self = this;
            this.collection.fetch({
                success: function () {
                    self.listView = new views.ListView({
                        model: self.collection,
                        // tpl-departments-list-item is template identifier for item
                        templateName: '#tpl-departments-list-item'
                    });
                }
            });

            $('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()));
        });
        if (self.requestedId) {
            self.details(self.requestedId);
        }
        var pagerOptions = {
            // target the pager markup
            container: $('.pager'),
            // output string - default is '{page}/{totalPages}';
            possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
            output: '{startRow} to {endRow} ({totalRows})',
            // starting page of the pager (zero based index)
            page: 0,
            // Number of visible rows - default is 10
            size: 10
        };
        $('#datatable').tablesorter({widthFixed: true,
            widgets: ['zebra']}).
    });
});
```

```

        tablesorterPager(pagerOptions);
    }
});

},
details: function (id) {
    if (this.collection) {
        this.departments = this.collection.get(id);
        if (this.view) {
            this.view.close();
        }
        var self = this;
        this.view = new views.ModelView({
            model: this.departments,
            // tpl-departments-details is template identifier for chosen
model element
            templateName: '#tpl-departments-details',
            getHashObject: function () {
                return self.getData();
            }
        });
        $('#details').html(this.view.render().el);
    } else {
        this.requestedId = id;
        this.list();
    }
},
create: function () {
    if (this.view) {
        this.view.close();
    }
    var self = this;
    var dataModel = new models.Departments();
    // see isNew() method implementation in the model
    dataModel.notSynced = true;
    this.view = new views.ModelView({
        model: dataModel,
        collection: this.collection,
        // tpl-departments-details is a template identifier for chosen
model element
        templateName: '#tpl-departments-details',
        navigate: function (id) {
            self.navigate(id, false);
        },
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
},
getData: function () {
    return {
        deptNo: $('#deptNo').val(),

```

```
        deptName: $('#deptName').val()
    );
}
});
new AppRouter();

Backbone.history.start();
});
```

## HTML STRANICA "DEPARTMENTS" I TEST

Za JavaScript klijent RESTful servisa DepartmentsFacadeREST kreira se HTML stranica.

Sledi listing stranice departments.html.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
    <link rel='stylesheet' href='http://mottie.github.com/tablesorter/addons/
pager/jquery.tablesorter.pager.css'>
    <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
    <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
    <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>
    <script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
    <script src='http://mottie.github.com/tablesorter/addons/pager/
jquery.tablesorter.pager.js'></script>
    <script src='Departments.js'></script>
  </head>
  <body>
    <div id='create'></div>

    <table id='datatable' class='tablesorter-blue'>
    </table>
    <div class='pager' id='pager'>
      <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
first.png' class='first' alt='First' />
      <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
prev.png' class='prev' alt='Prev' />
      <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
      <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
```

```
next.png' class='next' alt='Next'/>
    <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
last.png' class='last' alt='Last'/>
    <select class='pagesize'>
        <option selected='selected' value='10'>10</option>
        <option value='20'>20</option>
        <option value='30'>30</option>
        <option value='40'>40</option>
    </select>
</div>
<br/>

<div id='details'></div>

<!-- Templates -->
<script type='text/template' id='tpl-create'>
    <!--
        Put your controls to create new entity here.

        Class 'new' is used to listen on events in JS code.
    -->
    <button class='new'>Create</button>
</script>

<script type='text/template' id='thead'>
    <thead>
        <tr>
            <th>deptNo</th>
            <th>deptName</th>
        </tr>
    </thead>
</script>
<script type='text/template' id='tpl-departments-list-item'>
    <td><a href='#<%= deptNo %>'><%= deptNo %></a></td>
    <td><%= deptName %></td>
</script>

<script type='text/template' id='tpl-departments-details'>
    <div>
        <table>
            <tbody>
                <tr><td>Id</td>
                <td>
                    <input type='text' id='deptNo' name='id' value='<%= typeof(deptNo) !==
"undefined" ? deptNo : "" %>' />
                </td>
                </tr>
                <tr>
                    <td>deptName</td><td><input type='text' id='deptName' name='deptName'>
                    <input type='text' id='deptName' name='deptName' value='<%= deptName %>' /></td>
                </tr>
            </tbody>
        </table>
    </div>
</script>
```

```

<!--
Put your controls to create new entity here.
Classes 'save' and 'delete' are used to listen on events in JS code.
-->
<button class='save'>Save</button>
<button class='delete'>Delete</button>
</div>
</script>

</body>
</html>

```

Učitana stranica departments.html, sa odgovarajućim sadržajem iz baze podataka, prikazana je sledećom slikom.

Create	
deptNo	deptName
3	FAM
2	FDU
1	FIT

10 ▾

Id	<input type="text" value="1"/>
deptName	<input type="text" value="FIT"/>
<input type="button" value="Save"/> <input type="button" value="Delete"/>	

Slika 10.1.2 Test primene RESTful servisa DepartmentsFacadeREST [izvor: autor]

## TABELA "DEPT\_EMP" - RESTFUL SERVIS I JAVA KLIJENT

*Kreiraju se tražene datoteke nad tabelom "dept\_emp".*

Sledi listing RESTful servisa DeptEmpFacadeREST.

```

package com.metropolitan.restdemo.service;

import com.metropolitan.restdemo.DeptEmp;
import com.metropolitan.restdemo.DeptEmpPK;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.PathSegment;

/**

```

```
*  
* @author Vladimir Milicevic  
*/  
@Stateless  
@Path("com.metropolitan.restdemo.deptemp")  
public class DeptEmpFacadeREST extends AbstractFacade<DeptEmp> {  
  
    @PersistenceContext(unitName = "RESTfulDemoPU")  
    private EntityManager em;  
  
    private DeptEmpPK getPrimaryKey(PathSegment pathSegment) {  
        /*  
         * pathSemgent represents a URI path segment and any associated matrix  
         * parameters.  
         * URI path part is supposed to be in form of  
         'somePath;empNo=empNoValue;deptNo=deptNoValue'.  
         * Here 'somePath' is a result of getPath() method invocation and  
         * it is ignored in the following code.  
         * Matrix parameters are used as field names to build a primary key  
         instance.  
        */  
        com.metropolitan.restdemo.DeptEmpPK key = new  
com.metropolitan.restdemo.DeptEmpPK();  
        javax.ws.rs.core.MultivaluedMap<String, String> map =  
pathSegment.getMatrixParameters();  
        java.util.List<String> empNo = map.get("empNo");  
        if (empNo != null && !empNo.isEmpty()) {  
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));  
        }  
        java.util.List<String> deptNo = map.get("deptNo");  
        if (deptNo != null && !deptNo.isEmpty()) {  
            key.setDeptNo(deptNo.get(0));  
        }  
        return key;  
    }  
  
    public DeptEmpFacadeREST() {  
        super(DeptEmp.class);  
    }  
  
    @POST  
    @Override  
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public void create(DeptEmp entity) {  
        super.create(entity);  
    }  
  
    @PUT  
    @Path("{id}")  
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
    public void edit(@PathParam("id") PathSegment id, DeptEmp entity) {  
        super.edit(entity);  
    }
```

```
@DELETE
@Path("{id}")
public void remove(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.DeptEmpPK key = getPrimaryKey(id);
    super.remove(super.find(key));
}

@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public DeptEmp find(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.DeptEmpPK key = getPrimaryKey(id);
    return super.find(key);
}

@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<DeptEmp> findAll() {
    return super.findAll();
}

@GET
@Path("{from}/{to}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<DeptEmp> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
    return super.findRange(new int[]{from, to});
}

@GET
@Path("count")
@Produces(MediaType.TEXT_PLAIN)
public String countREST() {
    return String.valueOf(super.count());
}

@Override
protected EntityManager getEntityManager() {
    return em;
}

}
```

Sledi listing Java klijenta RESTful servisa .

```
package com.metropolitan.restdemo.jaxrs;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;
```

```
/**  
 * Jersey REST client generated for REST resource:DeptEmpFacadeREST  
 * [com.metropolitan.restdemo.deptemp]<br/>  
 * USAGE:  
 * <pre>  
 *     DeptEmp client = new DeptEmp();  
 *     Object response = client.XXX(...);  
 *     // do whatever with response  
 *     client.close();  
 *  
 *  
 * @author Vladimir Milicevic  
 */  
public class DeptEmp {  
  
    private WebTarget webTarget;  
    private Client client;  
    private static final String BASE_URI = "http://localhost:8080/RESTfulDemo/  
webresources";  
  
    public DeptEmp() {  
        client = javax.ws.rs.client.ClientBuilder.newClient();  
        webTarget =  
client.target(BASE_URI).path("com.metropolitan.restdemo.deptemp");  
    }  
  
    public String countREST() throws ClientErrorException {  
        WebTarget resource = webTarget;  
        resource = resource.path("count");  
        return  
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);  
    }  
  
    public void edit_XML(Object requestEntity, String id) throws  
ClientErrorException {  
        webTarget.path(java.text.MessageFormat.format("{0}", new  
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_XML).put(javax.ws.rs.c  
lient.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));  
    }  
  
    public void edit_JSON(Object requestEntity, String id) throws  
ClientErrorException {  
        webTarget.path(java.text.MessageFormat.format("{0}", new  
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(javax.ws.rs.  
client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));  
    }  
  
    public <T> T find_XML(Class<T> responseType, String id) throws  
ClientErrorException {  
        WebTarget resource = webTarget;  
        resource = resource.path(java.text.MessageFormat.format("{0}", new  
Object[]{id}));  
        return
```

```
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);  
}  
  
    public <T> T find_JSON(Class<T> responseType, String id) throws  
ClientErrorException {  
    WebTarget resource = webTarget;  
    resource = resource.path(java.text.MessageFormat.format("{0}", new  
Object[]{id}));  
    return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);  
}  
  
    public <T> T findRange_XML(Class<T> responseType, String from, String to)  
throws ClientErrorException {  
    WebTarget resource = webTarget;  
    resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new  
Object[]{from, to}));  
    return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);  
}  
  
    public <T> T findRange_JSON(Class<T> responseType, String from, String to)  
throws ClientErrorException {  
    WebTarget resource = webTarget;  
    resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new  
Object[]{from, to}));  
    return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);  
}  
  
    public void create_XML(Object requestEntity) throws ClientErrorException {  
  
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));  
}  
  
    public void create_JSON(Object requestEntity) throws ClientErrorException {  
  
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));  
}  
  
    public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {  
    WebTarget resource = webTarget;  
    return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);  
}  
  
    public <T> T findAll_JSON(Class<T> responseType) throws ClientErrorException {  
    WebTarget resource = webTarget;  
    return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);  
}
```

```
public void remove(String id) throws ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request().delete();
}

public void close() {
    client.close();
}

}
```

## TABELA "DEPT\_EMP" - JAVASCRIPT KLIJENT I HTML

*Za tabelu "dept\_emp" prilažu se datoteke JavaScript klijenta i HTML stranice*

Sledi listing JavaScript klijenta RESTful servisa .

```
var app = {
    // Create this closure to contain the cached modules
    module: function () {
        // Internal module cache.
        var modules = {};

        // Create a new module reference scaffold or load an
        // existing module.
        return function (name) {
            // If this module has already been created, return it.
            if (modules[name]) {
                return modules[name];
            }

            // Create a module and save it under this name
            return modules[name] = {Views: {}};
        };
    }()
};

(function (models) {

// Model for DeptEmp entity
models.DeptEmp = Backbone.Model.extend({
    urlRoot: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptemp/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
            }
        };
    }
});
```

```
// (f.e. locally) then Cross-origin resource sharing
// headers has to be set on the REST server side.
// Otherwise the JS client has to be copied into the
// some (f.e. the same) Web project on the same domain
alert('Unable to fulfil the request');
}

};

if (method === 'create') {
    options.url = 'http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptemp/';
}
var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
return result;
}

});

// Collection class for DeptEmp entities
models.DeptEmpCollection = Backbone.Collection.extend({
    model: models.DeptEmp,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptemp/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        };
        var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',

```

```
initialize: function (options) {
    this.options = options || {};
    this.model.bind("reset", this.render, this);
    var self = this;
    this.model.bind("add", function (modelName) {
        var row = new views.ListItemView({
            model: modelName,
            templateName: self.options.templateName
        }).render().el;
        $(self.el).append($(row));
        $(self.el).parent().trigger('addRows', [$(row)]);
    });
},
render: function (eventName) {
    var self = this;
    _.each(this.model.models, function (modelName) {
        $(this.el).append(new views.ListItemView({
            model: modelName,
            templateName: self.options.templateName
        }).render().el);
    }, this);
    return this;
}
});

views.ListItemView = Backbone.View.extend({
    tagName: 'tr',
    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },
    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },
    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },
    close: function () {
        var table = $(this.el).parent().parent();
        table.trigger('disable.pager');
        $(this.el).unbind();
    }
});
```

```

        $(this.el).remove();
        table.trigger('enable.pager');
    }

});

views.ModelView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    /*
     *  Classes "save" and "delete" are used on the HTML controls to listen
events.
     *  So it is supposed that HTML has controls with these classes.
     */
    events: {
        "change input": "change",
        "click .save": "save",
        "click .delete": "drop"
    },

    change: function (event) {
        var target = event.target;
        console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
    },

    save: function () {
        // TODO : put save code here
        var hash = this.options.getHashObject();
        this.model.set(hash);
        if (this.model.isNew() && this.collection) {
            var self = this;
            this.collection.create(this.model, {
                success: function () {
                    // see isNew() method implementation in the model
                }
            });
        }
    }
});

```

```
        self.model.notSynced = false;
        self.options.navigate(self.model.id);
    }
});
} else {
    this.model.save();
    this.model.el.parent().parent().trigger("update");
}
return false;
},

drop: function () {
    this.model.destroy({
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
            */
            window.history.back();
        }
    });
    return false;
},

close: function () {
    $(this.el).unbind();
    $(this.el).empty();
}
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend{

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },

    render: function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
        */
        return _.template($(this.options.templateName).html())(json);
    },
}

/*

```

```

        * Class "new" is used on the control to listen events.
        * So it is supposed that HTML has a control with "new" class.
        */
events: {
    "click .new": "create"
},

create: function (event) {
    this.options.navigate();
    return false;
}
});

})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
        routes: {
            '' : 'list',
            'new' : 'create'

            ,
            ':id' : 'details'
        },
        initialize: function () {
            var self = this;
            $('#create').html(new views.CreateView({
                // tpl-create is template identifier for 'create' block
                templateName: '#tpl-create',
                navigate: function () {
                    self.navigate('new', true);
                }
            }).render().el);
        },
        list: function () {
            this.collection = new models.DeptEmpCollection();
            var self = this;
            this.collection.fetch({
                success: function () {
                    self.listView = new views.ListView({
                        model: self.collection,
                        // tpl-deptemp-list-item is template identifier for item
                        templateName: '#tpl-deptemp-list-item'
                    });
                    $('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()));
                }
            });
            if (self.requestedId) {
                self.details(self.requestedId);
            }
        }
    });
});

```

```

        var pagerOptions = {
            // target the pager markup
            container: $('.pager'),
            // output string - default is '{page}/{totalPages}';
possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
            output: '{startRow} to {endRow} ({totalRows})',
            // starting page of the pager (zero based index)
            page: 0,
            // Number of visible rows - default is 10
            size: 10
        };
        $('#datatable').tablesorter({widthFixed: true,
            widgets: ['zebra']}).  

            tablesorterPager(pagerOptions);
        }
    });
},
details: function (id) {
    if (this.collection) {
        this.deptemp = this.collection.get(id);
        if (this.view) {
            this.view.close();
        }
        var self = this;
        this.view = new views.ModelView({
            model: this.deptemp,
            // tpl-deptemp-details is template identifier for chosen model
element
            templateName: '#tpl-deptemp-details',
            getHashObject: function () {
                return self.getData();
            }
        });
        $('#details').html(this.view.render().el);
    } else {
        this.requestedId = id;
        this.list();
    }
},
create: function () {
    if (this.view) {
        this.view.close();
    }
    var self = this;
    var dataModel = new models.DeptEmp();
    // see isNew() method implementation in the model
    dataModel.notSynced = true;
    this.view = new views.ModelView({
        model: dataModel,
        collection: this.collection,
        // tpl-deptemp-details is a template identifier for chosen model
element
        templateName: '#tpl-deptemp-details',
    }
}

```

```
        navigate: function (id) {
            self.navigate(id, false);
        },

        getHashObject: function () {
            return self.getData();
        }
    );
    $('#details').html(this.view.render().el);
},
getData: function () {
    return {
    };
}
});
new AppRouter();

Backbone.history.start();
});
```

Sledi listing HTML stranice za JavaScript klijent RESTful servisa .

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/addons/
pager/jquery.tablesorter.pager.css'>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>
        <script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
        <script src='http://mottie.github.com/tablesorter/addons/pager/
jquery.tablesorter.pager.js'></script>
        <script src='DeptEmp.js'></script>
    </head>
    <body>
        <div id='create'></div>

        <table id='datatable' class='tablesorter-blue'>
        </table>
        <div class='pager' id='pager'>
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
first.png' class='first' alt='First' />
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
```

```
prev.png' class='prev' alt='Prev'/>
    <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
    <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
next.png' class='next' alt='Next'/>
    <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
last.png' class='last' alt='Last'/>
    <select class='pagesize'>
        <option selected='selected' value='10'>10</option>
        <option value='20'>20</option>
        <option value='30'>30</option>
        <option value='40'>40</option>
    </select>
</div>
<br/>

<div id='details'></div>

<!-- Templates -->
<script type='text/template' id='tpl-create'>
    <!--
        Put your controls to create new entity here.

        Class 'new' is used to listen on events in JS code.
    -->
    <button class='new'>Create</button>
</script>

<script type='text/template' id='thead'>
    <thead>
        <tr>
        </tr>
    </thead>
</script>
<script type='text/template' id='tpl-deptemp-list-item'>
</script>

<script type='text/template' id='tpl-deptemp-details'>
    <div>
        <table>
            <tbody>
            </tbody>
        </table>
    <!--
        Put your controls to create new entity here.
        Classes 'save' and 'delete' are used to listen on events in JS code.
    -->
        <button class='save'>Save</button>
        <button class='delete'>Delete</button>
    </div>
</script>
```

```
</body>  
</html>
```

## TABELA "DEPT\_MANAGER" - RESTFUL SERVIS I JAVA KLIJENT

*Kreiraju se tražene datoteke nad tabelom "dept\_manager".*

Sledi listing RESTful servisa DeptManagerFacadeREST.

```
package com.metropolitan.restdemo.service;

import com.metropolitan.restdemo.DeptManager;
import com.metropolitan.restdemo.DeptManagerPK;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.PathSegment;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
@Path("com.metropolitan.restdemo.deptmanager")
public class DeptManagerFacadeREST extends AbstractFacade<DeptManager> {

    @PersistenceContext(unitName = "RESTfulDemoPU")
    private EntityManager em;

    private DeptManagerPK getPrimaryKey(PathSegment pathSegment) {
        /*
         * pathSemgent represents a URI path segment and any associated matrix
         * parameters.
         * URI path part is supposed to be in form of
         * 'somePath;deptNo=deptNoValue;empNo=empNoValue'.
         * Here 'somePath' is a result of getPath() method invocation and
         * it is ignored in the following code.
         * Matrix parameters are used as field names to build a primary key
         * instance.
        */
    }
}
```

```
com.metropolitan.restdemo.DeptManagerPK key = new
com.metropolitan.restdemo.DeptManagerPK();
    javax.ws.rs.core.MultivaluedMap<String, String> map =
pathSegment.getMatrixParameters();
    java.util.List<String> deptNo = map.get("deptNo");
    if (deptNo != null && !deptNo.isEmpty()) {
        key.setDeptNo(deptNo.get(0));
    }
    java.util.List<String> empNo = map.get("empNo");
    if (empNo != null && !empNo.isEmpty()) {
        key.setEmpNo(new java.lang.Integer(empNo.get(0)));
    }
    return key;
}

public DeptManagerFacadeREST() {
    super(DeptManager.class);
}

@POST
@Override
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public void create(DeptManager entity) {
    super.create(entity);
}

@PUT
@Path("{id}")
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public void edit(@PathParam("id") PathSegment id, DeptManager entity) {
    super.edit(entity);
}

@DELETE
@Path("{id}")
public void remove(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.DeptManagerPK key = getPrimaryKey(id);
    super.remove(super.find(key));
}

@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public DeptManager find(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.DeptManagerPK key = getPrimaryKey(id);
    return super.find(key);
}

@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<DeptManager> findAll() {
    return super.findAll();
```

```
}

@GET
@Path("{from}/{to}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<DeptManager> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
    return super.findRange(new int[]{from, to});
}

@GET
@Path("count")
@Produces(MediaType.TEXT_PLAIN)
public String countREST() {
    return String.valueOf(super.count());
}

@Override
protected EntityManager getEntityManager() {
    return em;
}

}
```

Sledi listing Java klijenta RESTful servisa .

```
package com.metropolitan.restdemo.jaxrs;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;

/**
 * Jersey REST client generated for REST resource:DeptManagerFacadeREST
 * [com.metropolitan.restdemo.deptmanager]<br/>
 * USAGE:
 * <pre>
 *     DeptManager client = new DeptManager();
 *     Object response = client.XXX(...);
 *     // do whatever with response
 *     client.close();
 *
 *
 *     @author Vladimir Milicevic
 * />
public class DeptManager {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/RESTfulDemo/
webresources";

    public DeptManager() {
```

```
client = javax.ws.rs.client.ClientBuilder.newClient();
webTarget =
client.target(BASE_URI).path("com.metropolitan.restdemo.deptmanager");
}

public String countREST() throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("count");
    return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
}

public void edit_XML(Object requestEntity, String id) throws
ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_XML).put(javax.ws.rs.c
lient.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
}

public void edit_JSON(Object requestEntity, String id) throws
ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(javax.ws.rs.
client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
}

public <T> T find_XML(Class<T> responseType, String id) throws
ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

public <T> T find_JSON(Class<T> responseType, String id) throws
ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public <T> T findRange_XML(Class<T> responseType, String from, String to)
throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}
```

```
public <T> T findRange_JSON(Class<T> responseType, String from, String to)
throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public void create_XML(Object requestEntity) throws ClientErrorException {
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
}

public void create_JSON(Object requestEntity) throws ClientErrorException {
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
}

public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

public <T> T findAll_JSON(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public void remove(String id) throws ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request().delete();
}

public void close() {
    client.close();
}

}
```

## TABELA "DEPT\_MANAGER" - JAVASCRIPT KLIJENT I HTML

*Za tabelu "dept\_manager" prilažu se datoteke JavaScript klijenta i HTML stranice.*

Sledi listing JavaScript klijenta RESTful servisa .

```
var app = {
    // Create this closure to contain the cached modules
    module: function () {
        // Internal module cache.
        var modules = {};

        // Create a new module reference scaffold or load an
        // existing module.
        return function (name) {
            // If this module has already been created, return it.
            if (modules[name]) {
                return modules[name];
            }

            // Create a module and save it under this name
            return modules[name] = {Views: {}};
        };
    }()
};

(function (models) {

    // Model for DeptManager entity
    models.DeptManager = Backbone.Model.extend({
        urlRoot: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptmanager/",
        sync: function (method, model, options) {
            options || (options = {});
            var errorHandler = {
                error: function (jqXHR, textStatus, errorThrown) {
                    // TODO: put your error handling code here
                    // If you use the JS client from the different domain
                    // (f.e. locally) then Cross-origin resource sharing
                    // headers has to be set on the REST server side.
                    // Otherwise the JS client has to be copied into the
                    // some (f.e. the same) Web project on the same domain
                    alert('Unable to fulfil the request');
                }
            };

            if (method === 'create') {
                options.url = 'http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptmanager/';
            }
            var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
            return result;
        }
    });

});
```

```

// Collection class for DeptManager entities
models.DeptManagerCollection = Backbone.Collection.extend({
    model: models.DeptManager,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.deptmanager/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        };
        var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',
        initialize: function (options) {
            this.options = options || {};
            this.model.bind("reset", this.render, this);
            var self = this;
            this.model.bind("add", function (modelName) {
                var row = new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el;
                $(self.el).append($(row));
                $(self.el).parent().trigger('addRows', [$(row)]);
            });
        },
        render: function (eventName) {
            var self = this;
            _.each(this.model.models, function (modelName) {
                $(this.el).append(new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el);
            });
        }
    });
})

```

```
        },
        return this;
    }
});

views.ListItemView = Backbone.View.extend({
    tagName: 'tr',

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML
         * $(this.options.templateName) is element access to the element
         * using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    close: function () {
        var table = $(this.el).parent().parent();
        table.trigger('disable.pager');
        $(this.el).unbind();
        $(this.el).remove();
        table.trigger('enable.pager');
    }
};

views.ModelView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    template: function (json) {
        /*
         * templateName is element identifier in HTML

```

```
        * $(this.options.templateName) is element access to the element
        * using jQuery
        */
    return _.template($(this.options.templateName).html())(json);
},

/*
 * Classes "save" and "delete" are used on the HTML controls to listen
events.
 * So it is supposed that HTML has controls with these classes.
*/
events: {
    "change input": "change",
    "click .save": "save",
    "click .delete": "drop"
},

change: function (event) {
    var target = event.target;
    console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
    },

save: function () {
    // TODO : put save code here
    var hash = this.options.getHashObject();
    this.model.set(hash);
    if (this.model.isNew() && this.collection) {
        var self = this;
        this.collection.create(this.model, {
            success: function () {
                // see isNew() method implementation in the model
                self.model.notSynced = false;
                self.options.navigate(self.model.id);
            }
        });
    } else {
        this.model.save();
        this.model.el.parent().parent().trigger("update");
    }
    return false;
},

drop: function () {
    this.model.destroy({
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
            */
            window.history.back();
        }
    });
}
```

```
        return false;
    },

    close: function () {
        $(this.el).unbind();
        $(this.el).empty();
    }
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },

    render: function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    /*
     *  Class "new" is used on the control to listen events.
     *  So it is supposed that HTML has a control with "new" class.
     */
    events: {
        "click .new": "create"
    },

    create: function (event) {
        this.options.navigate();
        return false;
    }
});

})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
```

```

routes: {
    '' : 'list',
    'new' : 'create'

    ,
    ':id' : 'details'
},
initialize: function () {
    var self = this;
    $('#create').html(new views.CreateView({
        // tpl-create is template identifier for 'create' block
        templateName: '#tpl-create',
        navigate: function () {
            self.navigate('new', true);
        }
    }).render().el);
},
list: function () {
    this.collection = new models.DeptManagerCollection();
    var self = this;
    this.collection.fetch({
        success: function () {
            self.listView = new views.ListView({
                model: self.collection,
                // tpl-deptmanager-list-item is template identifier for item
                templateName: '#tpl-deptmanager-list-item'
            });
        }
    });

    $('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()));
    if (self.requestedId) {
        self.details(self.requestedId);
    }
    var pagerOptions = {
        // target the pager markup
        container: $('.pager'),
        // output string - default is '{page}/{totalPages}'
possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
        output: '{startRow} to {endRow} ({totalRows})',
        // starting page of the pager (zero based index)
        page: 0,
        // Number of visible rows - default is 10
        size: 10
    };
    $('#datatable').tablesorter({widthFixed: true,
        widgets: ['zebra']}).
        tablesorterPager(pagerOptions);
    }
}),
details: function (id) {
    if (this.collection) {
        this.deptmanager = this.collection.get(id);
        if (this.view) {

```

```

        this.view.close();
    }
    var self = this;
    this.view = new views.ModelView({
        model: this.deptmanager,
        // tpl-deptmanager-details is template identifier for chosen
model element
        templateName: '#tpl-deptmanager-details',
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
} else {
    this.requestedId = id;
    this.list();
}
},
create: function () {
    if (this.view) {
        this.view.close();
    }
    var self = this;
    var dataModel = new models.DeptManager();
    // see isNew() method implementation in the model
    dataModel.notSynced = true;
    this.view = new views.ModelView({
        model: dataModel,
        collection: this.collection,
        // tpl-deptmanager-details is a template identifier for chosen
model element
        templateName: '#tpl-deptmanager-details',
        navigate: function (id) {
            self.navigate(id, false);
        },
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
},
getData: function () {
    return {
    };
}
});
new AppRouter();

Backbone.history.start();
});

```

Sledi listing HTML stranice za JavaScript klijent RESTful servisa .

```
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/addons/
pager/jquery.tablesorter.pager.css'>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>
        <script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
        <script src='http://mottie.github.com/tablesorter/addons/pager/
jquery.tablesorter.pager.js'></script>
        <script src='DeptManager.js'></script>
    </head>
    <body>
        <div id='create'></div>

        <table id='datatable' class='tablesorter-blue'>
        </table>
        <div class='pager' id='pager'>
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
first.png' class='first' alt='First' />
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
prev.png' class='prev' alt='Prev' />
            <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
next.png' class='next' alt='Next' />
            <img src='http://mottie.github.com/tablesorter/addons/pager/icons/
last.png' class='last' alt='Last' />
            <select class='pagesize'>
                <option selected='selected' value='10'>10</option>
                <option value='20'>20</option>
                <option value='30'>30</option>
                <option value='40'>40</option>
            </select>
        </div>
        <br/>

        <div id='details'></div>

        <!-- Templates -->
        <script type='text/template' id='tpl-create'>
            <!--
```

```
Put your controls to create new entity here.

Class 'new' is used to listen on events in JS code.
-->
<button class='new'>Create</button>
</script>

<script type='text/template' id='thead'>
  <thead>
    <tr>
    </tr>
  </thead>
</script>
<script type='text/template' id='tpl-deptmanager-list-item'>
</script>

<script type='text/template' id='tpl-deptmanager-details'>
  <div>
    <table>
      <tbody>
      </tbody>
    </table>
  <!--
  Put your controls to create new entity here.
  Classes 'save' and 'delete' are used to listen on events in JS code.
  -->
    <button class='save'>Save</button>
    <button class='delete'>Delete</button>
  </div>
</script>

</body>
</html>
```

## TABELA "SALARIES" - RESTFUL SERVIS I JAVA KLIJENT

*Kreiraju se tražene datoteke nad tabelom "salaries".*

Sledi listing RESTful servisa SalariesFacadeREST.

```
package com.metropolitan.restdemo.service;

import com.metropolitan.restdemo.Salaries;
import com.metropolitan.restdemo.SalariesPK;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
```

```
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.PathSegment;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
@Path("com.metropolitan.restdemo.salaries")
public class SalariesFacadeREST extends AbstractFacade<Salaries> {

    @PersistenceContext(unitName = "RESTfulDemoPU")
    private EntityManager em;

    private SalariesPK getPrimaryKey(PathSegment pathSegment) {
        /*
         * pathSegment represents a URI path segment and any associated matrix
         * parameters.
         * URI path part is supposed to be in form of
         * 'somePath;empNo=empNoValue;fromDate=fromDateValue'.
         * Here 'somePath' is a result of getPath() method invocation and
         * it is ignored in the following code.
         * Matrix parameters are used as field names to build a primary key
         * instance.
        */
        com.metropolitan.restdemo.SalariesPK key = new
com.metropolitan.restdemo.SalariesPK();
        javax.ws.rs.core.MultivaluedMap<String, String> map =
pathSegment.getMatrixParameters();
        java.util.List<String> empNo = map.get("empNo");
        if (empNo != null && !empNo.isEmpty()) {
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));
        }
        java.util.List<String> fromDate = map.get("fromDate");
        if (fromDate != null && !fromDate.isEmpty()) {
            key.setFromDate(new java.util.Date(fromDate.get(0)));
        }
        return key;
    }

    public SalariesFacadeREST() {
        super(Salaries.class);
    }

    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
```

```
public void create(Salaries entity) {
    super.create(entity);
}

@PUT
@Path("{id}")
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public void edit(@PathParam("id") PathSegment id, Salaries entity) {
    super.edit(entity);
}

@DELETE
@Path("{id}")
public void remove(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.SalariesPK key = getPrimaryKey(id);
    super.remove(super.find(key));
}

@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Salaries find(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.SalariesPK key = getPrimaryKey(id);
    return super.find(key);
}

@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Salaries> findAll() {
    return super.findAll();
}

@GET
@Path("{from}/{to}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Salaries> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
    return super.findRange(new int[]{from, to});
}

@GET
@Path("count")
@Produces(MediaType.TEXT_PLAIN)
public String countREST() {
    return String.valueOf(super.count());
}

@Override
protected EntityManager getEntityManager() {
    return em;
}
```

```
}
```

Sledi listing Java klijenta RESTful servisa .

```
package com.metropolitan.restdemo.jaxrs;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;

/**
 * Jersey REST client generated for REST resource:SalariesFacadeREST
 * [com.metropolitan.restdemo.salaries]<br/>
 * USAGE:
 * <pre>
 *     Salaries client = new Salaries();
 *     Object response = client.XXX(...);
 *     // do whatever with response
 *     client.close();
 *
 *
 * @author Vladimir Milicevic
 */
public class Salaries {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/RESTfulDemo/
webresources";

    public Salaries() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget =
client.target(BASE_URI).path("com.metropolitan.restdemo.salaries");
    }

    public String countREST() throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("count");
        return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
    }

    public void edit_XML(Object requestEntity, String id) throws
ClientErrorException {
        webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_XML).put(javax.ws.rs.c
lient.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
    }

    public void edit_JSON(Object requestEntity, String id) throws
ClientErrorException {
```

```
        webTarget.path(java.text.MessageFormat.format("{0}", new
Object[] {id})).request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(javax.ws.rs.
client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
    }

    public <T> T find_XML(Class<T> responseType, String id) throws
ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[] {id}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
    }

    public <T> T find_JSON(Class<T> responseType, String id) throws
ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[] {id}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
    }

    public <T> T findRange_XML(Class<T> responseType, String from, String to)
throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[] {from, to}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
    }

    public <T> T findRange_JSON(Class<T> responseType, String from, String to)
throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[] {from, to}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
    }

    public void create_XML(Object requestEntity) throws ClientErrorException {
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML).post(javax.ws.rs.client.
Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
    }

    public void create_JSON(Object requestEntity) throws ClientErrorException {
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(javax.ws.rs.client.
Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
    }
```

```
public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}

public <T> T findAll_JSON(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public void remove(String id) throws ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request().delete();
}

public void close() {
    client.close();
}

}
```

## TABELA "SALARIES" - JAVASCRIPT KLIJENT I HTML

*Za tabelu "salaries" prilažu se datoteke JavaScript klijenta i HTML stranice.*

Sledi listing JavaScript klijenta RESTful servisa .

```
var app = {
    // Create this closure to contain the cached modules
    module: function () {
        // Internal module cache.
        var modules = {};

        // Create a new module reference scaffold or load an
        // existing module.
        return function (name) {
            // If this module has already been created, return it.
            if (modules[name]) {
                return modules[name];
            }

            // Create a module and save it under this name
            return modules[name] = {Views: {}};
        };
    }()
};

(function (models) {
```

```
// Model for Salaries entity
models.Salaries = Backbone.Model.extend({
    urlRoot: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.salaries/",
    defaults: {
        salary: ""
    },
    toViewJson: function () {
        var result = this.toJSON(); // displayName property is used to render
item in the list
        result.displayName = this.get('salary');
        return result;
    },
    isNew: function () {
        // default isNew() method implementation is
        // based on the 'id' initialization which
        // sometimes is required to be initialized.
        // So isNew() is rediefined here
        return this.notSynced;
    },
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        };
        if (method === 'create') {
            options.url = 'http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.salaries/';
        }
        var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

// Collection class for Salaries entities
models.SalariesCollection = Backbone.Collection.extend({
    model: models.Salaries,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.salaries/",
```

```
sync: function (method, model, options) {
    options || (options = {});
    var errorHandler = {
        error: function (jqXHR, textStatus, errorThrown) {
            // TODO: put your error handling code here
            // If you use the JS client from the different domain
            // (f.e. locally) then Cross-origin resource sharing
            // headers has to be set on the REST server side.
            // Otherwise the JS client has to be copied into the
            // some (f.e. the same) Web project on the same domain
            alert('Unable to fulfil the request');
        }
    };
    var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
    return result;
}
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',
        initialize: function (options) {
            this.options = options || {};
            this.model.bind("reset", this.render, this);
            var self = this;
            this.model.bind("add", function (modelName) {
                var row = new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el;
                $(self.el).append($(row));
                $(self.el).parent().trigger('addRows', [$row]);
            });
        },
        render: function (eventName) {
            var self = this;
            _.each(this.model.models, function (modelName) {
                $(this.el).append(new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el);
            }, this);
            return this;
        }
    });
})�;
```

```
views.ListItemView = Backbone.View.extend({
    tagName: 'tr',

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    close: function () {
        var table = $(this.el).parent().parent();
        table.trigger('disable.pager');
        $(this.el).unbind();
        $(this.el).remove();
        table.trigger('enable.pager');
    }
});

views.ModelView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.model.bind("change", this.render, this);
    },

    render: function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },
});
```

```
/*
 * Classes "save" and "delete" are used on the HTML controls to listen
events.
 * So it is supposed that HTML has controls with these classes.
*/
events: {
    "change input": "change",
    "click .save": "save",
    "click .delete": "drop"
},

change: function (event) {
    var target = event.target;
    console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
},

save: function () {
    // TODO : put save code here
    var hash = this.options.getHashObject();
    this.model.set(hash);
    if (this.model.isNew() && this.collection) {
        var self = this;
        this.collection.create(this.model, {
            success: function () {
                // see isNew() method implementation in the model
                self.model.notSynced = false;
                self.options.navigate(self.model.id);
            }
        });
    } else {
        this.model.save();
        this.model.el.parent().parent().trigger("update");
    }
    return false;
},

drop: function () {
    this.model.destroy({
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
            */
            window.history.back();
        }
    });
    return false;
},

close: function () {
    $(this.el).unbind();
```

```

        $(this.el).empty();
    }
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend({

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },

    render: function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    template: function (json) {
        /*
         *  templateName is element identifier in HTML
         *  $(this.options.templateName) is element access to the element
         *  using jQuery
         */
        return _.template($(this.options.templateName).html())(json);
    },

    /*
     *  Class "new" is used on the control to listen events.
     *  So it is supposed that HTML has a control with "new" class.
     */
    events: {
        "click .new": "create"
    },

    create: function (event) {
        this.options.navigate();
        return false;
    }
});

})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
        routes: {
            '' : 'list',
            'new' : 'create'
            ,
            ':id' : 'details'
        }
    });
});

```

```

        },
        initialize: function () {
            var self = this;
            $('#create').html(new views.CreateView({
                // tpl-create is template identifier for 'create' block
                templateName: '#tpl-create',
                navigate: function () {
                    self.navigate('new', true);
                }
            }).render().el);
        },
        list: function () {
            this.collection = new models.SalariesCollection();
            var self = this;
            this.collection.fetch({
                success: function () {
                    self.listView = new views.ListView({
                        model: self.collection,
                        // tpl-salaries-list-item is template identifier for item
                        templateName: '#tpl-salaries-list-item'
                    });
                }
            });
            $('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()));
            if (self.requestedId) {
                self.details(self.requestedId);
            }
            var pagerOptions = {
                // target the pager markup
                container: $('.pager'),
                // output string - default is '{page}/{totalPages}';
possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
                output: '{startRow} to {endRow} ({totalRows})',
                // starting page of the pager (zero based index)
                page: 0,
                // Number of visible rows - default is 10
                size: 10
            };
            $('#datatable').tablesorter({widthFixed: true,
                widgets: ['zebra']});
            tablesorterPager(pagerOptions);
        }
    });
},
details: function (id) {
    if (this.collection) {
        this.salaries = this.collection.get(id);
        if (this.view) {
            this.view.close();
        }
        var self = this;
        this.view = new views.ModelView({
            model: this.salaries,

```

```

        // tpl-salaries-details is template identifier for chosen model
element
        templateName: '#tpl-salaries-details',
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
} else {
    this.requestedId = id;
    this.list();
}
},
create: function () {
    if (this.view) {
        this.view.close();
    }
    var self = this;
    var dataModel = new models.Salaries();
    // see isNew() method implementation in the model
    dataModel.notSynced = true;
    this.view = new views.ModelView({
        model: dataModel,
        collection: this.collection,
        // tpl-salaries-details is a template identifier for chosen model
element
        templateName: '#tpl-salaries-details',
        navigate: function (id) {
            self.navigate(id, false);
        },
        getHashObject: function () {
            return self.getData();
        }
    });
    $('#details').html(this.view.render().el);
},
getData: function () {
    return {
        salary: $('#salary').val()
    };
}
});
new AppRouter();

Backbone.history.start();
});

```

Sledi listing HTML stranice za JavaScript klijent RESTful servisa .

```

<html>
    <head>

```

```
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
    <link rel='stylesheet' href='http://mottie.github.com/tablesorter addons/
pager/jquery.tablesorter.pager.css'>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>
        <script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
        <script src='http://mottie.github.com/tablesorter addons/pager/
jquery.tablesorter.pager.js'></script>
        <script src='Salaries.js'></script>
</head>
<body>
    <div id='create'></div>

    <table id='datatable' class='tablesorter-blue'>
    </table>
    <div class='pager' id='pager'>
        <img src='http://mottie.github.com/tablesorter addons/pager/icons/
first.png' class='first' alt='First' />
        <img src='http://mottie.github.com/tablesorter addons/pager/icons/
prev.png' class='prev' alt='Prev' />
            <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
        <img src='http://mottie.github.com/tablesorter addons/pager/icons/
next.png' class='next' alt='Next' />
        <img src='http://mottie.github.com/tablesorter addons/pager/icons/
last.png' class='last' alt='Last' />
        <select class='pagesize'>
            <option selected='selected' value='10'>10</option>
            <option value='20'>20</option>
            <option value='30'>30</option>
            <option value='40'>40</option>
        </select>
    </div>
    <br/>

    <div id='details'></div>

    <!-- Templates -->
    <script type='text/template' id='tpl-create'>
        <!--
            Put your controls to create new entity here.

            Class 'new' is used to listen on events in JS code.
        -->
```

```

        <button class='new'>Create</button>
    </script>

    <script type='text/template' id='thead'>
        <thead>
            <tr>
                <th>salary</th>
            </tr>
        </thead>
    </script>
    <script type='text/template' id='tpl-salaries-list-item'>
        <td><%= salary %></td>
    </script>

    <script type='text/template' id='tpl-salaries-details'>
        <div>
            <table>
                <tbody>
                    <tr>
                        <td>salary</td><td><input type='text' id='salary' name='salary' value='<%= salary %>' /></td></tr>
                    </tbody>
                </table>
            <!--
                Put your controls to create new entity here.
                Classes 'save' and 'delete' are used to listen on events in JS code.
            -->
            <button class='save'>Save</button>
            <button class='delete'>Delete</button>
        </div>
    </script>

    </body>
</html>

```

## TABELA "TITLES" - RESTFUL SERVIS I JAVA KLIJENT

*Kreiraju se tražene datoteke nad tabelom "titles".*

Sledi listing RESTful servisa TitlesFacadeREST.

```

package com.metropolitan.restdemo.service;

import com.metropolitan.restdemo.Titles;
import com.metropolitan.restdemo.TitlesPK;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;

```

```
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.PathSegment;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
@Path("com.metropolitan.restdemo.titles")
public class TitlesFacadeREST extends AbstractFacade<Titles> {

    @PersistenceContext(unitName = "RESTfulDemoPU")
    private EntityManager em;

    private TitlesPK getPrimaryKey(PathSegment pathSegment) {
        /*
         * pathSemgent represents a URI path segment and any associated matrix
         * parameters.
         * URI path part is supposed to be in form of
         * 'somePath;empNo=empNoValue;title=titleValue;fromDate=fromDateValue'.
         * Here 'somePath' is a result of getPath() method invocation and
         * it is ignored in the following code.
         * Matrix parameters are used as field names to build a primary key
         * instance.
         */
        com.metropolitan.restdemo.TitlesPK key = new
com.metropolitan.restdemo.TitlesPK();
        javax.ws.rs.core.MultivaluedMap<String, String> map =
pathSegment.getMatrixParameters();
        java.util.List<String> empNo = map.get("empNo");
        if (empNo != null && !empNo.isEmpty()) {
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));
        }
        java.util.List<String> title = map.get("title");
        if (title != null && !title.isEmpty()) {
            key.setTitle(title.get(0));
        }
        java.util.List<String> fromDate = map.get("fromDate");
        if (fromDate != null && !fromDate.isEmpty()) {
            key.setFromDate(new java.util.Date(fromDate.get(0)));
        }
        return key;
    }

    public TitlesFacadeREST() {
        super(Titles.class);
    }
}
```

```
}

@POST
@Override
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public void create(Titles entity) {
    super.create(entity);
}

@PUT
@Path("{id}")
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public void edit(@PathParam("id") PathSegment id, Titles entity) {
    super.edit(entity);
}

@DELETE
@Path("{id}")
public void remove(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.TitlesPK key = getPrimaryKey(id);
    super.remove(super.find(key));
}

@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Titles find(@PathParam("id") PathSegment id) {
    com.metropolitan.restdemo.TitlesPK key = getPrimaryKey(id);
    return super.find(key);
}

@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Titles> findAll() {
    return super.findAll();
}

@GET
@Path("{from}/{to}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Titles> findRange(@PathParam("from") Integer from, @PathParam("to")
Integer to) {
    return super.findRange(new int[]{from, to});
}

@GET
@Path("count")
@Produces(MediaType.TEXT_PLAIN)
public String countREST() {
    return String.valueOf(super.count());
}
```

```
@Override
protected EntityManager getEntityManager() {
    return em;
}

}
```

Sledi listing Java klijenta RESTful servisa .

```
package com.metropolitan.restdemo.jaxrs;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;

/**
 * Jersey REST client generated for REST resource:TitlesFacadeREST
 * [com.metropolitan.restdemo.titles]<br/>
 * USAGE:
 * <pre>
 *     Titles client = new Titles();
 *     Object response = client.XXX(...);
 *     // do whatever with response
 *     client.close();
 *
 *
 * @author Vladimir Milicevic
 */
public class Titles {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/RESTfulDemo/
webresources";

    public Titles() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget =
client.target(BASE_URI).path("com.metropolitan.restdemo.titles");
    }

    public String countREST() throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("count");
        return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
    }

    public void edit_XML(Object requestEntity, String id) throws
ClientErrorException {
        webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_XML).put(javax.ws.rs.c
lient.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
    }
}
```

```
}

    public void edit_JSON(Object requestEntity, String id) throws
ClientErrorException {
        webTarget.path(java.text.MessageFormat.format("{0}", new
Object[]{id})).request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(javax.ws.rs.
client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
    }

    public <T> T find_XML(Class<T> responseType, String id) throws
ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
    }

    public <T> T find_JSON(Class<T> responseType, String id) throws
ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}", new
Object[]{id}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
    }

    public <T> T findRange_XML(Class<T> responseType, String from, String to)
throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
    }

    public <T> T findRange_JSON(Class<T> responseType, String from, String to)
throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}", new
Object[]{from, to}));
        return
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
    }

    public void create_XML(Object requestEntity) throws ClientErrorException {

webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML).post(javax.ws.rs.client.
Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
    }

    public void create_JSON(Object requestEntity) throws ClientErrorException {
```

```
webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));  
  
    }  
  
    public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {  
        WebTarget resource = webTarget;  
        return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);  
    }  
  
    public <T> T findAll_JSON(Class<T> responseType) throws ClientErrorException {  
        WebTarget resource = webTarget;  
        return  
resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);  
    }  
  
    public void remove(String id) throws ClientErrorException {  
        webTarget.path(java.text.MessageFormat.format("{0}", new  
Object[]{id})).request().delete();  
    }  
  
    public void close() {  
        client.close();  
    }  
}
```

## TABELA "TITLES" - JAVASCRIPT KLIJENT I HTML

*Za tabelu "titles" prilaže se datoteke JavaScript klijenta i HTML stranice.*

Sledi listing JavaScript klijenta RESTful servisa .

```
var app = {  
    // Create this closure to contain the cached modules  
    module: function () {  
        // Internal module cache.  
        var modules = {};  
  
        // Create a new module reference scaffold or load an  
        // existing module.  
        return function (name) {  
            // If this module has already been created, return it.  
            if (modules[name]) {  
                return modules[name];  
            }  
  
            // Create a module and save it under this name  
            return modules[name] = {Views: {}};  
        };  
    };
```

```
    }()
};

(function (models) {

// Model for Titles entity
    models.Titles = Backbone.Model.extend({
        urlRoot: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.titles/",
        sync: function (method, model, options) {
            options || (options = {});
            var errorHandler = {
                error: function (jqXHR, textStatus, errorThrown) {
                    // TODO: put your error handling code here
                    // If you use the JS client from the different domain
                    // (f.e. locally) then Cross-origin resource sharing
                    // headers has to be set on the REST server side.
                    // Otherwise the JS client has to be copied into the
                    // some (f.e. the same) Web project on the same domain
                    alert('Unable to fulfil the request');
                }
            };
            if (method === 'create') {
                options.url = 'http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.titles/';
            }
            var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
            return result;
        }
    });

// Collection class for Titles entities
models.TitlesCollection = Backbone.Collection.extend({
    model: models.Titles,
    url: "http://localhost:8080/RESTfulDemo/webresources/
com.metropolitan.restdemo.titles/",
    sync: function (method, model, options) {
        options || (options = {});
        var errorHandler = {
            error: function (jqXHR, textStatus, errorThrown) {
                // TODO: put your error handling code here
                // If you use the JS client from the different domain
                // (f.e. locally) then Cross-origin resource sharing
                // headers has to be set on the REST server side.
                // Otherwise the JS client has to be copied into the
                // some (f.e. the same) Web project on the same domain
                alert('Unable to fulfil the request');
            }
        }
    }
});
```

```
};

    var result = Backbone.sync(method, model, _.extend(options,
errorHandler));
        return result;
    }
});

})(app.module("models"));

(function (views) {

    views.ListView = Backbone.View.extend({
        tagName: 'tbody',
        initialize: function (options) {
            this.options = options || {};
            this.model.bind("reset", this.render, this);
            var self = this;
            this.model.bind("add", function (modelName) {
                var row = new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el;
                $(self.el).append($(row));
                $(self.el).parent().trigger('addRows', [$(row)]);
            });
        },
        render: function (eventName) {
            var self = this;
            _.each(this.model.models, function (modelName) {
                $(this.el).append(new views.ListItemView({
                    model: modelName,
                    templateName: self.options.templateName
                }).render().el);
            }, this);
            return this;
        }
    });

    views.ListItemView = Backbone.View.extend({
        tagName: 'tr',

        initialize: function (options) {
            this.options = options || {};
            this.model.bind("change", this.render, this);
            this.model.bind("destroy", this.close, this);
        },
        template: function (json) {
            /*
             * templateName is element identifier in HTML

```

```
        * $(this.options.templateName) is element access to the element
        * using jQuery
        */
    return _.template($(this.options.templateName).html())(json);
},

render: function (eventName) {
    $(this.el).html(this.template(this.model.toJSON()));
    return this;
},

close: function () {
    var table = $(this.el).parent().parent();
    table.trigger('disable.pager');
    $(this.el).unbind();
    $(this.el).remove();
    table.trigger('enable.pager');
}

});

views.ModelView = Backbone.View.extend({

initialize: function (options) {
    this.options = options || {};
    this.model.bind("change", this.render, this);
},

render: function (eventName) {
    $(this.el).html(this.template(this.model.toJSON()));
    return this;
},

template: function (json) {
    /*
     * templateName is element identifier in HTML
     * $(this.options.templateName) is element access to the element
     * using jQuery
     */
    return _.template($(this.options.templateName).html())(json);
},

/*
 * Classes "save" and "delete" are used on the HTML controls to listen
events.
 * So it is supposed that HTML has controls with these classes.
*/
events: {
    "change input": "change",
    "click .save": "save",
    "click .delete": "drop"
},
}
```

```
change: function (event) {
    var target = event.target;
    console.log('changing ' + target.id + ' from: ' + target.defaultValue +
' to: ' + target.value);
},

save: function () {
    // TODO : put save code here
    var hash = this.options.getHashObject();
    this.model.set(hash);
    if (this.model.isNew() && this.collection) {
        var self = this;
        this.collection.create(this.model, {
            success: function () {
                // see isNew() method implementation in the model
                self.model.notSynced = false;
                self.options.navigate(self.model.id);
            }
        });
    } else {
        this.model.save();
        this.model.el.parent().parent().trigger("update");
    }
    return false;
},

drop: function () {
    this.model.destroy({
        success: function () {
            /*
             * TODO : put your code here
             * f.e. alert("Model is successfully deleted");
            */
            window.history.back();
        }
    });
    return false;
},

close: function () {
    $(this.el).unbind();
    $(this.el).empty();
}
});

// This view is used to create new model element
views.CreateView = Backbone.View.extend{

    initialize: function (options) {
        this.options = options || {};
        this.render();
    },
}
```

```

        render: function (eventName) {
            $(this.el).html(this.template());
            return this;
        },

        template: function (json) {
            /*
             *  templateName is element identifier in HTML
             *  $(this.options.templateName) is element access to the element
             *  using jQuery
             */
            return _.template($(this.options.templateName).html())(json);
        },

        /*
         *  Class "new" is used on the control to listen events.
         *  So it is supposed that HTML has a control with "new" class.
         */
        events: {
            "click .new": "create"
        },

        create: function (event) {
            this.options.navigate();
            return false;
        }
    });

})(app.module("views"));

$(function () {
    var models = app.module("models");
    var views = app.module("views");

    var AppRouter = Backbone.Router.extend({
        routes: {
            '' : 'list',
            'new' : 'create'
        },
        ':id': 'details'
    },
    initialize: function () {
        var self = this;
        $('#create').html(new views.CreateView({
            // tpl-create is template identifier for 'create' block
            templateName: '#tpl-create',
            navigate: function () {
                self.navigate('new', true);
            }
        }).render().el);
    },
    list: function () {

```

```
this.collection = new models.TitlesCollection();
var self = this;
this.collection.fetch({
    success: function () {
        self.listView = new views.ListView({
            model: self.collection,
            // tpl-titles-list-item is template identifier for item
            templateName: '#tpl-titles-list-item'
        });
    }

($('#datatable').html(self.listView.render().el).append(_.template($('#thead').html()))
()));

        if (self.requestedId) {
            self.details(self.requestedId);
        }
        var pagerOptions = {
            // target the pager markup
            container: $('.pager'),
            // output string - default is '{page}/{totalPages}';
possiblevariables: {page}, {totalPages},{startRow}, {endRow} and {totalRows}
            output: '{startRow} to {endRow} ({totalRows})',
            // starting page of the pager (zero based index)
            page: 0,
            // Number of visible rows - default is 10
            size: 10
        };
        $('#datatable').tablesorter({widthFixed: true,
            widgets: ['zebra']});
            tablesorterPager(pagerOptions);
        }
    });
},
details: function (id) {
    if (this.collection) {
        this.titles = this.collection.get(id);
        if (this.view) {
            this.view.close();
        }
        var self = this;
        this.view = new views.ModelView({
            model: this.titles,
            // tpl-titles-details is template identifier for chosen model
element
            templateName: '#tpl-titles-details',
            getHashObject: function () {
                return self.getData();
            }
        });
        $('#details').html(this.view.render().el);
    } else {
        this.requestedId = id;
        this.list();
    }
}
```

```

        },
        create: function () {
            if (this.view) {
                this.view.close();
            }
            var self = this;
            var dataModel = new models.Titles();
            // see isNew() method implementation in the model
            dataModel.notSynced = true;
            this.view = new views.ModelView({
                model: dataModel,
                collection: this.collection,
                // tpl-titles-details is a template identifier for chosen model
                element
                templateName: '#tpl-titles-details',
                navigate: function (id) {
                    self.navigate(id, false);
                },
                getHashObject: function () {
                    return self.getData();
                }
            });
            $('#details').html(this.view.render().el);
        },
        getData: function () {
            return {
            };
        }
    );
    new AppRouter();

    Backbone.history.start();
});

```

Sledi listing HTML stranice za JavaScript klijent RESTful servisa .

```

<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter/css/
theme.blue.css'>
        <link rel='stylesheet' href='http://mottie.github.com/tablesorter addons/
pager/jquery.tablesorter.pager.css'>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.6.0/
underscore-min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/
jquery.min.js'></script>
        <script src='http://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/
backbone-min.js'></script>

```

```
<script src='http://mottie.github.com/tablesorter/js/
jquery.tablesorter.min.js'></script>
<script src='http://mottie.github.com/tablesorter addons/pager/
jquery.tablesorter.pager.js'></script>
<script src='Titles.js'></script>
</head>
<body>
<div id='create'></div>

<table id='datatable' class='tablesorter-blue'>
</table>
<div class='pager' id='pager'>
    <img src='http://mottie.github.com/tablesorter addons/pager/icons/
first.png' class='first' alt='First' />
    <img src='http://mottie.github.com/tablesorter addons/pager/icons/
prev.png' class='prev' alt='Prev' />
    <span class='pagedisplay'></span> <!-- this can be any element,
including an input -->
    <img src='http://mottie.github.com/tablesorter addons/pager/icons/
next.png' class='next' alt='Next' />
    <img src='http://mottie.github.com/tablesorter addons/pager/icons/
last.png' class='last' alt='Last' />
    <select class='pagesize'>
        <option selected='selected' value='10'>10</option>
        <option value='20'>20</option>
        <option value='30'>30</option>
        <option value='40'>40</option>
    </select>
</div>
<br/>

<div id='details'></div>

<!-- Templates -->
<script type='text/template' id='tpl-create'>
    <!--
        Put your controls to create new entity here.

        Class 'new' is used to listen on events in JS code.
    -->
    <button class='new'>Create</button>
</script>

<script type='text/template' id='thead'>
    <thead>
        <tr>
        </tr>
    </thead>
</script>
<script type='text/template' id='tpl-titles-list-item'>
</script>
```

```
<script type='text/template' id='tpl-titles-details'>
  <div>
    <table>
      <tbody>
      </tbody>
    </table>
    <!--
        Put your controls to create new entity here.
        Classes 'save' and 'delete' are used to listen on events in JS code.
    -->
    <button class='save'>Save</button>
    <button class='delete'>Delete</button>
  </div>
</script>

</body>
</html>
```

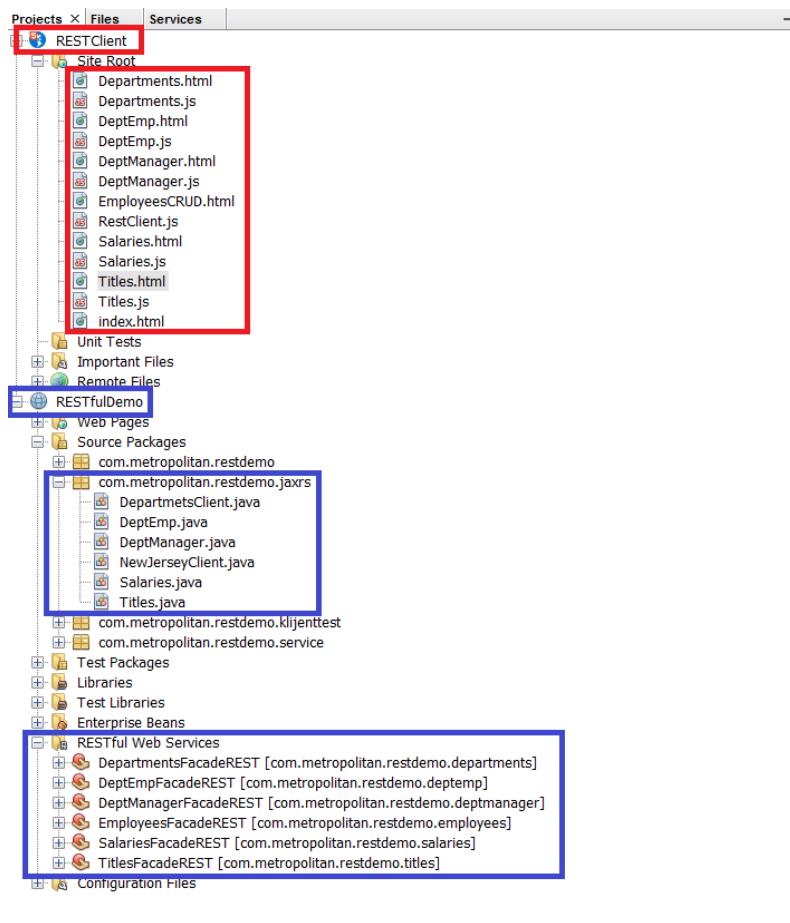
## KOMPLETIRAN PROJEKAT RESTFULDEMO

*Neophodno je na kraju prikazati konačnu strukturu realizovanog projekta.*

Za svaku tabelu klase *company* kreiran je odgovarajući RESTful servis. Dalje, za svaki RESTful Servis kreirani su *Java* i *JavaScript* klijenti. Za *JavaScript* klijente generisane su *HTML* stranice koje učitavaju njihov kod.

Sledećom slikom je prikazana struktura kompletno urađenog projekta.

Kompletno urađen i testiran projekat je priložen kao dodatni materijal uz materijale ove lekcije - odmah iza sekcije vežbi.



Slika 10.1.3 Konačna struktura projekata RESTfulDemo i RESTClient [izvor: autor]

## ✓ 10.2 SOAP veb servisi sa JAX - RS (25 min)

### POSTAVKA, KREIRANJE PROJEKTA I WEB SERVISA

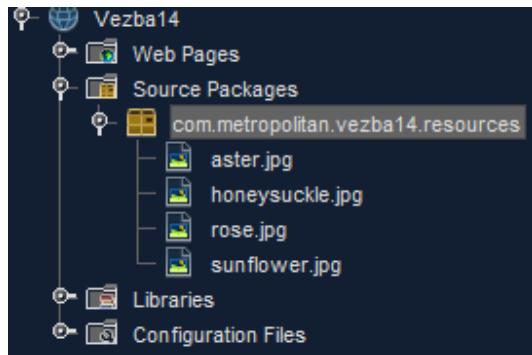
*Kreiranje Java veb projekta sa SOAP veb servisima.*

Zadatak: ([https://netbeans.org/kb/docs/websvc/flower\\_ws.html](https://netbeans.org/kb/docs/websvc/flower_ws.html))

1. Kreirati Veb aplikaciju Vezbe14;
2. Kreirati Veb servis;
3. Implementirati veb servis;
4. Kodirati i testirati veb servis;
5. Modifikovati šemu i WSDL fajl tako da prosleđuju binarne fajlove;
6. Kreirati Java klijent.

REŠENJE:

U razvojnom okruženju NetBeans IDE kreiran je nov projekat pod nazivom Vezba14. U kreiranom projektu biće kreiran paket com.metropolitan.vezba14.resources u kojem će biti čuvane slike.

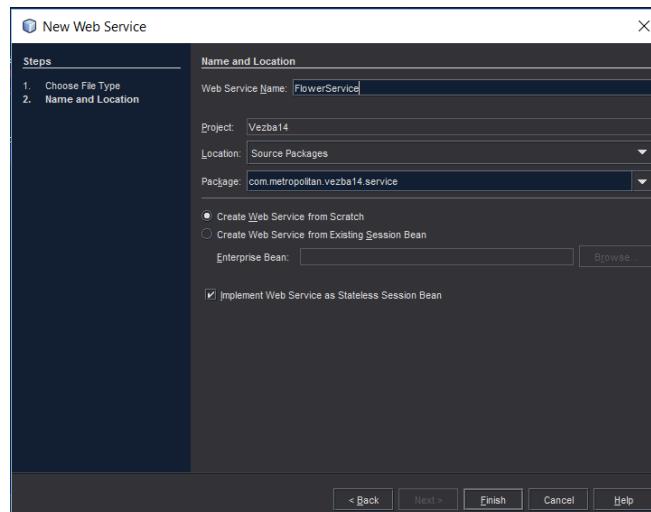


Slika 10.2.1 Slike su dodate u projekat [izvor: autor]

U nastavku će biti kreiran veb servis. Veb servis će posedovati sledeće dve opracije:

- operacija koja preuzima naziv cveta i preuzima njemu odgovarajuću sliku;
- operacija koja preuzima dostupne slike i vraća odgovarajuću listu.

Pristupa se kreiranju zrna sesije bez stanja koje odgovara veb servisu. U NetBeans čarobnjaku potrebno je obezbediti informacije kao na sledećoj slici.

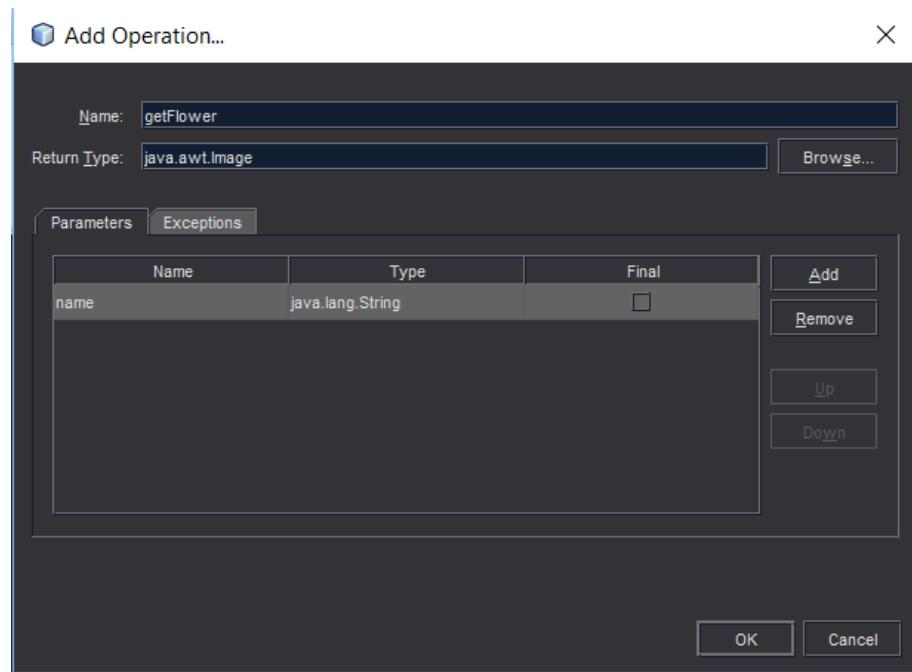


Slika 10.2.2 Kreiranje veb servisa [izvor: autor]

## KREIRANJE VEB SERVISA

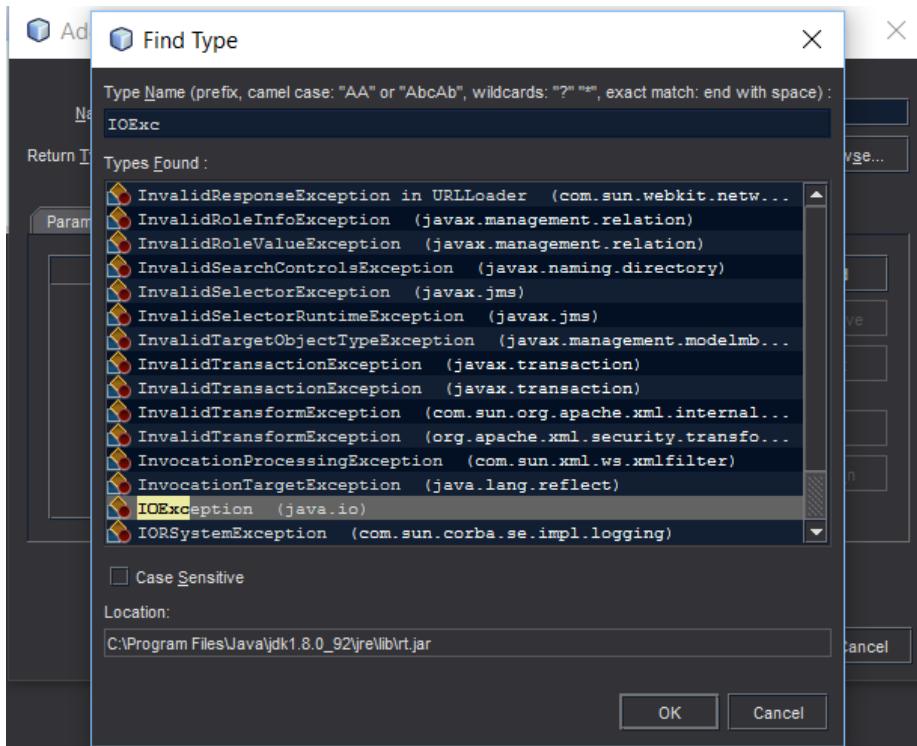
*Kreiranje veb servis je neophodno snabdeti traženim operacijama.*

Iz veb servisa je prvo neophodno obrisati predefinisanu operaciju. Zatim se kreira operacija veb servisa koja preuzima naziv cveta, a vraća odgovarajuću sliku. Kreiranje navedene operacije je prikazano sledećom slikom.



Slika 10.2.3 Definisanje operacije getFlower() [izvor: autor]

Još jedan zadatak će biti obavljen. U Tabu Exception, biće izabran izuzetak IOException na način prikazan sledećom slikom.



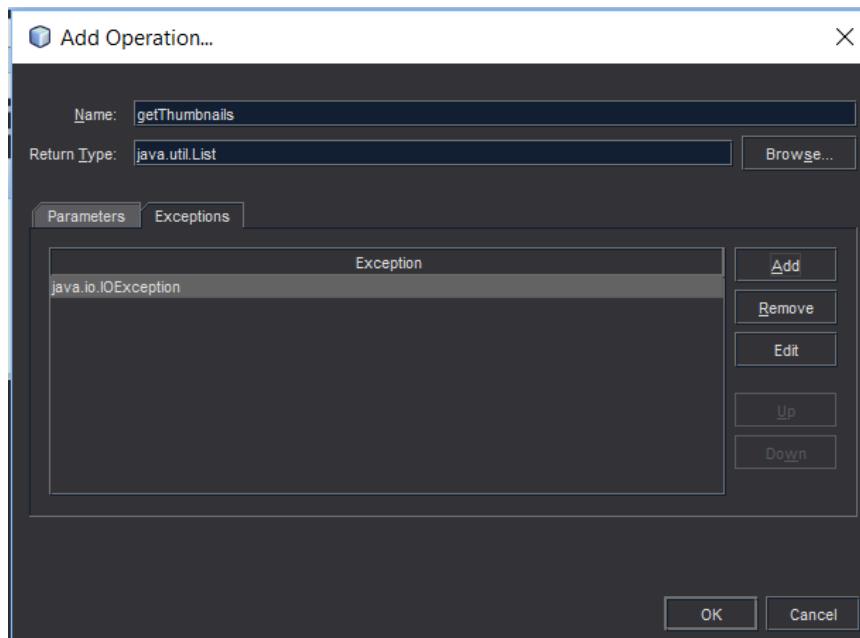
Slika 10.2.4 Definisanje tipa izuzetka [izvor: autor]

## DODAVANJE OPERACIJA SERVISA

*Neophodno je dodati i drugu operaciju servisa.*

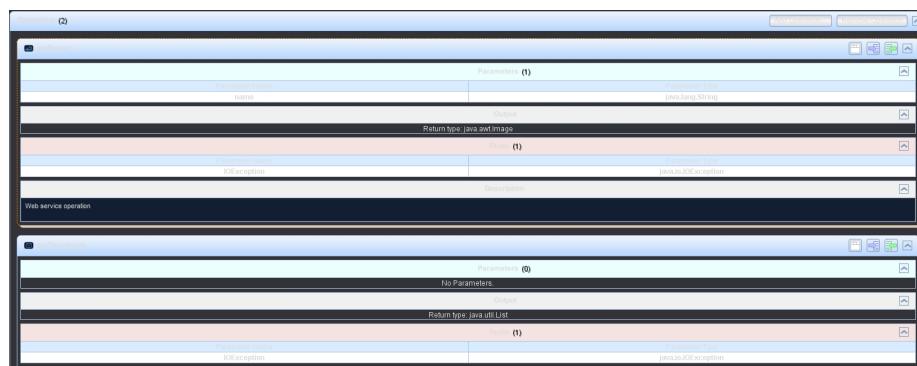
Kreira se druga operacija servisa po sledećim zahtevima:

- naziv operacije: getThumbnails() ;
- povratni tip: java.util.List;
- Izuzetak: IOException.



Slika 10.2.5 Kreiranje druge operacije veb servisa [izvor: autor]

Sledećom slikom su prikazane operacije u dizajneru veb servisa.



Slika 10.2.6 Kreirane su obe operacije [izvor: autor]

## KODIRANJE VEB SERVISA

*Sledi redefinisanje klase veb servisa.*

U klasu veb servisa je neophodno dodati funkcionalnost kojom se preuzima JPG datoteka kao niz bajtova:

```
private byte[] getFlowerBytes(String name) throws IOException {
    URL resource = this.getClass().getResource("/com/metropolitan/vezba14/
```

```
resources+"/"+name+".jpg");
    return getBytes(resource);
}
```

Razvojno okruženje će tražiti da se doda odgovarajuća import instrukcija ([import java.net.URL;](#)) i da se kreira metoda getBytes() na sledeći način:

```
private byte[] getBytes(URL resource) throws IOException {
    InputStream in = resource.openStream();
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    byte[] buf = new byte[1024];
    for(int read; (read = in.read(buf)) != -1;) {
        bos.write(buf, 0, read);
    }
    return bos.toByteArray();
}
```

Nakon dodavanja ovog koda u klasu veb servisa, biće neophodno dodati još odgovarajućih import instrukcija.

Zatim je neophodno kreirati metodu koja čita niz bajtova kao sliku. Metoda getImage() je priložena sledećim listingom:

```
private Image getImage(byte[] bytes, boolean isThumbnail) throws IOException {
    ByteArrayInputStream bis = new ByteArrayInputStream(bytes);
    Object source = bis; // File or InputStream
    ImageInputStream iis = ImageIO.createImageInputStream(source);
    Iterator readers = ImageIO.getImageReadersByFormatName("jpeg");
    ImageReader reader = (ImageReader) readers.next();
    ImageReadParam param = reader.getDefaultReadParam();
    if (isThumbnail) {
        param.setSourceSubsampling(4, 4, 0, 0);
    }
    reader.setInput(iis, true);
    return reader.read(0, param);
}
```

Metoda se dodaje u klasu veb servisa, kao i odgovarajuće import instrukcije.

## REDEFINISANJE OPERACIJA VEB SERVISA

*Sledi redefinisanje kreiranih operacija veb servisa.*

Sledi implementacija metode getFlower() koja je prva kreirana operacija veb servisa:

```
@WebMethod(operationName = "getFlower")
public Image getFlower(@WebParam(name = "name") String name) throws IOException {
    byte[] bytes = getFlowerBytes(name);
    return getImage(bytes, false);
}
```

Pre redefinisanja preostale operacije, biće dodato još par neophodnih linija koda.

Na samom vrhu koda klase naći će se instrukcija kojom se definiše niz stringova naziva pojedinačnih cvetova:

```
private static final String[] FLOWERS = {"aster", "honeysuckle", "rose",  
"sunflower"};
```

Potom će biti dodata i metoda koja kreira ArrayList i dodaje byte niz svakog cveta u listu:

```
private List allFlowers() throws IOException {  
    List flowers = new ArrayList();  
    for (String flower:FLOWERS) {  
        URL resource = this.getClass().getResource("/com/metropolitan/vezba14/  
resources/"+flower+".jpg");  
        flowers.add(getBytes(resource));  
    }  
    return flowers;  
}
```

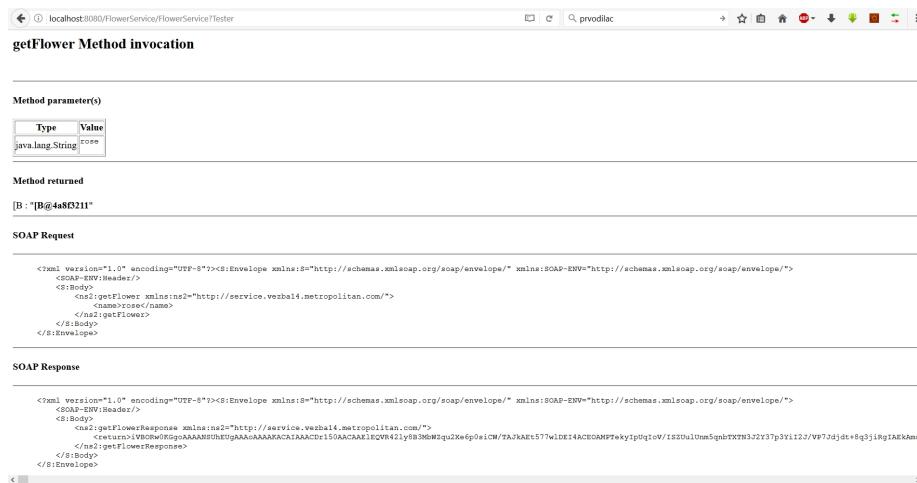
Konačno implementira se i preostala operacija veb servisa:

```
@WebMethod(operationName = "getThumbnails")  
public List<Image> getThumbnails() throws IOException {  
    List<byte[]> flowers = allFlowers();  
    List<Image> flowerList = new ArrayList<Image>(flowers.size());  
    for (byte[] flower : flowers) {  
        flowerList.add(getImage(flower, true));  
    }  
    return flowerList;  
}
```

## TESTIRANJE VEB SERVISA

*Neophodno je izvršiti testiranje kreiranih operacija veb servisa.*

Desnim klikom na veb servis i izborom opcije Test Web Service pristupa se testiranju. Biće unet String "rose" za potrebe testa. Rezultati testiranja su priloženi sledećim slikama:



Slika 10.2.7 Test veb servisa [izvor: autor]



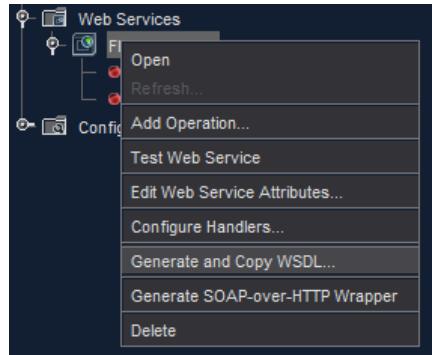
Slika 10.2.8 Test poziva metode veb servisa [izvor: autor]

## MODIFIKOVANJE ŠEME I WDSL ZA PROSLEĐIVANJE BINARNIH PODATAKA

*Sledi dalji razvoj projekta.*

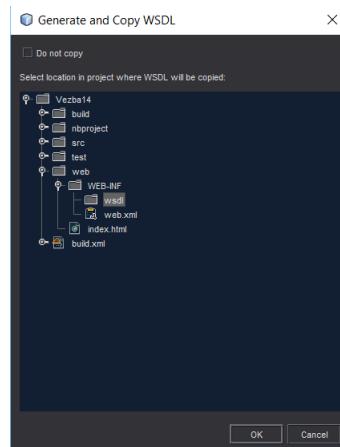
U nastavku razvoja ovog projekta biće akcenat na modifikaciji WSDL datoteke i XML šeme sa ciljem omogućavanja klijentu parsiranja JPEG slike koja je prosleđena kao niz binarnih podataka.

Za početak u **WEB - INF** folderu biće kreiran nov folder pod nazivom **wsdl**. Dalje, neophodno je proširiti čvor Web Services i desnim klikom na čvor FlowerServices omogućiti izbor opcije **Generate and Copy WSDL** (sledeća slika).

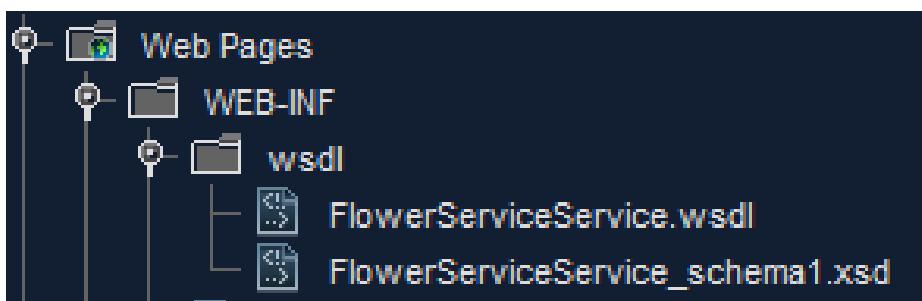


Slika 10.2.9 Izbor opcije Generate and Copy WSDL [izvor: autor]

Za lokaciju datoteka koje će biti kreirane kao rezultat ovog izbora, neophodno je izabrati prethodno kreirani folder **Web - INF/wsdl**. Klikom na dugme OK (Slika 10) počinje generisanje neophodnih datoteka.



Slika 10.2.10 Izbor lokacije za WSDL i XML datoteku [izvor: autor]



Slika 10.2.11 Kreirane tražene datoteke [izvor: autor]

## MODIFIKOVANJE DATOTEKE FLOWERSERVICE\_SCHEMA1.XSD

*Neophodno je modifikovati datoteku FlowerService\_schema1.xsd.*

Neophodno je modifikovati datoteku *FlowerService\_schema1.xsd* tako što će redom biti dodavane sledeće linije koda za *getThumbnailsResponse* i *getFlowerResponse*.

```
<xs:complexType name="getThumbnailsResponse">
    <xs:sequence>
        <xs:element name="return" type="xs:base64Binary" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="getFlowerResponse">
    <xs:sequence>
        <xs:element name="return" type="xs:base64Binary" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

Za oba prethodna elementa neophodno je dodati sledeću liniju:

```
xmime:expectedContentTypes="image/jpeg" xmlns:xmime="http://www.w3.org/2005/05/
xmlmime"
```

Sledi kompletiran kod šeme *FlowerService\_schema1.xsd* :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://service.flower.org/"
xmlns:tns="http://service.flower.org/" xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="IOException" type="tns:IOException"/>

    <xs:element name="getFlower" type="tns:getFlower"/>

    <xs:element name="getFlowerResponse" type="tns:getFlowerResponse"/>

    <xs:element name="getThumbnails" type="tns:getThumbnails"/>

    <xs:element name="getThumbnailsResponse" type="tns:getThumbnailsResponse"/>

    <xs:complexType name="getThumbnails">
        <xs:sequence/>
    </xs:complexType>

    <xs:complexType name="getThumbnailsResponse">
        <xs:sequence>
            <xs:element name="return" type="xs:base64Binary" minOccurs="0"
maxOccurs="unbounded"
                xmime:expectedContentTypes="image/jpeg" xmlns:xmime="http://www.w3.org/2005/
05/xmlmime"/>
        </xs:sequence>
    </xs:complexType>
```

```

</xs:sequence>
</xs:complexType>

<xs:complexType name="IOException">
<xs:sequence>
<xs:element name="message" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="getFlower">
<xs:sequence>
<xs:element name="name" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

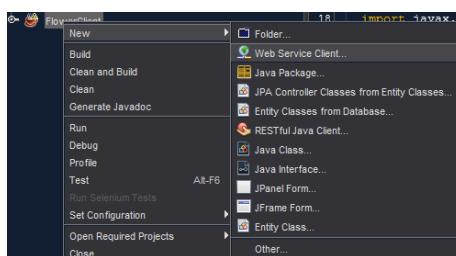
<xs:complexType name="getFlowerResponse">
<xs:sequence>
<xs:element name="return" type="xs:base64Binary" minOccurs="0"
xmime:expectedContentTypes="image/jpeg" xmlns:xmimeType="http://www.w3.org/2005/
05/xmlmime"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

## KREIRANJE SWING KLIJENTA

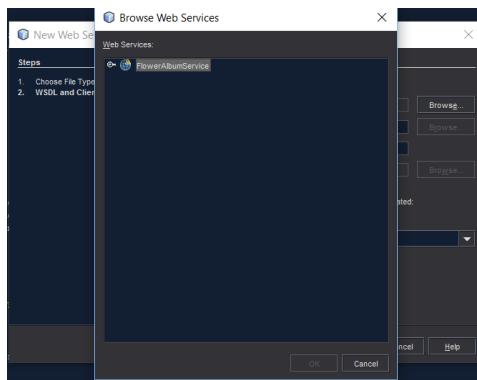
*Sledi kreiranje Java klijenta generisanog veb servisa FlowerService.*

Kreira se nov Java projekat pod nazivom FlowerClient. Desnim klikom na projekat on se registruje kao nov projekat tipa *Web Service Client* (sledeća slika).



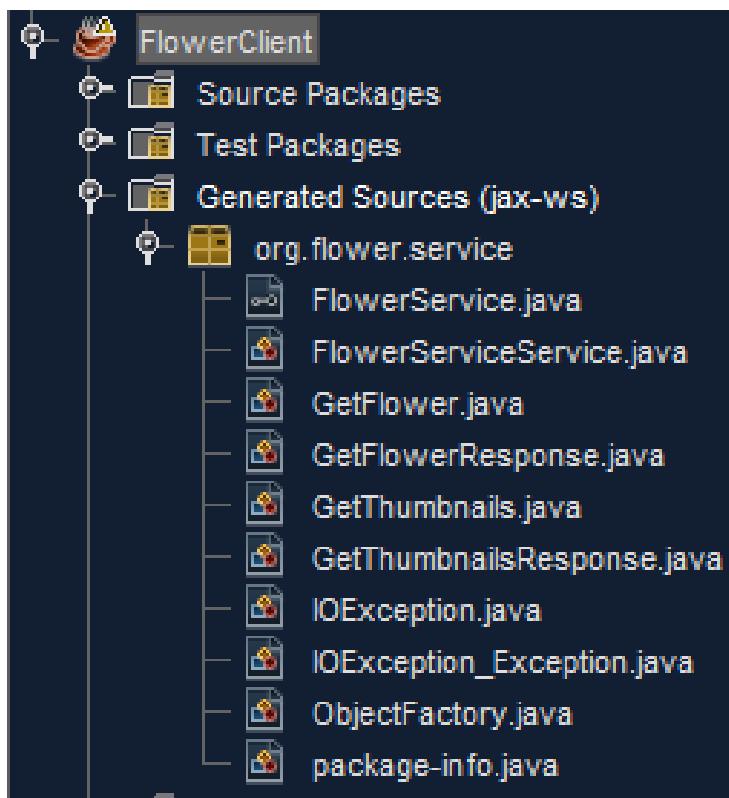
Slika 10.2.12 Nov projekat Web Service Client [izvor: autor]

Otvoriće se prozor u kojem je neophodno izabrati veb servis za koji se klijent kreira. To će biti dobro poznati servis FlowerService. Navedeno je prikazano sledećom slikom.



Slika 10.2.13 Izbor veb servisa za kreirani klijent [izvor: autor]

Konačno, bira se i paket kojem će pripadati klasa klijenta i završava se ovo podešavanje. Sada dolazi do generisanja čitavog niza klasa koje su prikazane sledećom slikom.



Slika 10.2.14 Generisane klase u projektu klijenta [izvor: autor]

## JAVA KLASE SWING KLIJENTA

### *Sledi kodiranje Swing klasa klijenta*

Sada je neophodno kodirati klase: glavnu klasu klijenta i pomoćnu - za implementaciju GUI.

Sledi listing klase FlowerFrame.java za GUI koji će biti ugrađen u glavnu klasu:

```
package flowerclient;
```

```
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.Map;
import javax.swing.ImageIcon;

public class FlowerFrame extends javax.swing.JFrame {

    protected static final String[] FLOWERS = {"aster", "honeysuckle", "rose",
"sunflower"};
    private Map<String, Image> flowers;

    /** Creates new form FlowerFrame */
    public FlowerFrame(Map<String, Image> flowers) {
        this.flowers = flowers;
        for (String flower : FLOWERS) {
            flowers.put(flower, null);
        }
        initComponents();
        setTitle("Garden Flowers [waiting for picture]");

        ItemListener rbListener = new RBListener();
        asterRadioButton.addItemListener(rbListener);
        honeysuckleRadioButton.addItemListener(rbListener);
        roseRadioButton.addItemListener(rbListener);
        sunflowerRadioButton.addItemListener(rbListener);

        ActionListener bListener = new ButtonListener();
        asterButton.addActionListener(bListener);
        honeysuckleButton.addActionListener(bListener);
        roseButton.addActionListener(bListener);
        sunflowerButton.addActionListener(bListener);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        gardenFlowersPanel = new javax.swing.JPanel();
        titleLabel = new javax.swing.JLabel();
        asterRadioButton = new javax.swing.JRadioButton();
        honeysuckleRadioButton = new javax.swing.JRadioButton();
        roseRadioButton = new javax.swing.JRadioButton();
        sunflowerRadioButton = new javax.swing.JRadioButton();
    }
}
```

```
mainScrollPane = new javax.swing.JScrollPane();
mainPanel = new javax.swing.JPanel();
mainPictureButton = new javax.swing.JButton();
thumbnailsScrollPane = new javax.swing.JScrollPane();
thumbnailsPanel = new javax.swing.JPanel();
asterButton = new javax.swing.JButton();
honeysuckleButton = new javax.swing.JButton();
roseButton = new javax.swing.JButton();
sunflowerButton = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

titleLabel.setFont(new java.awt.Font("Tahoma", 1, 18));
titleLabel.setText("Garden Flowers");

buttonGroup1.add(asterRadioButton);
asterRadioButton.setSelected(true);
asterRadioButton.setText("Aster");

buttonGroup1.add(honeysuckleRadioButton);
honeysuckleRadioButton.setText("Honeysuckle");

buttonGroup1.add(roseRadioButton);
roseRadioButton.setText("Rose");

buttonGroup1.add(sunflowerRadioButton);
sunflowerRadioButton.setText("Sunflower");

mainPanel.setLayout(new java.awt.BorderLayout());

mainPictureButton.setText("Waiting for picture...");
mainPanel.add(mainPictureButton, java.awt.BorderLayout.CENTER);

mainScrollPane.setViewportView(mainPanel);

thumbnailsPanel.setLayout(new java.awt.GridLayout(1, 0));

asterButton.setText("Waiting...");
thumbnailsPanel.add(asterButton);

honeysuckleButton.setText("Waiting...");
thumbnailsPanel.add(honeysuckleButton);

roseButton.setText("Waiting...");
thumbnailsPanel.add(roseButton);

sunflowerButton.setText("Waiting...");
thumbnailsPanel.add(sunflowerButton);

thumbnailsScrollPane.setViewportView(thumbnailsPanel);

javax.swing.GroupLayout gardenFlowersPanelLayout = new
javax.swing.GroupLayout(gardenFlowersPanel);
```

```
gardenFlowersPanel.setLayout(gardenFlowersPanelLayout);
gardenFlowersPanelLayout.setHorizontalGroup(
    gardenFlowersPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
    gardenFlowersPanelLayout.createSequentialGroup()
        .addContainerGap(374, Short.MAX_VALUE)
        .addComponent(titleLabel)
        .addGap(209, 209, 209))
    .addGroup(gardenFlowersPanelLayout.createSequentialGroup()
        .addGap(19, 19, 19)
        .addComponent(asterRadioButton)
        .addGap(18, 18, 18)
        .addComponent(honeysuckleRadioButton)
        .addGap(18, 18, 18)
        .addComponent(roseRadioButton)
        .addGap(18, 18, 18)
        .addComponent(sunflowerRadioButton)
        .addContainerGap(393, Short.MAX_VALUE))
    .addGroup(gardenFlowersPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(thumbnailsScrollPane,
    javax.swing.GroupLayout.DEFAULT_SIZE, 704, Short.MAX_VALUE)
        .addContainerGap())
    .addGroup(gardenFlowersPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(mainScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE,
    704, Short.MAX_VALUE)
        .addContainerGap()))
);
gardenFlowersPanelLayout.setVerticalGroup(
    gardenFlowersPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addGroup(gardenFlowersPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(titleLabel)
            .addGap(7, 7, 7))
    .addGroup(gardenFlowersPanelLayout.createSequentialGroup()
        .addComponent(asterRadioButton)
        .addComponent(honeysuckleRadioButton)
        .addComponent(roseRadioButton)
        .addComponent(sunflowerRadioButton))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(mainScrollPane,
    javax.swing.GroupLayout.PREFERRED_SIZE, 324, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(thumbnailsScrollPane,
    javax.swing.GroupLayout.PREFERRED_SIZE, 115, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
```

```
Short.MAX_VALUE))
    );

    javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(gardenFlowersPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(gardenFlowersPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );

    pack();
}// </editor-fold>

/** This is the custom ItemListener class for the radio buttons.
 */
private class RBLListener implements ItemListener {

    public void itemStateChanged(ItemEvent e) {
        showFlower();
    }
}

private class ButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == asterButton) {
            asterRadioButton.setSelected(true);
        } else if (e.getSource() == honeysuckleButton) {
            honeysuckleRadioButton.setSelected(true);
        } else if (e.getSource() == roseButton) {
            roseRadioButton.setSelected(true);
        } else if (e.getSource() == sunflowerButton) {
            sunflowerRadioButton.setSelected(true);
        }
    }
}

void showFlower() {
    Image img = null;
    if (asterRadioButton.isSelected()) {
        img = flowers.get("aster");
        if (img != null) {
            mainPictureButton.setIcon(new ImageIcon(img));
            setTitle("Garden Flowers [Aster]");
        }
    }
}
```

```
        } else if (honeysuckleRadioButton.isSelected()) {
            img = flowers.get("honeysuckle");
            if (img != null) {
                mainPictureButton.setIcon(new ImageIcon(img));
                setTitle("Garden Flowers [Honeysuckle]");
            }

        } else if (roseRadioButton.isSelected()) {
            img = flowers.get("rose");
            if (img != null) {
                mainPictureButton.setIcon(new ImageIcon(img));
                setTitle("Garden Flowers [Rose]");
            }

        } else if (sunflowerRadioButton.isSelected()) {
            img = flowers.get("sunflower");
            if (img != null) {
                mainPictureButton.setIcon(new ImageIcon(img));
                setTitle("Garden Flowers [Sunflower]");
            }

        }
        if (img == null) {
            mainPictureButton.setIcon(null);
            setTitle("Garden Flowers [waiting for picture]");
        } else {
            mainPictureButton.setText("");
        }
    }

void setThumbnails(Map<String, Image> thumbs) {
    Image img = thumbs.get("aster");
    if (img != null) {
        asterButton.setIcon(new ImageIcon(img));
        asterButton.setText("");
    }
    img = thumbs.get("honeysuckle");
    if (img != null) {
        honeysuckleButton.setIcon(new ImageIcon(img));
        honeysuckleButton.setText("");
    }
    img = thumbs.get("rose");
    if (img != null) {
        roseButton.setIcon(new ImageIcon(img));
        roseButton.setText("");
    }
    img = thumbs.get("sunflower");
    if (img != null) {
        sunflowerButton.setIcon(new ImageIcon(img));
        sunflowerButton.setText("");
    }
}
// Variables declaration - do not modify
private javax.swing.JButton asterButton;
private javax.swing.JRadioButton asterRadioButton;
```

```
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JPanel gardenFlowersPanel;
private javax.swing.JButton honeysuckleButton;
private javax.swing.JRadioButton honeysuckleRadioButton;
private javax.swing.JPanel mainPanel;
private javax.swing.JButton mainPictureButton;
private javax.swing.JScrollPane mainScrollPane;
private javax.swing.JButton roseButton;
private javax.swing.JRadioButton roseRadioButton;
private javax.swing.JButton sunflowerButton;
private javax.swing.JRadioButton sunflowerRadioButton;
private javax.swing.JPanel thumbnailsPanel;
private javax.swing.JScrollPane thumbnailsScrollPane;
private javax.swing.JLabel titleLabel;
// End of variables declaration
}
```

Sledi listing klase Main.java:

```
package org.flower.service;

import java.awt.Image;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;
import javax.imageio.ImageIO;
import javax.imageio.ImageReadParam;
import javax.imageio.ImageReader;
import javax.imageio.stream.ImageInputStream;

/**
 *
 * @author jeff
 */
@WebService(wsdlLocation = "WEB-INF/wsdl/FlowerServiceService.wsdl")
@Stateless()
public class FlowerService {

    private static final String[] FLOWERS = {"aster", "honeysuckle", "rose",
"sunflower"};

    /**
     * Web service operation
     */
}
```

```
@WebMethod(operationName = "getFlower")
public Image getFlower(@WebParam(name = "name") String name) throws IOException
{
    byte[] bytes = getFlowerBytes(name);
    return getImage(bytes, false);
}

/**
 * Web service operation
 */
@WebMethod(operationName = "getThumbnails")
public List<Image> getThumbnails() throws IOException {
    List<byte[]> flowers = allFlowers();
    List<Image> flowerList = new ArrayList<Image>(flowers.size());
    for (byte[] flower : flowers) {
        flowerList.add(getImage(flower, true));
    }
    return flowerList;
}

private byte[] getFlowerBytes(String name) throws IOException {
    URL resource = this.getClass().getResource("/org/flower/resources/" + name
+ ".jpg");
    return getBytes(resource);
}

private byte[] getBytes(URL resource) throws IOException {
    InputStream in = resource.openStream();
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    byte[] buf = new byte[1024];
    for (int read; (read = in.read(buf)) != -1;) {
        bos.write(buf, 0, read);
    }
    return bos.toByteArray();
}

private Image getImage(byte[] bytes, boolean isThumbnail) throws IOException {
    ByteArrayInputStream bis = new ByteArrayInputStream(bytes);
    Iterator readers = ImageIO.getImageReadersByFormatName("jpeg");
    ImageReader reader = (ImageReader) readers.next();
    Object source = bis; // File or InputStream
    ImageInputStream iis = ImageIO.createImageInputStream(source);
    reader.setInput(iis, true);
    ImageReadParam param = reader.getDefaultReadParam();
    if (isThumbnail) {
        param.setSourceSubsampling(4, 4, 0, 0);
    }
    return reader.read(0, param);
}

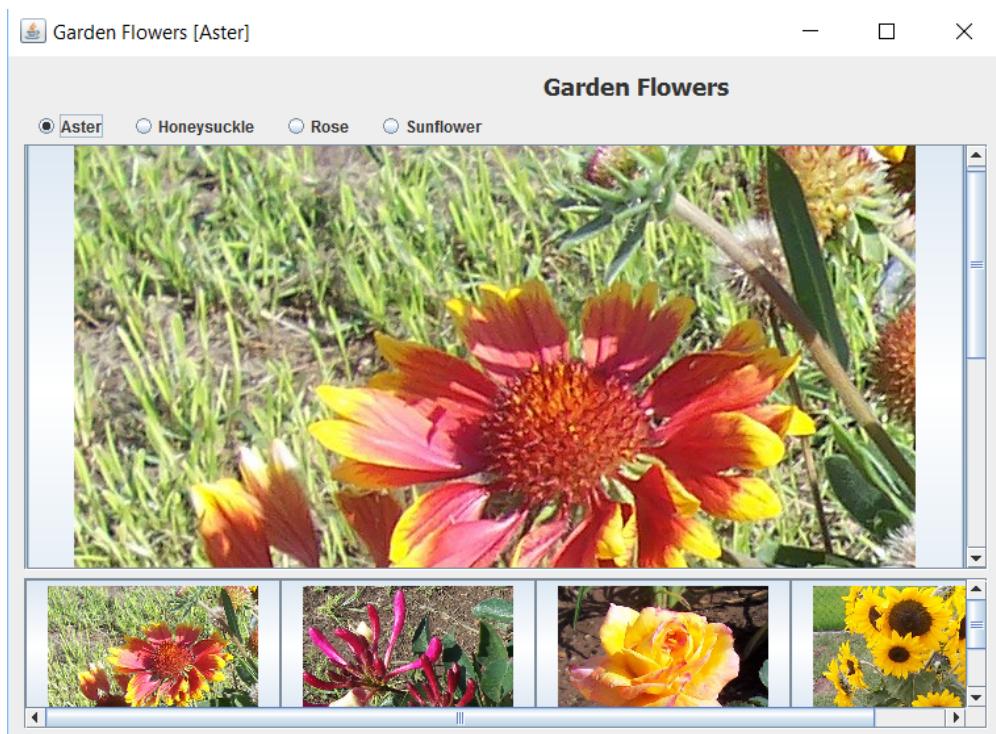
private List<byte[]> allFlowers() throws IOException {
    List<byte[]> flowers = new ArrayList<byte[]>();
    for (String flower : FLOWERS) {
```

```
        URL resource = this.getClass().getResource("/org/flower/resources/" +  
flower + ".jpg");  
        flowers.add(getBytes(resource));  
    }  
    return flowers;  
}  
}
```

## DEMO

*Konačno pokretanje kreirane aplikacije.*

Nakon prevođenja, veb server se postavlja na Deploy, a klijent se startuje. Ako je sve urađeno na pravi način, dobija se sledeći rezultat.



Slika 10.2.15 Rezultat izvršavanja kreirane aplikacije [izvor: autor]

## ▼ Poglavlje 11

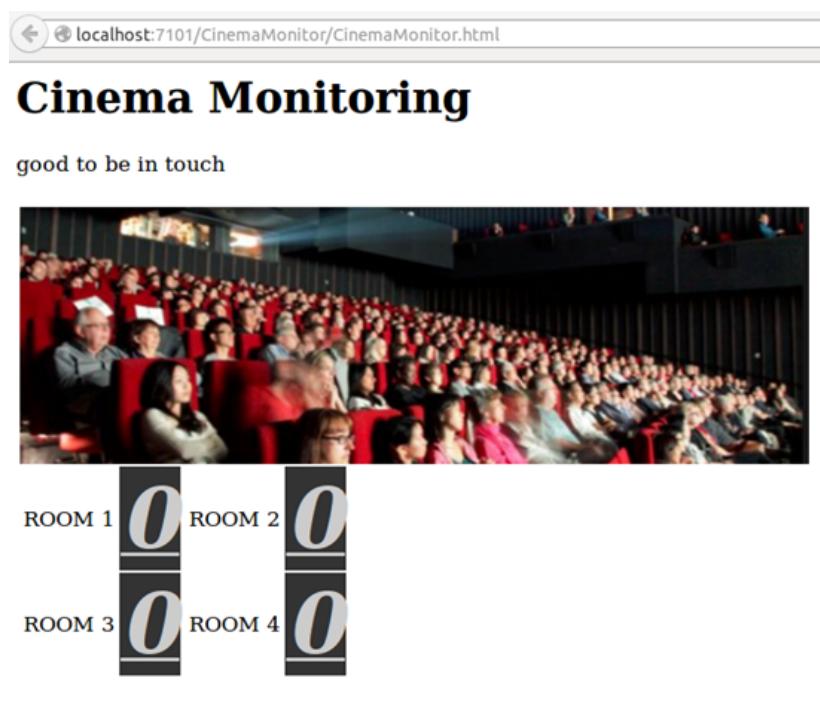
### Individualna vežba 14 (135 min)

#### INDIVIDUALNA VEŽBA 1 (90 MIN)

*Pokušajte samostalno da odradite aplikaciju za praćenje posećenosti bioskopskih sala.*

Pokušajte samostalno da odradite aplikaciju za praćenje posećenosti bioskopskih sala. Zadatak je rađen korak po korak i detaljno je objašnjen na linku <https://technology.amis.nl/2015/05/14/java-web-application-sending-json-messages-through-websocket-to-html5-browser-application-for-real-time-push/>.

Kada se pokrene aplikacija, monitoring se vrši u HTML stranici na način prikazan sledećom slikom.



Slika 11.1 Aplikacija za monitoring posećenosti bioskopskih sala [izvor: autor]

#### INDIVIDUALNA VEŽBA 2 (45 MIN)

*Pokušajte da kreirate Web aplikaciju koja koristi SOAP servise.*

Pokušajte da kreirate Web aplikaciju koja koristi SOAP servise i odgovarajući klijent po uzoru na aplikaciju sa sledećeg linka: <http://www.journaldev.com/9123/jax-ws-tutorial>

Obaviti testiranje kreiranog veb servisa (njegovih operacija) na način koji je demonstriran na predavanjima i vežbama.

Pokrenuti aplikaciju i proveriti korektnost urađenih razvojnih zadataka.

## ✓ Poglavlje 12

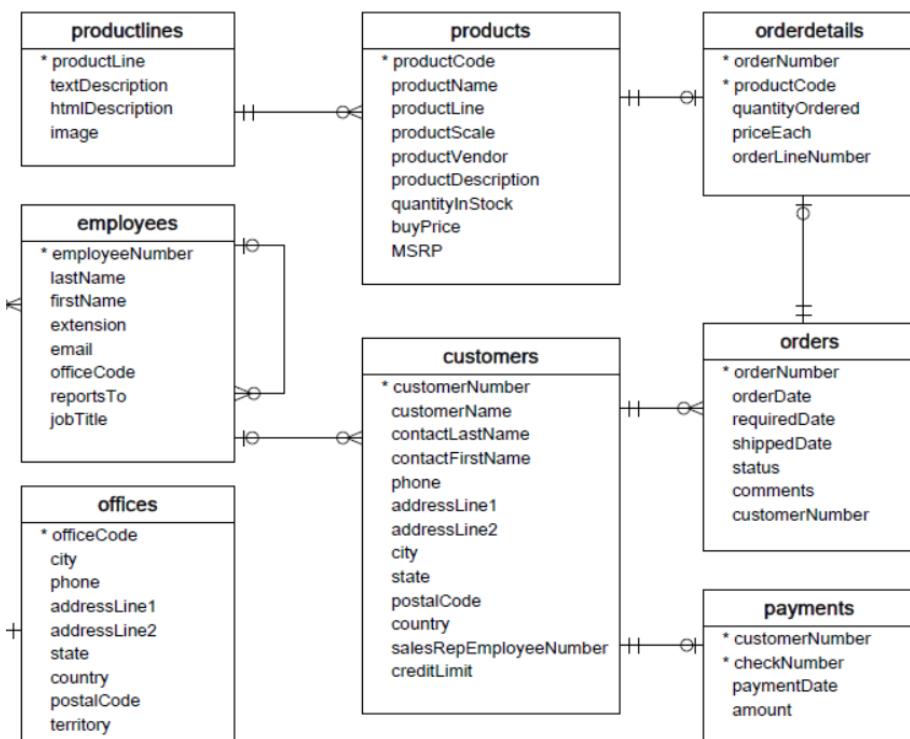
### Domaći zadatak 13 (180 min)

#### ZADATAK 1 (90 MIN)

*Uradite domaći zadatak*

Zadatak:

1. Kreirati projekat DZ13;
2. Nad šemom baze podataka prikazane Slikom 1, generisati RESTful web servise;
3. Za sve RESTful web servise kreirati Java klijente;
4. Za sve RESTful web servise kreirati JavaScript klijente - koristiti novi projekat kao u predavanjima;
5. Za JavaScript klijente generisati odgovarajuće HTML stranice;
6. Testirati urađenu aplikaciju.



Slika 12.1 Šema baze podataka za domaći zadatak [izvor: autor]

#### ZADATAK 2 (90 MIN)

*Uradite za domaći rad.*

Kreirati veb aplikaciju po sopstvenom izboru koja koristi SOAP servise i čiji klijent koristi JSF ili JSP stranice za pozivanje operacija veb servisa. Na sledećim linkovima možete pronaći pomoć i / ili ideju za rešavanje ovog domaćeg zadatka:

1. [http://www.javamission.com/webservices\\_client\\_jsf.html](http://www.javamission.com/webservices_client_jsf.html)
2. <https://netbeans.org/kb/docs/websvc/jax-ws.html>
3. <https://www.baeldung.com/jax-ws>

Nakon urađenih obaveznih zadataka, studenti dobijaju različite zadatke na email od predmetnog asistenta.

## ▼ Poglavlje 13

### Zaključak

## ZAKLJUČAK

*Lekcija 13 se bavila RESTful veb servisima u Java EE platformi.*

Lekcija 13 se bavila RESTful veb servisima u Java EE platformi. Posebno je istaknuto da REST (Representational State Transfer) predstavlja arhitekturni stil u kojem su veb servisi reprezentovani kao resursi i mogu biti identifikovani preko jedinstvenih identifikatora resursa (URI - Uniform Resource Identifiers).

U uvodnom delu lekcija ističe da su Veb servisi, koji su razvijeni primenom REST stila, poznati kao RESTful veb servisi i da je uvođenjem Java EE 6 platforme predstavljena je i nova Java podrška za RESTful veb servise preko novog Java API za RESTful veb servise (Java API for RESTful Web Services) pod nazivom JAX-RS. Dalje je istaknuto da je u početku JAX-RS je bio dostupan kao samostalni API, da bi nakon izvesnog vremena postao integralni deo Java EE 6 specifikacije.

Jedan od opštih načina primene RESTful veb servisa jeste njihovo ponašanje kao frontend -a za bazu podataka. To praktično znači, klijent RESTful veb servisa koristi RESTful veb servis za izvođenje CRUD (create, read, update i delete) operacija nad bazom podataka. Posebno značajnu pomoć implementaciji RESTful web servisa, u savremenim Java veb aplikacijama, obezbeđuju savremena razvojna okruženja. Pomoću razvojnog okruženja NetBeans IDE obezbeđena je podrška za kreiranje RESTful veb servisa u nekoliko jednostavnih koraka.

Lekcija se posebno fokusirala na nekoliko pažljivo odabralih tema, pri čemu je izlaganje bilo podržano pažljivo izabranim primerima:

- Generisanje RESTful veb servisa iz postojeće baze podataka;
- Testiranje RESTful veb servisa pomoću alata koje obezbeđuje razvojno okruženje NetBeans IDE;
- Generisanje RESTful Java klijent koda RESTful veb servisa;
- Generisanje RESTful JavaScript klijenata RESTful veb servisa;

Savladavanjem ove lekcije student je osposobljen u potpunosti razume i koristi RESTful veb servise u JAVA EE aplikacijama.

## ZAKLJUČAK - NASTAVAK

*Lekcija se bavila analizom i diskusijom problematike SOAP veb servisa primenom JAX - WS.*

Lekcija 13 se, takođe, bavila analizom i diskusijom problematike SOAP veb servisa primenom JAX - WS.

U uvodnom izlaganju je istaknuto da distribuirani veb servisi dozvoljavaju razvoj funkcionalnosti kojima se pristupa preko mreže. Zatim je naglašeno da, ono što razlikuje veb servise od sličnih tehnologija, poput EJB (*Enterprise Java Beans*) ili poziva udaljenih metoda (*RMI - Remote Method Invocation*), jeste to da veb servisi ne zavise od programskog jezika i / ili platforme. To praktično znači da veb servisu, koji je razvijen pomoću programskog jezika Java, moguće je pristupiti klijent programima koji su kreirani upotrebom nekog drugog programskog ili skript jezika i obrnuto.

U lekcijama koje slede nakon uvodnog izlaganja, za analizu i demonstraciju problematike SOAP veb servisa primenom JAX - WS, pažljivo su izabrane teme koje je ova lekcija obrađivala. Posebno je bilo insistirano na konkretnim primerima koji su predstavljali podršku analizi i diskusiji koja je bila vođena u lekciji.

Među brojnim atraktivnim tema, iz konkretne problematike, Lekcija 13 se fokusirala na sledeće izabrane teme:

- Uvod u veb servise;
- Kreiranje jednostavnog veb servisa;
- Testiranje i razvoj klijenta za veb servis;
- Enterprise Java zrno kao veb servis;
- Implementiranje novog veb servisa kao EJB;
- Kreiranje veb servisa iz postojećeg WSDL.

Savladavanjem ove lekcije student je u potpunosti osposobljen da razume, kreira i koristi SOAP veb servise u JAVA EE aplikacijama.

## LITERATURA

*Za pripremu lekcije korišćena je najnovija literatura.*

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. Heffelfinger. 2015. Java EE7 Development With NetBeans 8, PACK Publishing
3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh JUneau. 2015. Java EE7 Recipes, Apress
5. <https://technology.amis.nl/2015/05/14/java-web-application-sending-json-messages-through-websocket-to-html5-browser-application-for-real-time-push/> .
6. [https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)
7. [https://netbeans.org/kb/docs/websvc/flower\\_ws.html](https://netbeans.org/kb/docs/websvc/flower_ws.html)
8. <http://www.mkyong.com/tutorials/jax-ws-tutorials/>
9. <https://www.baeldung.com/jax-ws>
10. <https://www.theserverside.com/news/1365535/EJB-21-Web-Services-Part-1>