PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

# KI204 - JAVA 7: JAVA ENTERPRISE EDITION

## RESTFul veb servisi sa JAX - RS

Lekcija 05

PRIRUČNIK ZA STUDENTE

# KI204 - JAVA 7: JAVA ENTERPRISE EDITION

## Lekcija 05

## *RESTFUL VEB SERVISI SA JAX - RS*

- RESTFul 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 JavaSrcipt RESTFul klijenta
- Poglavlje 5: Domaći zadatak 4
- Zaključak

# UVOD

*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 svetlu, lekcija će se posebno fokusirati na nekoliko pažljivo odabranih 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 kodaRESTfulveb servisa;
- Generisanje RESTful JavaScript klijenataRESTful veb servisa;

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

# Poglavlje 1

# Generisanje RESTFul servisa iz baze podataka

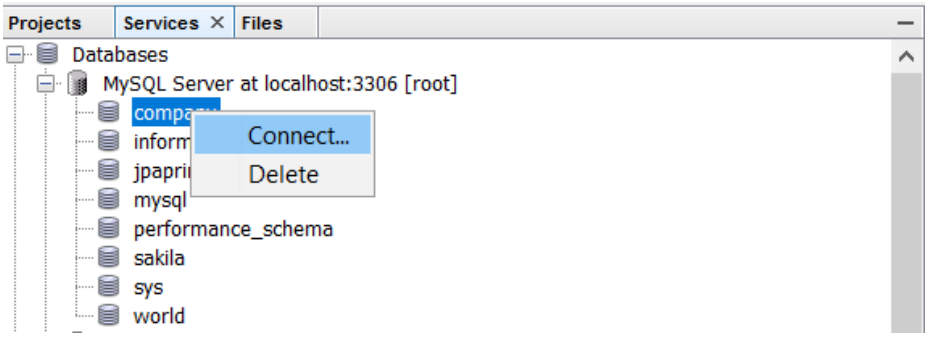
## NETBEANS IDE ČAROBNJAK ZA GENERISANJE RESTFUL SERVISA

*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.

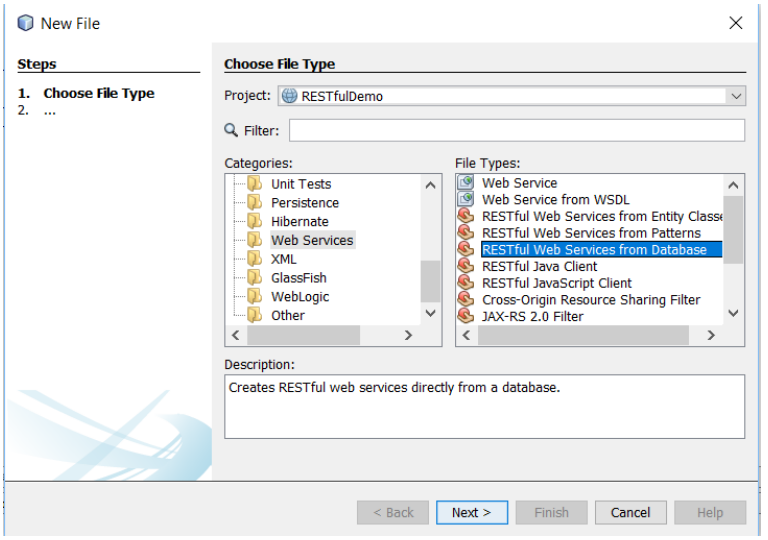



Slika 1.1 Povezivanje na postojeću bazu podataka

# PRIMER 1 - KREIRANJE RESTFUL SERVISA IZ POSTOJEĆE BAZE PODATAKA

*Razvija se konkretan primer za RESTFul veb servise*

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.

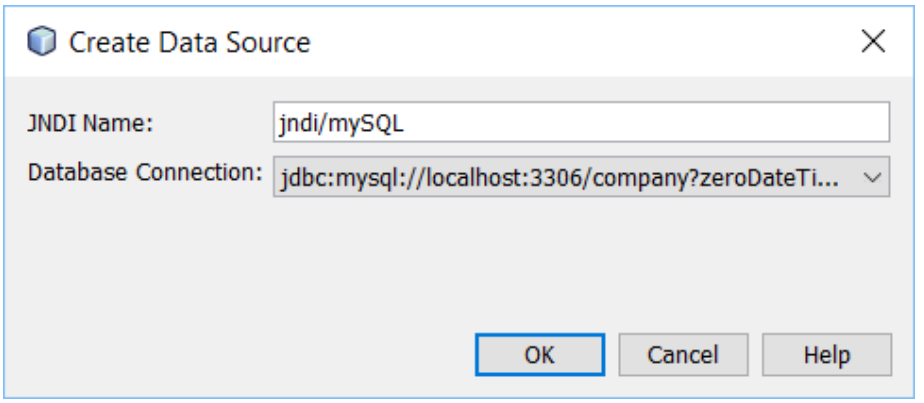



Slika 1.2 Izbor opcije RESTful Web Services From Database

# PRIMER 1 - IZBOR TABELA IZ BAZE PODATAKA U RAZVOJNOM OKRUŽENJU

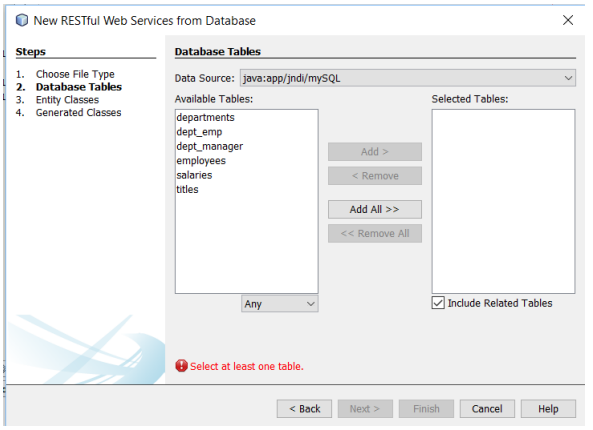
*U nastavku je neophodno izabrati tabele nad kojima će biti kreirane klase RESTful servisa.*

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.

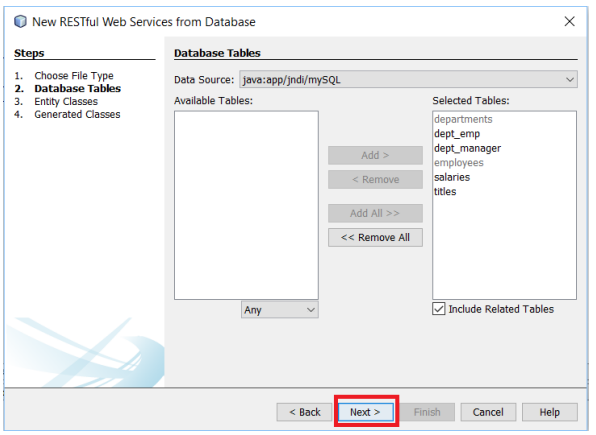



Slika 1.3 Podešavanje baze podataka kao izvora podataka

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

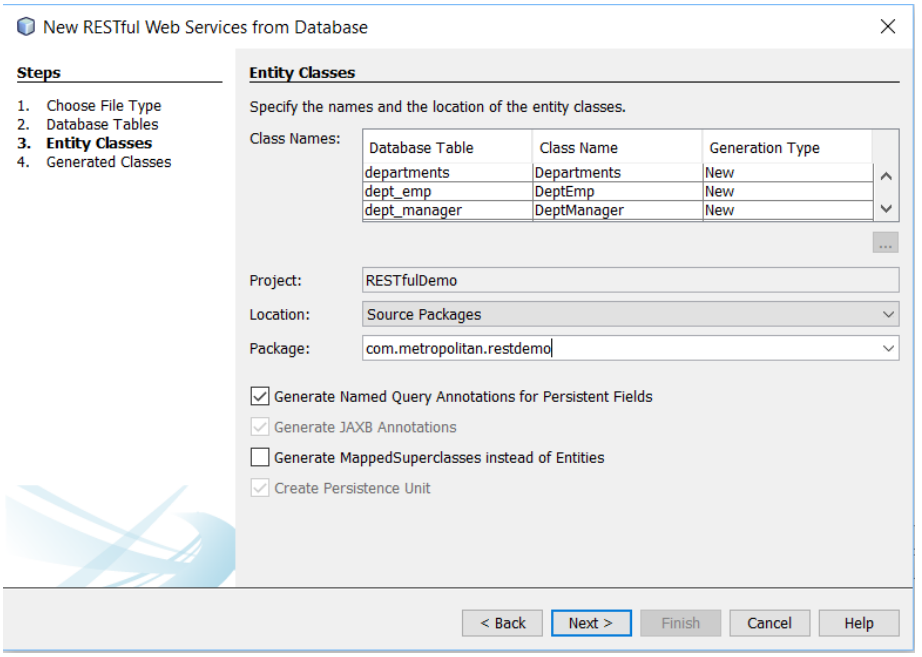



Slika 1.5 Izabrane tabele za RESTful servise

# PRIMER 1 - PODEŠAVANJE KLASA RESTFUL SERVISA

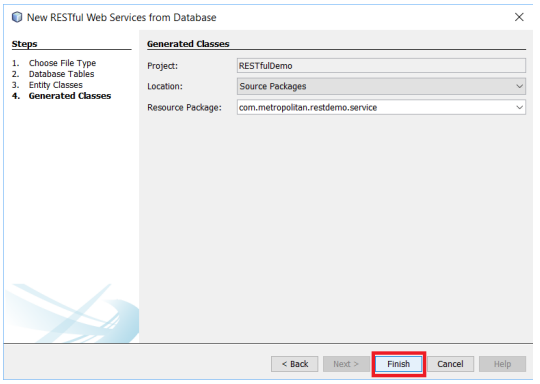
*U NetBeans IDE okruženju je neophodno dodatno podesiti klase koje su u fazi kreiranja.*

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.



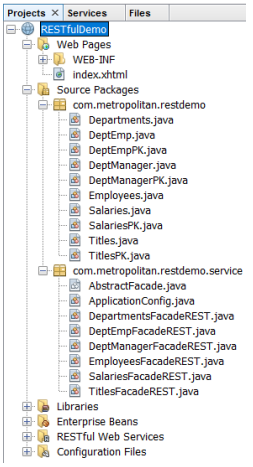

Slika 1.6 Podešavanje novih RESFful servisa

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

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




Slika 1.8 Trenutna struktura projekta RESTfulDemo

# ANALIZA GENERISANOG KODA RESTFUL SERVISA - 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 izabrano par reprezentativnih datoteka, dok će ostale biti priložene u sekciji V*ežbe* 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
&amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp; other.empNo !=
null) || (this.empNo != null
&amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;
!this.empNo.equals(other.empNo))) {
            return false;
        }
        return true;
    }

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

}
```

# ANALIZA GENERISANOG KODA RESTFUL SERVISA - ABSTRACTFACADE 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.

# ANALIZA GENERISANOG KODA RESTFUL SERVISA - RESTFUL KLASE

## *Konačno, sledi analiza koda RESTful klasa.*

> 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.***

# ZADATAK 1

## *Vežbanje kreiranja RESTFul servisa*

Nad jednostavnom bazom podataka, koju ćete samostalno kreirati, kreirajte RESTFul servis.

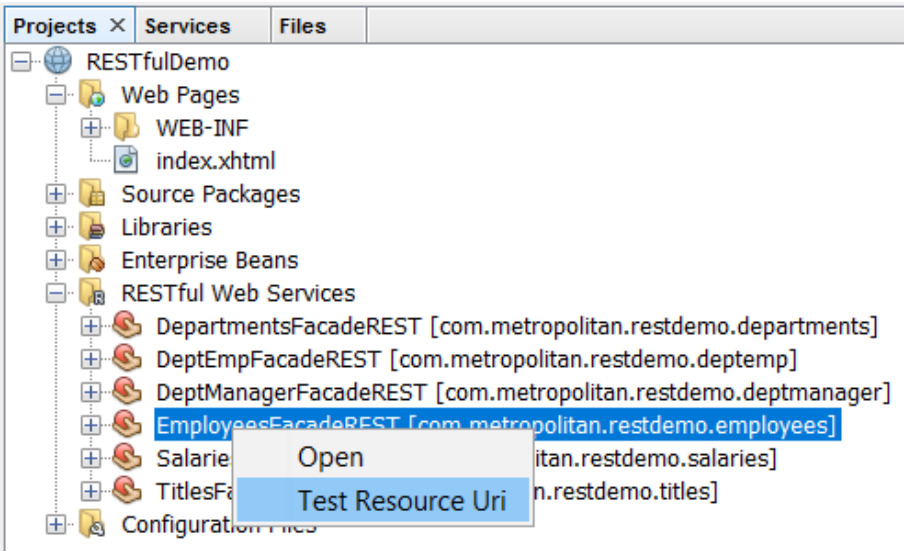
# Poglavlje 2

# Testiranje RESTful veb servisa

## TEST RESOURCE URI

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

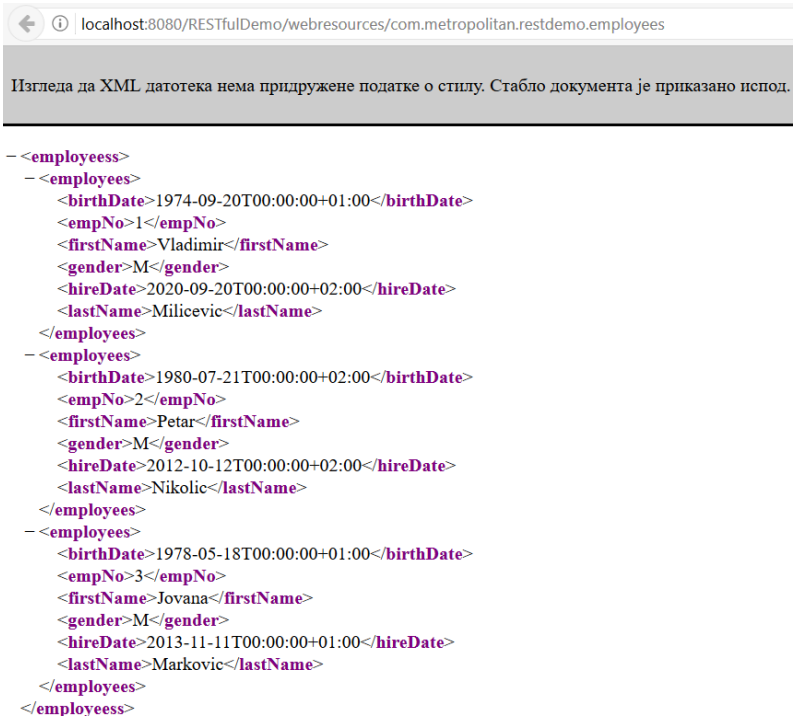
Nakon što je razvijen veb projekat primera, moguće je izvršiti proveru da li su i *RESTful* veb servisi kreirani na adekvatan način. Procedura provere započinje tako što se izvrši desni klik mišem na odgovarajući RESTful veb servis, u folderu *RESTFulWeb Services,* a zatim izabere opcija Test Resource 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

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.

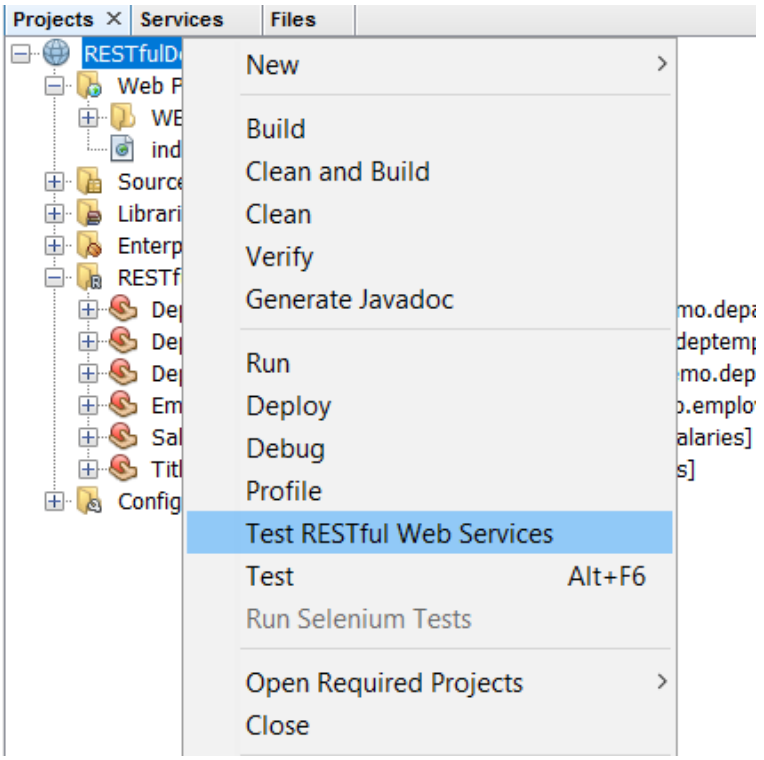



Slika 2.2 Generisani XML kao odgovor na metodu findAll()

# PRIMER 2 - TEST RESTFUL WEB SERVICES

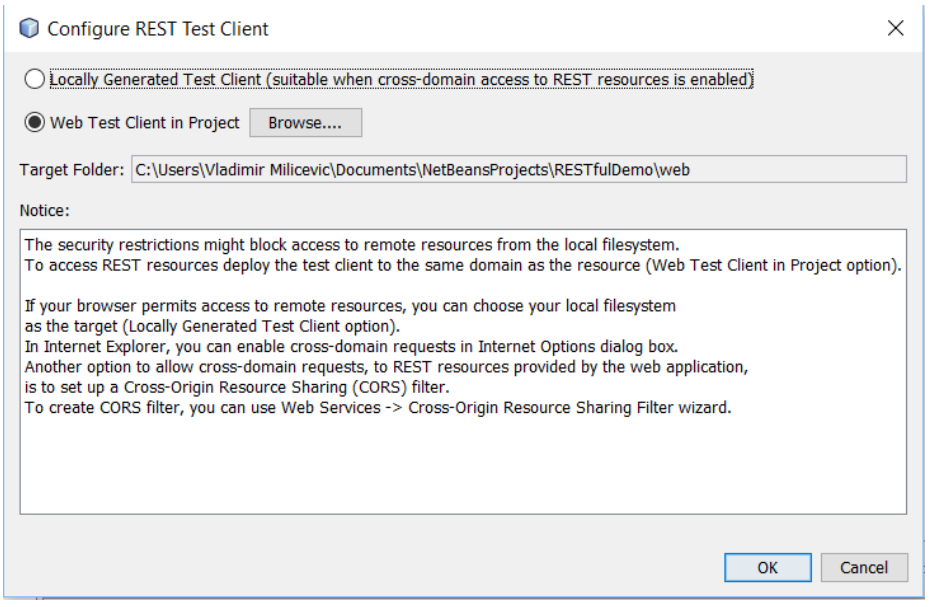
*Sledeće testiranje započinje izborom opcije Test RESTful Web Services.*

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 izabrati opciju Test RESTful Web Services. Navedeno je moguće ilustrovati sledećom slikom.




Slika 2.3 Izbor opcije za testiranje Test RESTful Web Services

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



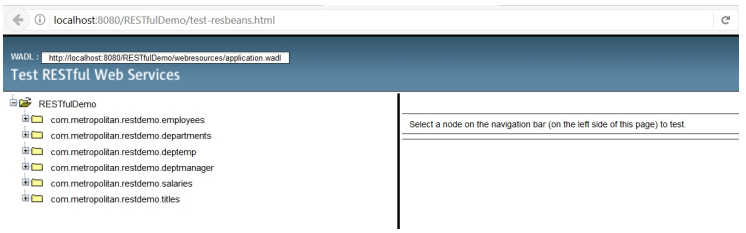

Slika 2.4 Podešavanje RESTful test klijenta

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

# STRANICA TEST RESTFUL WEB SERVICES

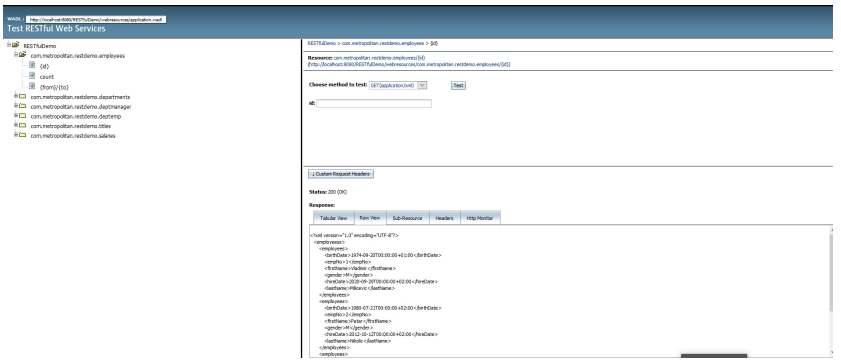
*Nakon izbora veb test klijenta otvoriće se u veb pregledaču stranica Test RESTful Web Services.*

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



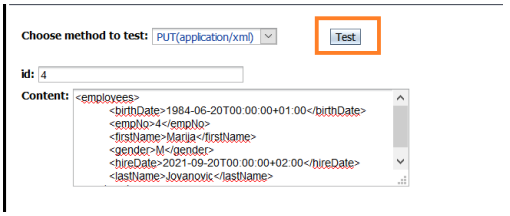

Slika 2.5 Stranica Test RESTful Web Services

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 veb servisu* i vraća se *XML* ili *JSON* odgovor, respektivno.



Slika 2.6 Vraćanje XML odgovora

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

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


Slika 2.8 Test nakon unetog novog zapisa

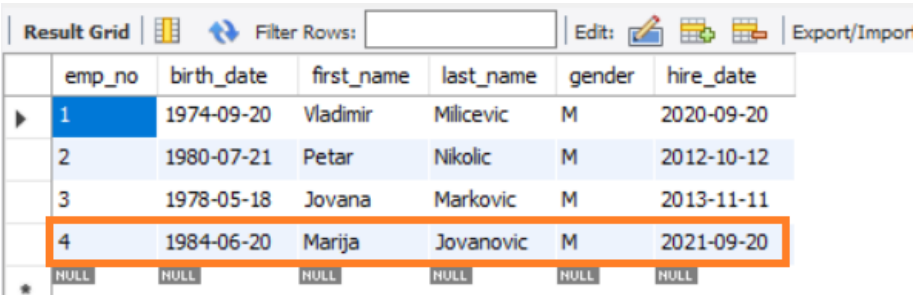
# KLASA APPLICATIONCONFIG.JAVA

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

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

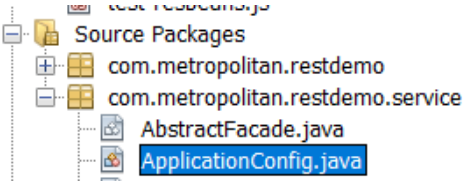
```
select * from employees
```

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




Slika 2.9 Dodatna provera u MySQL Workbench okruženju

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 0 Položaj klase ApplicationConfig.java u hijerarhiji projekta

```
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 veb 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);
    }
```

# ZADATAK 2

## *Vežbanje testiranja RESTFul servisa*

Na način prikazan u ovom delu lekcije, testirajte kreirani RESTFul servis u Zadatku 1.

# Poglavlje 3

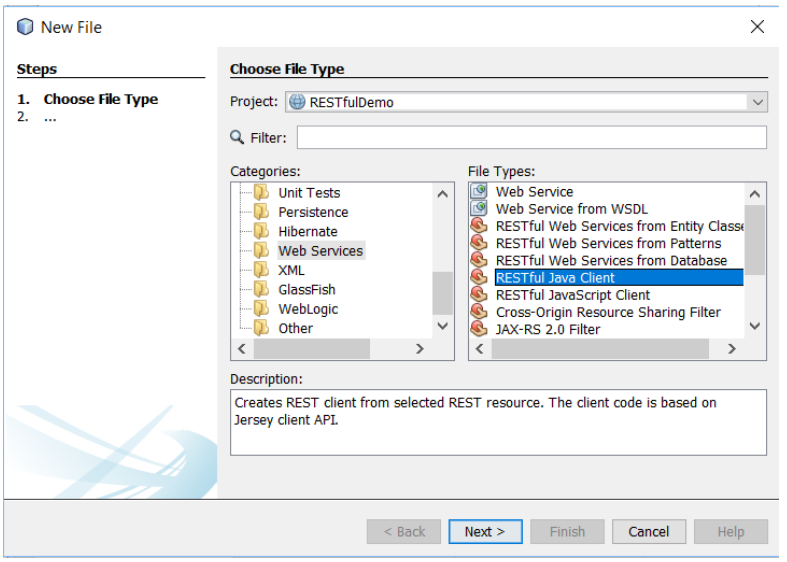
# Generisanje RESTful Java klijent koda

## GENERISANJE DATOTEKE RESTFUL JAVA KLIJENT KODA

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

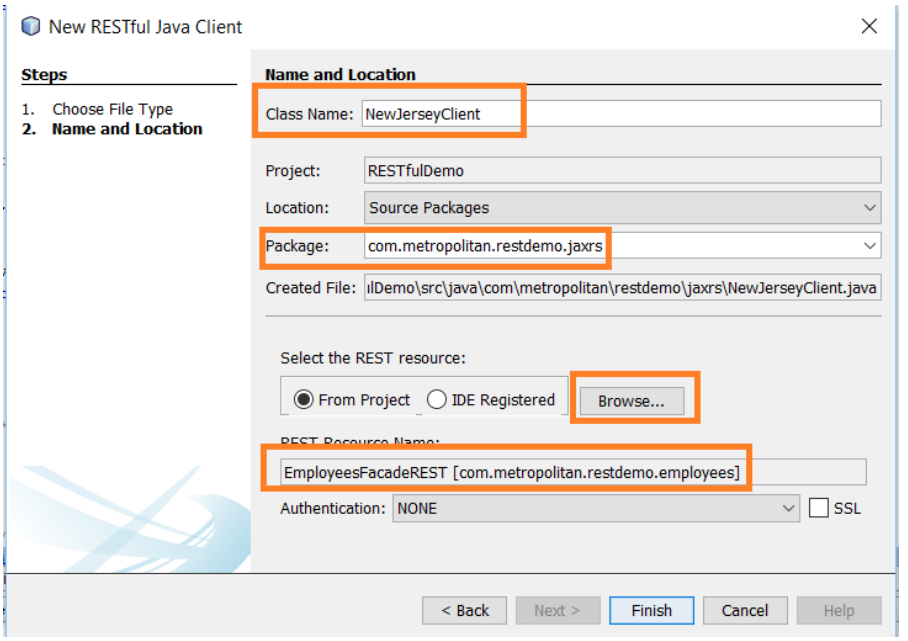
Razvojno okruženje *NetBeans IDE* obezbeđuje čarobnjak za automatsko generisanje Java klijent koda *RESTful servisa*. Ovaj kod ima zadatak da poziva metode *RESTful servisa* na osnovu odgovarajućeg *HTTP zahteva*.

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

U sledećem koraku čarobnjaka, neophodno je obezbediti naziv klasi, i njenom paketu, *JAX-RS* klijenta. U ov om 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

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

# PRIMER 3 - 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 uNetBeans IDE, ovo razvojno okruženje, po podrazumevanim podešavanjima, 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* dobile 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;
    }

}
```

# TESTIRANJE KODA KLIJENTA RESTFUL SERVISA - 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.

# TESTIRANJE KODA KLIJENTA RESTFUL SERVISA - 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 konkretan 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.

Result Grid | Filter Rows: | Edit: | Export/Impor

| 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

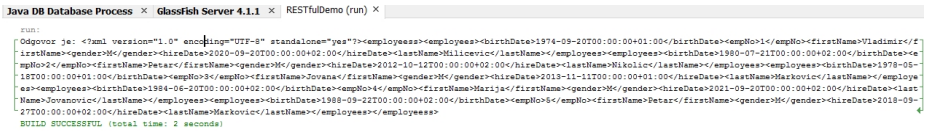
# TESTIRANJE KODA KLIJENTA RESTFUL SERVISA - PRIKAZIVANJE SVIH ZAPISA

*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 koje 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*.




Slika 3.5 Testiranje metode findAll_XML() koda klijenta

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"?>
<employeess>
    <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>
        <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>
</employeess>
```

# ZADATAK 3

## *Kreiranje i testiranje Java klijenta.*

Nastavite vaš primer iz Zadataka 1 i 2, kreiranjem i testiranjem Java klijenta za kreirani RESTFul servis.

# Poglavlje 4

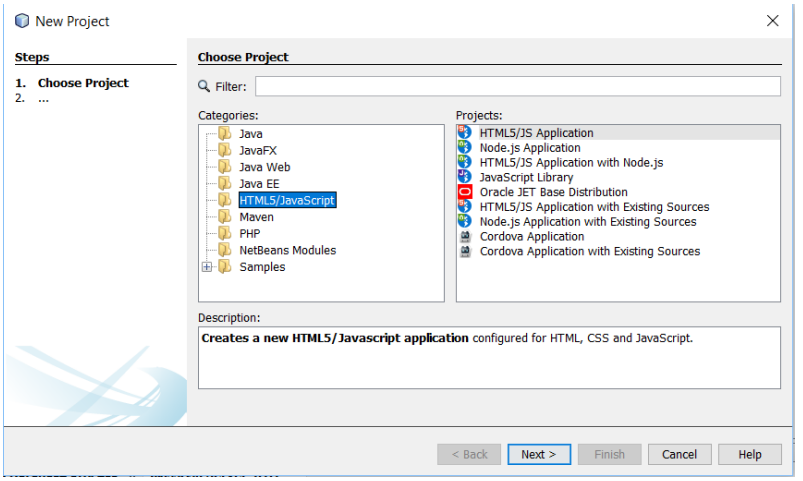
# Generisanje JavaSrcipt RESTFul klijenta

## KREIRANJE PROJEKTA JAVASRCIPT RESTFUL KLIJENTA

*Poželjno je JavaSrcipt RESTFul klijente kreirati u posebnom projektu.*

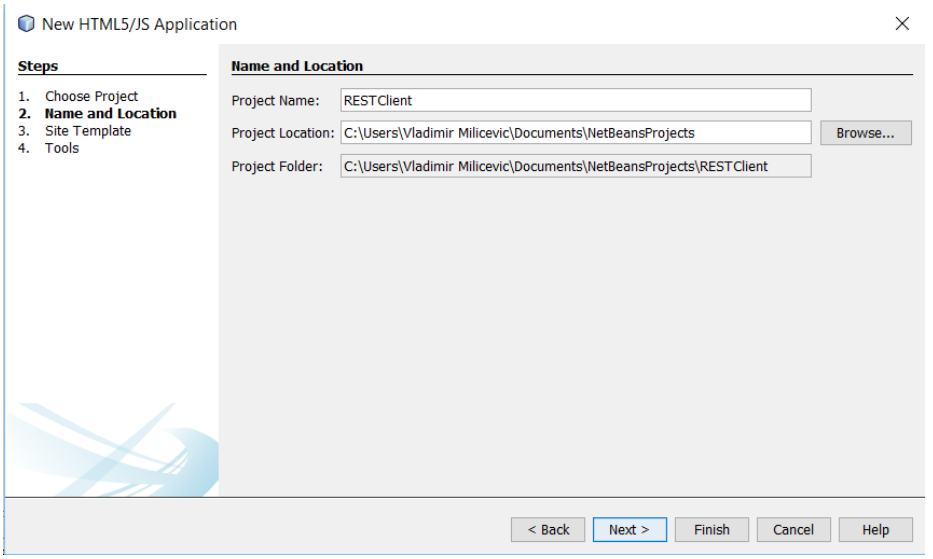
U prethodnom izlaganju je pokazano kako se u razvojnom okruženju NetBeans IDE kreiraju *Java klijenti RESTful servisa*. U opštem slučaju, u savremenim aplikacijama, uporedno se razvijaju*RESTful servisi*, *Java klijenti RESTful servisa* i *JavaScript klijenti RESTful servisa* koji se izvršavaju u veb 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 4.1.1 Kreiranje projekta iz kategorije HTML5 / JavaScript

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



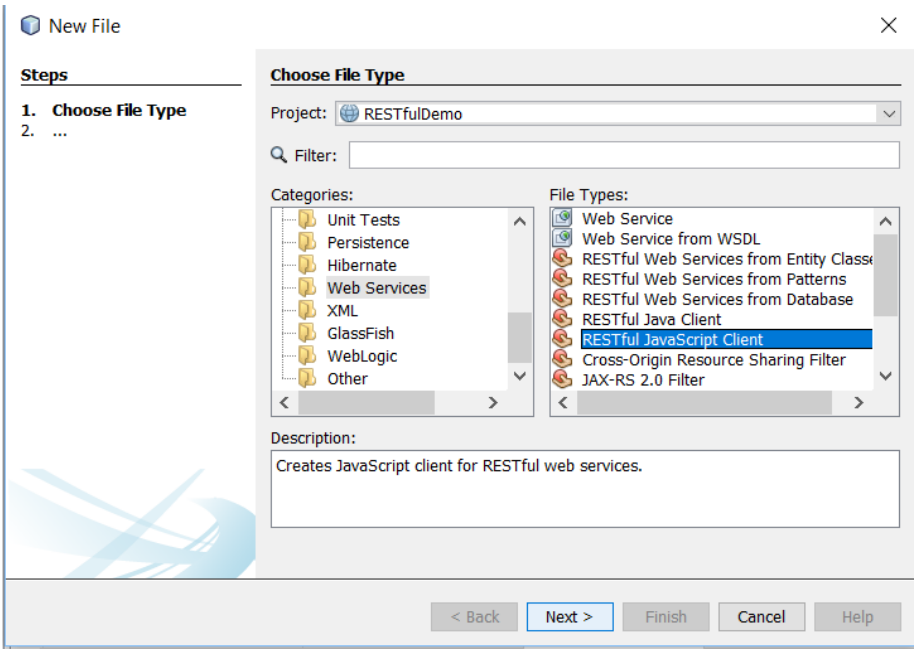

Slika 4.1.2 Dodela naziva projektu iz kategorije HTML5 / JavaScript

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.

# PRIMER 4 - KREIRANJE DATOTEKA JAVASRCIPT RESTFUL KLIJENATA

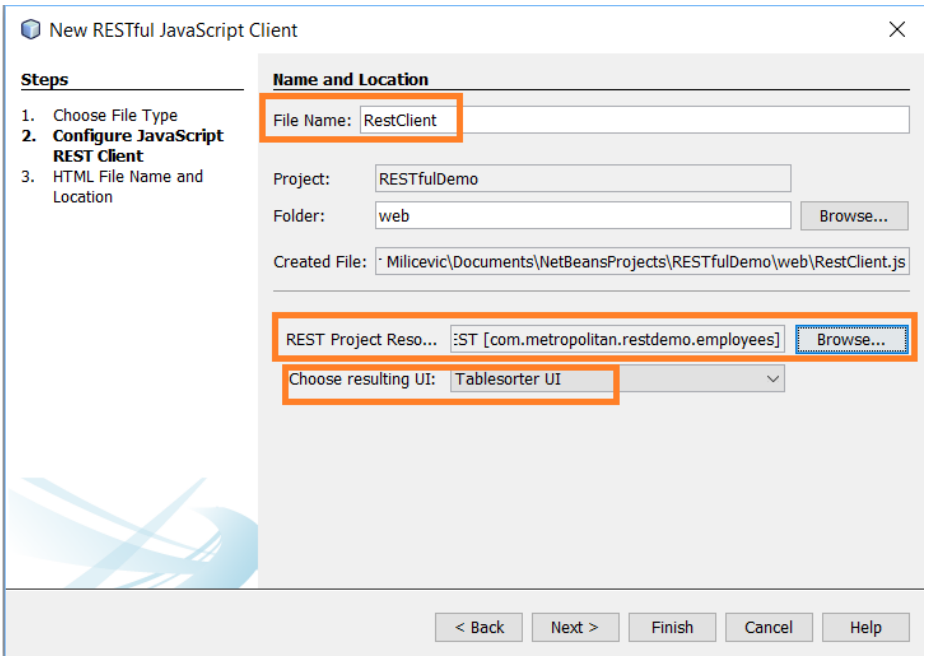
*U nastavku, u kreiranom projektu je neophodno kreirati datoteke JavaSrcipt RESTFul klijenata.*

Pošto je nov projekat uspešno kreiran, moguće je pristupiti kreiranju datoteka *JavaSrcipt 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 4.1.3 Izbor kreiranja datoteke tipa RESTful JavaScript Client

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



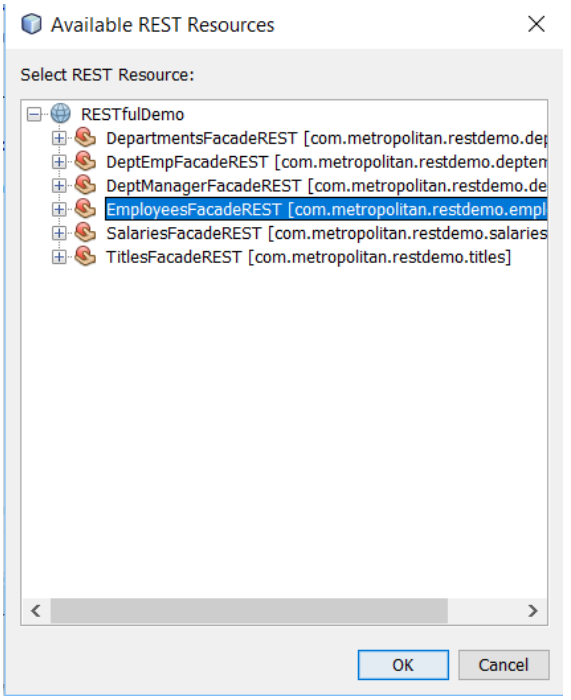

Slika 4.1.4 Izbror naziva, REST resura i predefinisanog korisničkog interfejsa

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

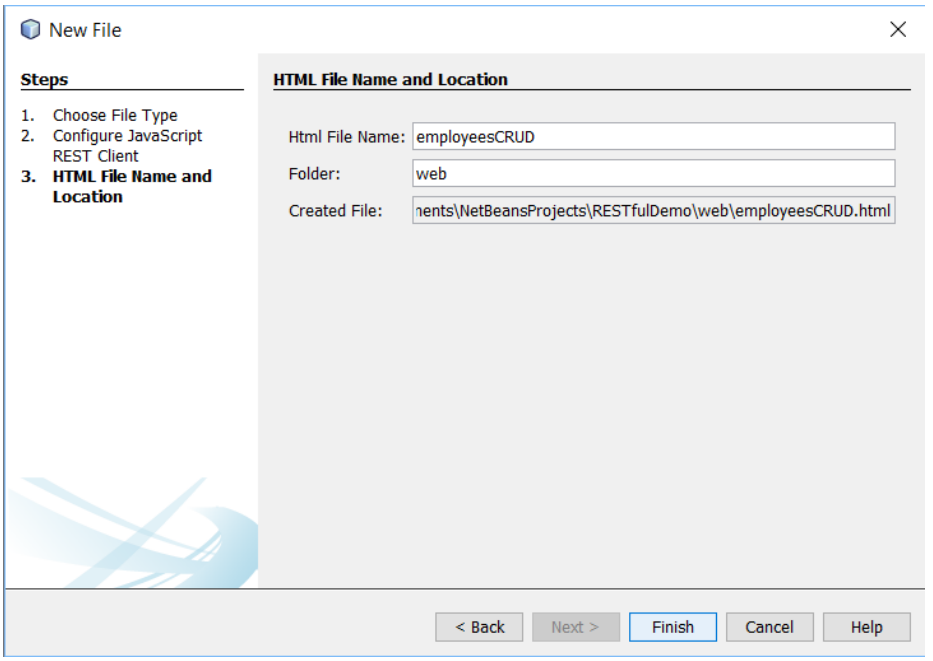
*Neophodno je kreirati i HTML stranu koja učitava odgovarajuću JS datoteku.*

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* otvoriće 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 4.1.5 Izbor REST resursa

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 generisanu *RestClient.js* i koja će omogućiti korisniku da izvodi *CRUD* operacije, koristeći *RESTFul servis EmployeesFacadeREST.java*, nad tabelom *employees*, baze podataka *company*.




Slika 4.1.6 Kreiranje HTML stranice JavaScript klijenta

# 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 imlementation 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.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);
```

```
            }, 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()
&amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp; 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-itemis 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.
            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

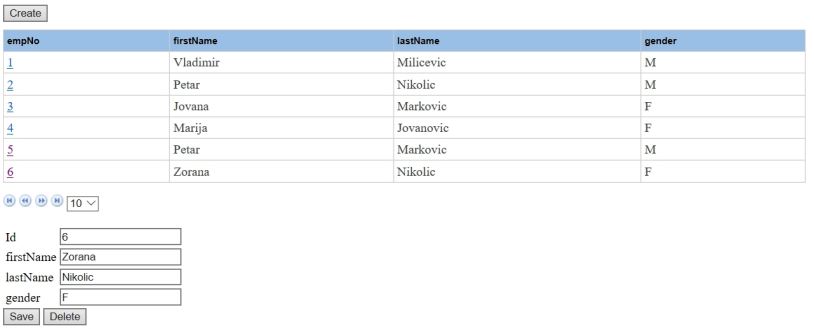
*Nakon kreiranja odgovarajućih datoteka JavaScript klijenta RESTFul servisa, sledi testiranje.*

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

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.

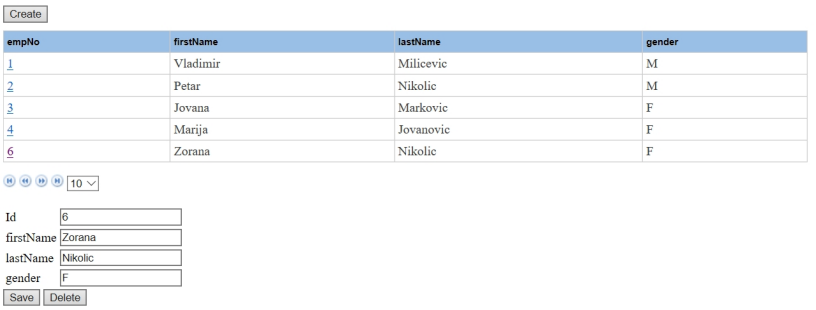

Slika 4.1.7 Stranica EmployeesCRUD sa odgovarajućim UI

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.




Slika 4.1.8 Dodavanje novog zapisa u tabelu

Nov zapis, nakon popunjavanja forme sa slike, klikom na dugme *Save* prosleđuje 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*).




Slika 4.1.9 N

# RESTFUL VEB SERVISI SA JAX - RS - 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.**

# ZADATAK 4

*Kreiranje i testiranje JavaScript klijenta.*

Nastavite vaš primer iz Zadataka 1 i 2, kreiranjem i testiranjem JavaScript klijenta za kreirani RESTFul servis.

## 4.1 Primer 5 - Veb apliakcije sa RESTFul servisima

### ZADATAK 1

*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 postoje unosi izvršiti testiranje;
5. Šema baze podataka aplikacije je data Slikom 1;

Slika 4.2.1 Šema baze podataka

## 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.
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.clien
t.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.clie
nt.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 imlementation 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()
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
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-itemis 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();
});
```

## TABELA "DEPARTMENTS" - HTML STRANICA I TEST

*Za JavaScript klijent RESTFul servisa DepartmentsFacadeREST kreira se HTML stranica.*

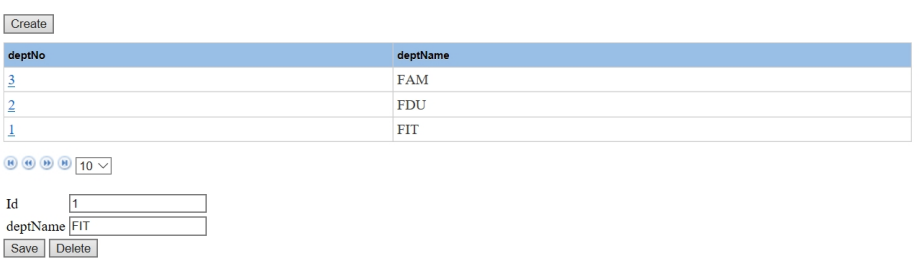
Sledi listing stranice departments.html.

```
<!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'
value='<%= deptName %>'/></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>
```

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



Slika 4.2.2 Test primene RESTFul servisa DepartmentsFacadeREST

## 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
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!empNo.isEmpty()) {
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));
        }
        java.util.List<String> deptNo = map.get("deptNo");
        if (deptNo != null
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!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.clien
t.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.clie
nt.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()
&amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
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-itemis 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
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!deptNo.isEmpty()) {
            key.setDeptNo(deptNo.get(0));
        }
        java.util.List<String> empNo = map.get("empNo");
        if (empNo != null
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!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.clien
t.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.clie
nt.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);
            }, 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()
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
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-itemis 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) {
        /*
         * pathSemgent 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
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!empNo.isEmpty()) {
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));
        }
        java.util.List<String> fromDate = map.get("fromDate");
        if (fromDate != null
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
!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.clien
t.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.clie
nt.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 imlementation 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()
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
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-itemis 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
&amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;
!empNo.isEmpty()) {
            key.setEmpNo(new java.lang.Integer(empNo.get(0)));
        }
        java.util.List<String> title = map.get("title");
        if (title != null
&amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;
!title.isEmpty()) {
            key.setTitle(title.get(0));
        }
        java.util.List<String> fromDate = map.get("fromDate");
        if (fromDate != null
&amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;
!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.clien
t.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.clie
```

```
nt.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" 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 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()
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
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-itemis 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>
```

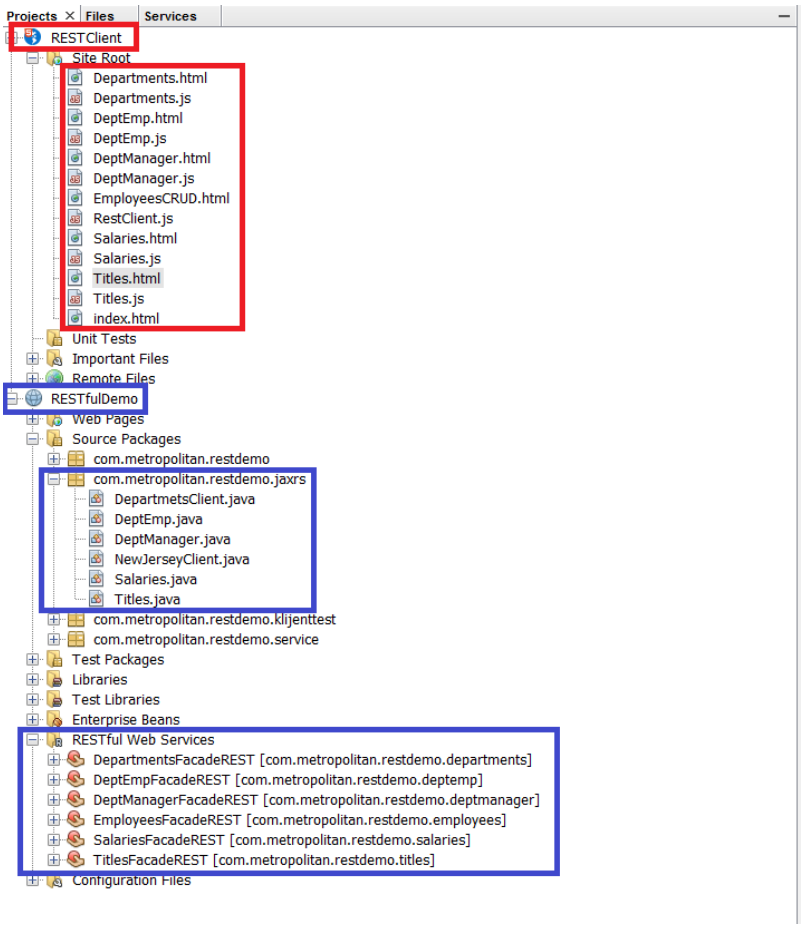
## KOMPLETIRAN PROJEKAT RESTFULDEMO - KONAČNA STRUKTURA

*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.



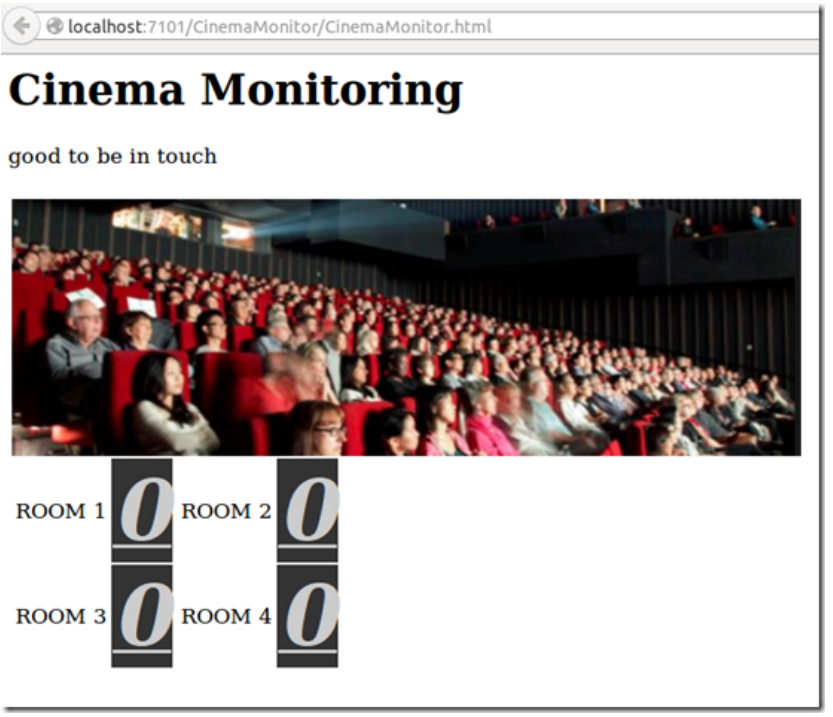

Slika 4.2.3 Konačna struktura projakata RESTFulDemo i RESTClient

# ZADATAK 5

## *Pokušajte sami.*

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 4.2.4 Aplikacija za monitoring posećenosti bioskopskih sala

# Poglavlje 5

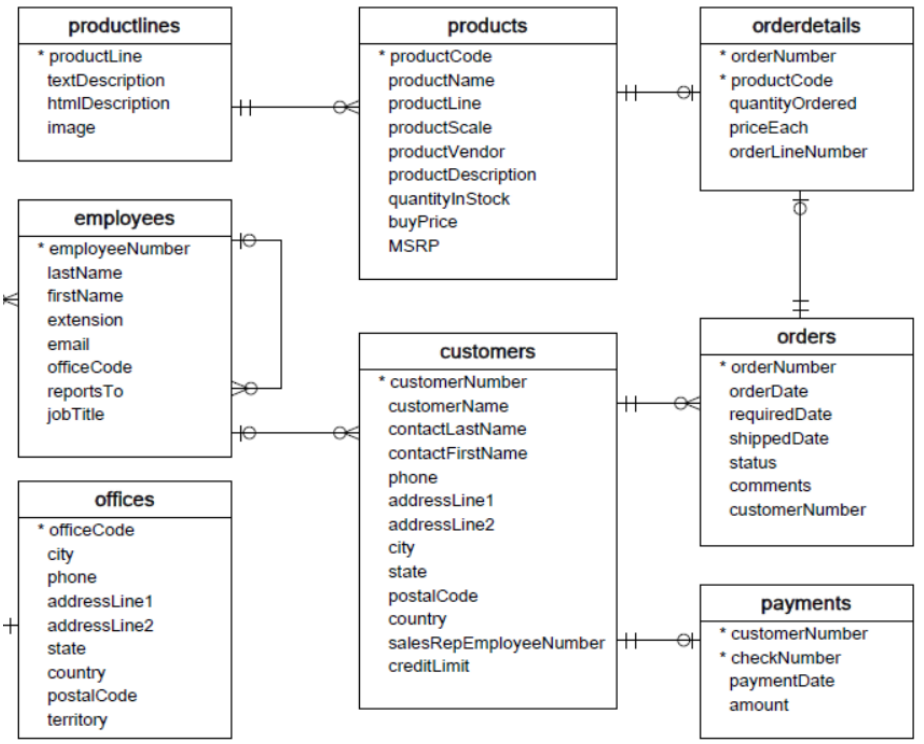
# Domaći zadatak 4

## ZADATAK 1

### *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 5.1 Šema baze podataka za domaći zadatak

# Poglavlje 6

# Zaključak

## ZAKLJUČAK

*Lekcija 4 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 odabranih 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 kodaRESTfulveb servisa;
- Generisanje RESTful JavaScript klijenataRESTful veb servisa;

Savladavanjem ove lekcije student je osposobljen u potpunosti razume i koristi RESTFul 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 Develpomnet 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