5. Laborator: Servicii REST

5.1 DB

Ne vom folosi de baza de date creata in cadrul laboratorului 3. Pentru recapitulare:

Nume : **lab3DB,** user: **lab3,** pass: **lab3**

Continut:

```
-- DROP TABLE DEPARTMENT;
CREATE TABLE DEPARTMENT (
    "ID" INTEGER not null GENERATED ALWAYS AS IDENTITY primary key,
    "NAME" VARCHAR(30)
);

-- CREATE
INSERT INTO DEPARTMENT ("NAME") VALUES('dept a');
INSERT INTO DEPARTMENT ("NAME") VALUES('dept b');
INSERT INTO DEPARTMENT ("NAME") VALUES('dept c');
INSERT INTO DEPARTMENT ("NAME") VALUES('dept c');

-- DROP TABLE EMPLOYEE;
CREATE TABLE EMPLOYEE (
    "ID" INTEGER not null  GENERATED ALWAYS AS IDENTITY primary key,
    "FIRSTNAME" VARCHAR(30),
    "LASTNAME" VARCHAR(30),
    "IDDEPARMENT" INTEGER CONSTRAINT dept_fk REFERENCES DEPARTMENT ("ID") ON
    DELETE SET NULL
);

-- CREATE
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES('user 1',
    'popescu', 3);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES('user 2',
    'popa', 3);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES('user 3',
    'moraru', 4);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES('user 4',
    'muntean', 2);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES('user 5',
    'besoiu', 2);
```

5.2 Configurare detalii Db in Glassfish

Vom defini un bazin de conexiuni (Connection Pool) si o resursa jdbc (JDBC Resource).

Pentru asta accesam interfata de a administrare a Glassfish-ului: http://localhost:4848

Lab 5

Connection Pool:

a) Resource > JDBC > JDBC Connection Pools: New

b) Pool Name: Lab3Pool

Resource type: java.sqlDataSource

Database Driver Vendor: Derby-30

c) User: lab3

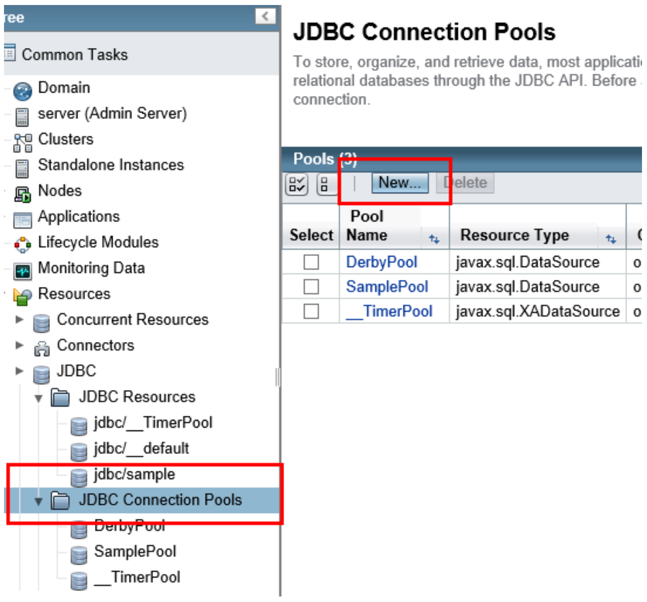
Password: lab3

DatabaseName: lab3DB

ServerName: localhost

PortNumber: 1527

(restul parametrilor trebuie eliminati)

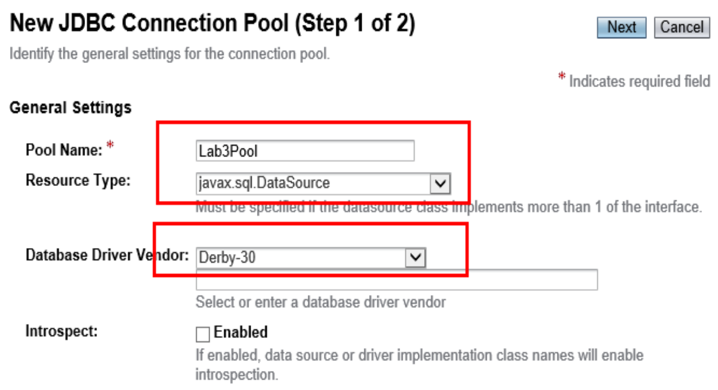


JDBC Connection Pools

To store, organize, and retrieve data, most applicati relational databases through the JDBC API. Before connection.

Pools (3)

Select	Pool Name	Resource Type
☐	DerbyPool	javax.sql.DataSource
☐	SamplePool	javax.sql.DataSource
☐	__TimerPool	javax.sql.XADataSource

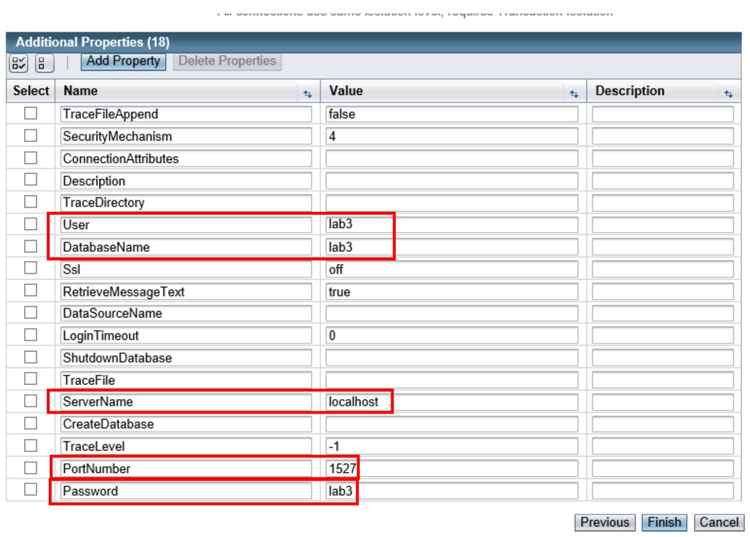

New JDBC Connection Pool (Step 1 of 2)

Identify the general settings for the connection pool.

* Indicates required field

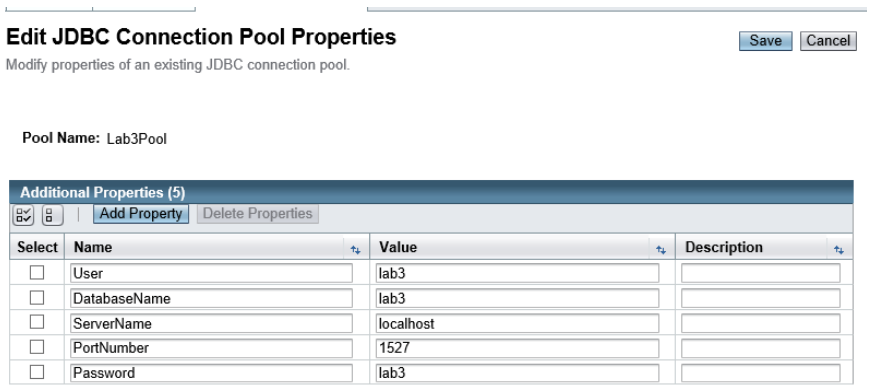
General Settings

Pool Name: *	Lab3Pool
Resource Type:	javax.sql.DataSource
	Must be specified if the datasource class implements more than 1 of the interface.
Database Driver Vendor:	Derby-30
	Select or enter a database driver vendor
Introspect:	☐ Enabled
	If enabled, data source or driver implementation class names will enable introspection.


Additional Properties (18)

Select	Name	Value	Description
☐	TraceFileAppend	false	
☐	SecurityMechanism	4	
☐	ConnectionAttributes		
☐	Description		
☐	TraceDirectory		
☐	User	lab3	
☐	DatabaseName	lab3	
☐	Ssl	off	
☐	RetrieveMessageText	true	
☐	DataSourceName		
☐	LoginTimeout	0	
☐	ShutdownDatabase		
☐	TraceFile		
☐	ServerName	localhost	
☐	CreateDatabase		
☐	TraceLevel	-1	
☐	PortNumber	1527	
☐	Password	lab3	

Dupa eliminare parametrilor nenecesari:


Edit JDBC Connection Pool Properties

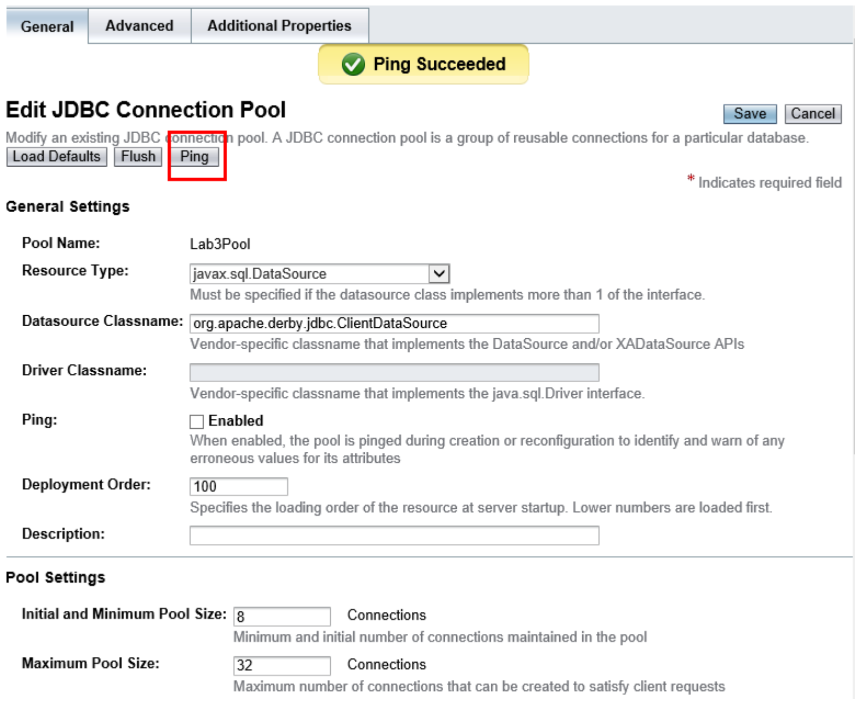
Modify properties of an existing JDBC connection pool.

Pool Name: Lab3Pool

Additional Properties (5)

Select	Name	Value	Description
☐	User	lab3	
☐	DatabaseName	lab3	
☐	ServerName	localhost	
☐	PortNumber	1527	
☐	Password	lab3	

d) Verificare conexiune:



Ping Succeeded

Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

* Indicates required field

General Settings

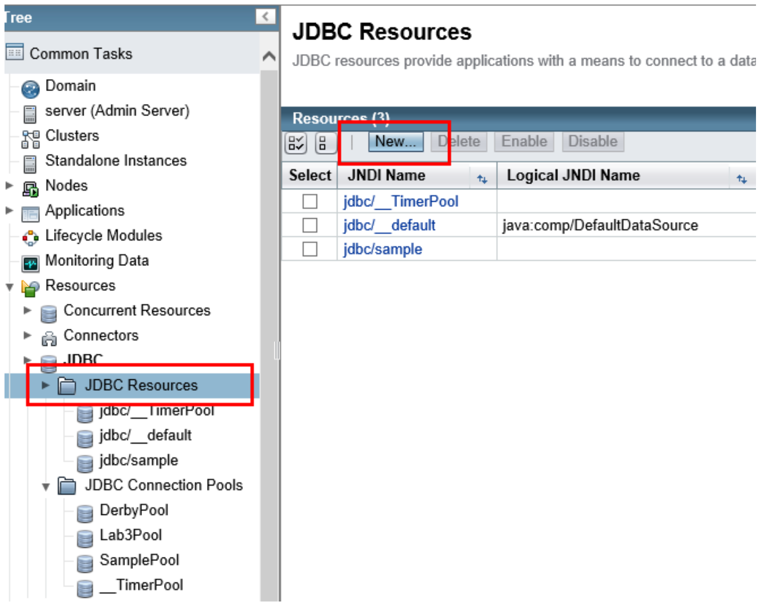
Pool Name:	Lab3Pool
Resource Type:	javax.sql.DataSource Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:	org.apache.derby.jdbc.ClientDataSource Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:	 Vendor-specific classname that implements the java.sql.Driver interface.
Ping:	☐ Enabled When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Deployment Order:	100 Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:	

Pool Settings

Initial and Minimum Pool Size:	8 Connections Minimum and initial number of connections maintained in the pool
Maximum Pool Size:	32 Connections Maximum number of connections that can be created to satisfy client requests

JDBC Resource:

a) Resource > JDBC > JDBC Resources: New
b) JNDI Name: jdbc/lab5res
   Pool Name: lab3pool

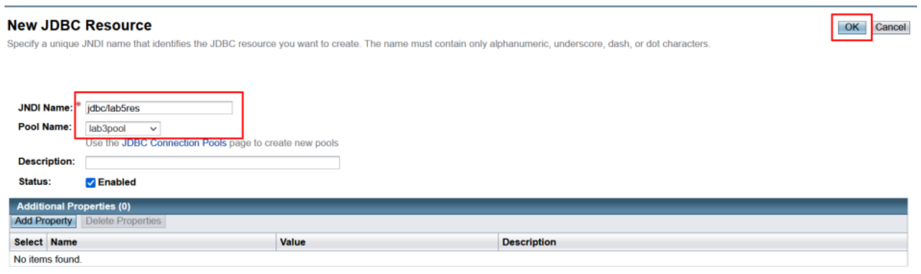


JDBC Resources

JDBC resources provide applications with a means to connect to a data

Resources (3)

Select	JNDI Name	Logical JNDI Name
☐	jdbc/__TimerPool	
☐	jdbc/__default	java:comp/DefaultDataSource
☐	jdbc/sample	



New JDBC Resource

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: *	jdbc/lab5res
Pool Name:	lab3pool Use the JDBC Connection Pools page to create new pools
Description:	
Status:	✓ Enabled

Additional Properties (0)

Select	Name	Value	Description
No items found.			

5.3 Aplicatia

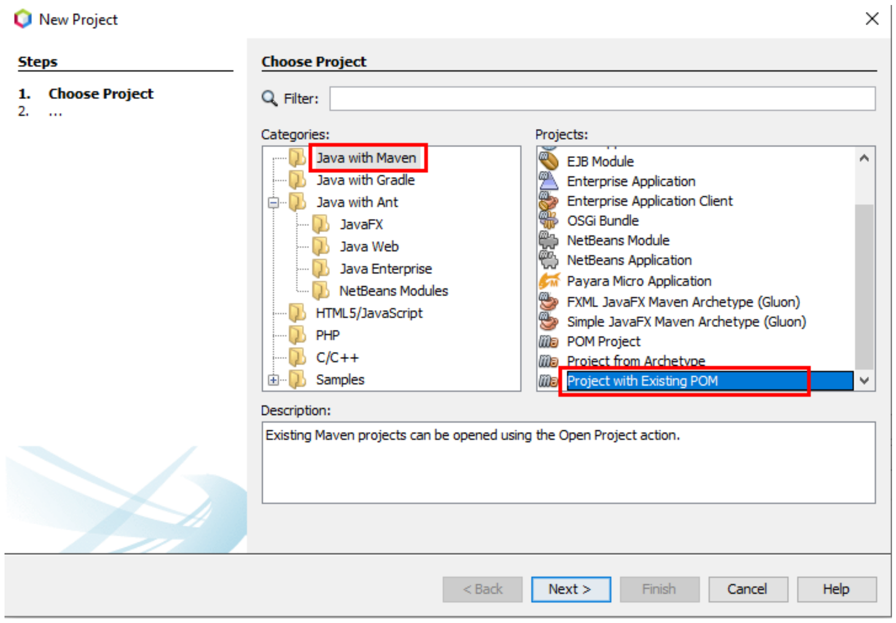
Vom incepe pe un schelet de aplicatie de tip Maven, cu 3 module: db, rest, si ear.

Modulul **db** va contine partea de persistenta: clase Entity, persistence.xml, clasa EJB care folosindu-se de EntityManager intermediaza obtinerea de instante de tip Entity

Modulul **rest** pentru servicii de tip REST.

Modulul **ear** este folosit pentru definirea modului in care cele doua module de mai sus si librariile aferente sunt impachetate.

Dezarhivati arhiva laborator5ee_starter.zip pe disc, si apoi din Netbeans deschideti acest proiect maven, cu optiunea 'Project with existing POM' / 'existing sources'.



Existing Maven projects can be opened using the Open Project action.

5.3.1 Adaugare entitatilor

Deschideti modulul **db**, in surse adaugati entitatile:

```
package ro.ulbs.ism.ip;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```

Lab 5

```
@Table(name = "DEPARTMENT")
public class Department implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;


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

    public Department() {
    }

    public Department(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

```
package ro.ulbs.ism.ip;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "EMPLOYEE")
public class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;


    @Column(name = "firstname")
    private String firstname;


    @Column(name = "lastname")
    private String lastname;

    @JoinColumn(name = "iddepartment", referencedColumnName = "id")
    @ManyToOne
    private Department department;

    public Employee() {
    }

    public Employee(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

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

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public Department getDepartment() {
        return department;
    }

    public void setDepartment(Department dep) {
        this.department = dep;
    }
}
```

Lab 5

Adaugati clasa prin care gestionam instantele entity:

```java
/* Fisierul DepartmentEjb.java */
package ro.ulbs.ism.ip;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class DepartmentEjb {

    @PersistenceContext(unitName = "thepersistenceunit")  //see main/resources/META-
INF/peristence.xml
    protected EntityManager manager;

    public EntityManager getEntityManager() {
        return manager;
    }

    public List<Department> getList() {
        EntityManager manager = getEntityManager();
        List<Department> depts = new ArrayList<>();
        try {
            depts = manager.createQuery("SELECT d FROM Department d").getResultList();
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return depts;
    }

    public List<Department> filter(String filter) {
        EntityManager manager = getEntityManager();
        List<Department> depts = manager
                .createQuery("SELECT d FROM Department d WHERE d.name like :param")
                .setParameter("param", filter)
                .getResultList();
        return depts;

    }

    public Integer createDepartment(Department dept) {
        EntityManager manager = getEntityManager();
        manager.persist(dept);
        manager.flush();
        return dept.getId();
    }
}
```

Anotarea @Stateless determina ca o clasa java sa fie considerata EJB (Enterprise Java Bean), si in consecinta sa beneficieze de tranzactionabilitate

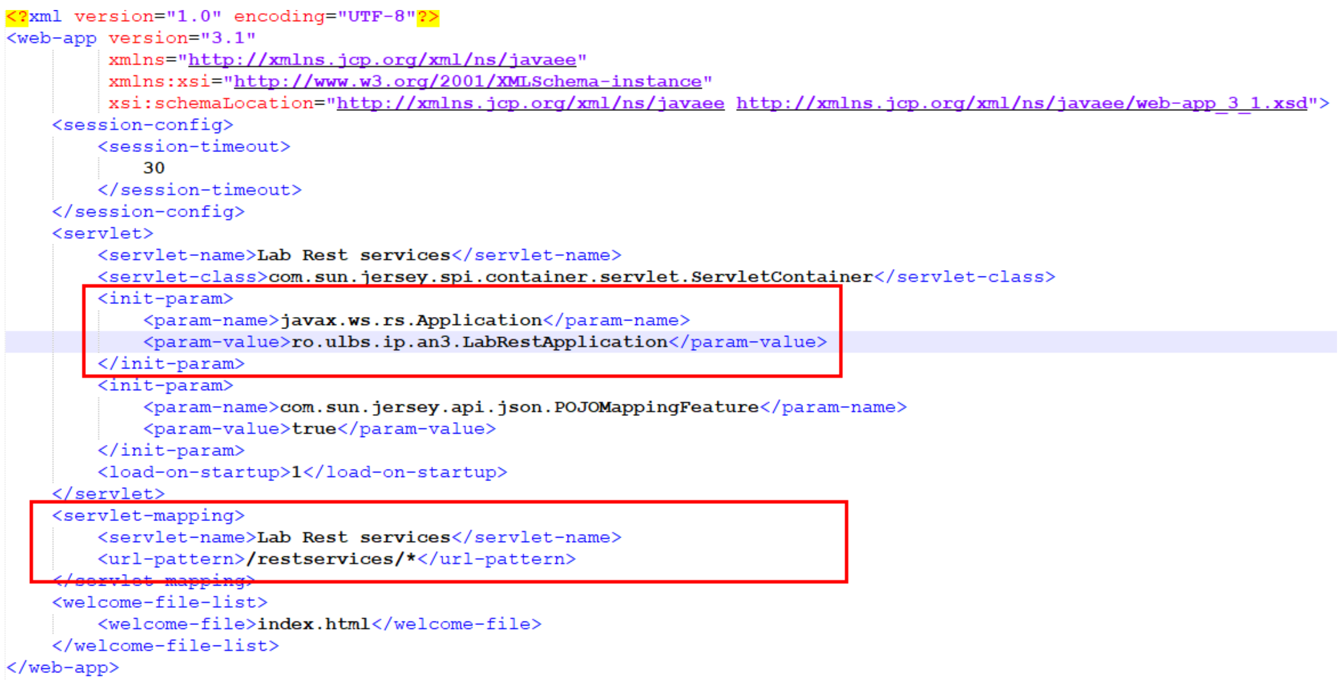
5.3.2 Serviciul REST

Jersey este o implementare in java a api-ului REST. Vom configura aceasta librarie precizand cum sa ne foloseasca codul nostru.

Avem nevoie sa definim o ‚aplicatie’:

```java
/* Fisierul LabRestApplication.java */
package ro.ulbs.ip.an3;

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

@ApplicationPath("/")
public class LabRestApplication extends Application {
    @Override
    public Set<Class<?>> getClasses() {
        final Set<Class<?>> classes = new HashSet<>();
        // register root resource
        classes.add(RestDepartments.class);
        return classes;
    }

}
```

Aceasta clasa este definita in web.xml:



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <servlet>
        <servlet-name>Lab Rest services</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>javax.ws.rs.Application</param-name>
            <param-value>ro.ulbs.ip.an3.LabRestApplication</param-value>
        </init-param>
        <init-param>
            <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Lab Rest services</servlet-name>
        <url-pattern>/restservices/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Tot aici am definit url-ul de baza (/restservices) pentru clasa de mai sus.

Iar aceasta clasa determina incarcarea clasei RestDepartments. Aici putem adauga si alte clase care sa se ocupe de servicii rest pentru alte entitati.

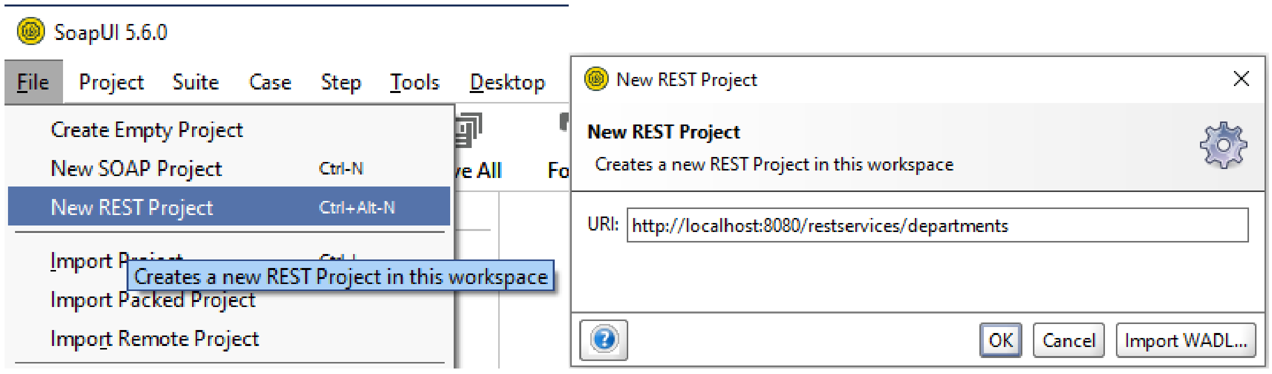
```
/* Fisierul RestDepartments.java */
package ro.ulbs.ip.an3;

import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PUT;
import javax.ws.rs.core.MediaType;

import java.util.List;
import javax.ejb.EJB;
import javax.ws.rs.core.Response;

import javax.ejb.Stateless;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import ro.ulbs.ism.ip.Department;
import ro.ulbs.ism.ip.DepartmentEjb;

/**
 * REST Web Service
 */

/* WHEN ERRORS: https://stackoverflow.com/questions/40540110/javax-naming-
namenotfoundexception-cdiextension
http://localhost:4848/
Glassfish admin console: Configuration > server-config > JVM Settings > tab 'JVM
Options' >
Add a new entry:
    -Dcom.sun.jersey.server.impl.cdi.lookupExtensionInBeanManager=true
Restart your server
*/

@Path("/departments")
@Stateless
public class RestDepartments {

    @Context
    private UriInfo context;

    @EJB
    private DepartmentEjb departmentsEjb;

    /**
     * Creates a new instance of WebServices
     */
    public RestDepartments() {

    }
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response getList() {
        List<Department> depts = departmentsEjb.getList();
        return Response.ok(depts).build();
    }

    @GET
    @Path("/search")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response filter(@DefaultValue("") @QueryParam("filter") String
filterTxt, @Context HttpServletRequest servletRequest) {

        List<Department> depts = departmentsEjb.filter(filterTxt);
        return Response.ok(depts).build();
    }

    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response newDepartment(Department dept) {

        Integer id = departmentsEjb.createDepartment(dept);
        return Response.ok(id).build();
    }
}
```

5.3.3 Deployment si consumare servicii REST

Arhiva ear trebuie deploy-ata in Glassfish. Pentru asta: http://localhost:4848 > applications > deploy

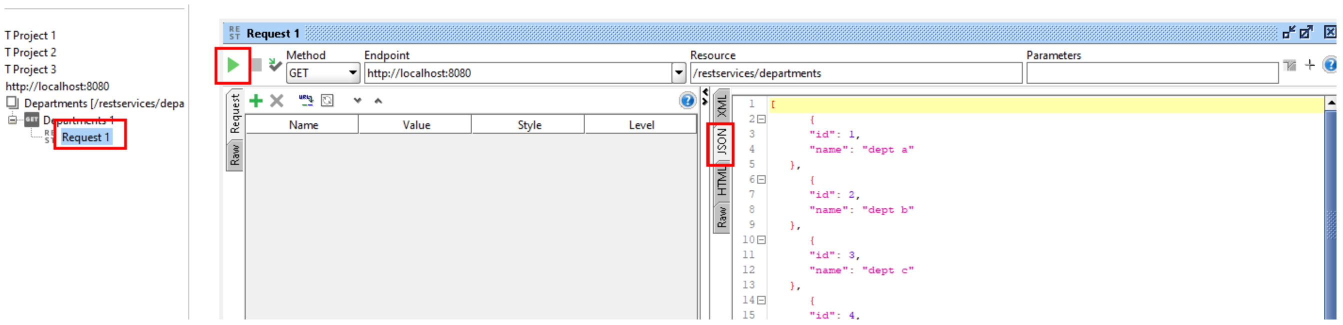
Consumarea se face folosind:
SoapUi (https://www.soapui.org/downloads/soapui/)
PostMan (https://www.postman.com/downloads/ > “Postman Desktop Agent” sau plugin in browser “Postman on the web”)

5.3.3.a)
Pentru SoapUI trebuie creat un nou nou proiect si apoi generate requesturi.



New REST Project
Creates a new REST Project in this workspace
URI: http://localhost:8080/restservices/departments

Request de tip GET - list all:



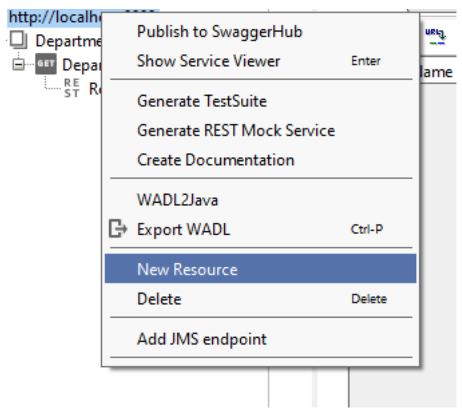
Method: GET
Endpoint: http://localhost:8080
Resource: /restservices/departments

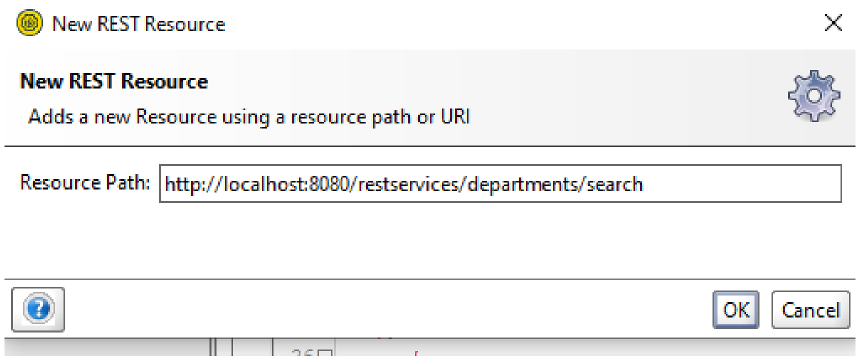
```
[
    {
    "id": 1,
    "name": "dept a"
  },
    {
    "id": 2,
    "name": "dept b"
  },
    {
    "id": 3,
    "name": "dept c"
  },
    {
    "id": 4,
```

5.3.3 b) Request de tip GET – filter :

SoapUI > select http://localhost:8080 > right click > New Resource:

url: http://localhost:8080/restservices/departments/search





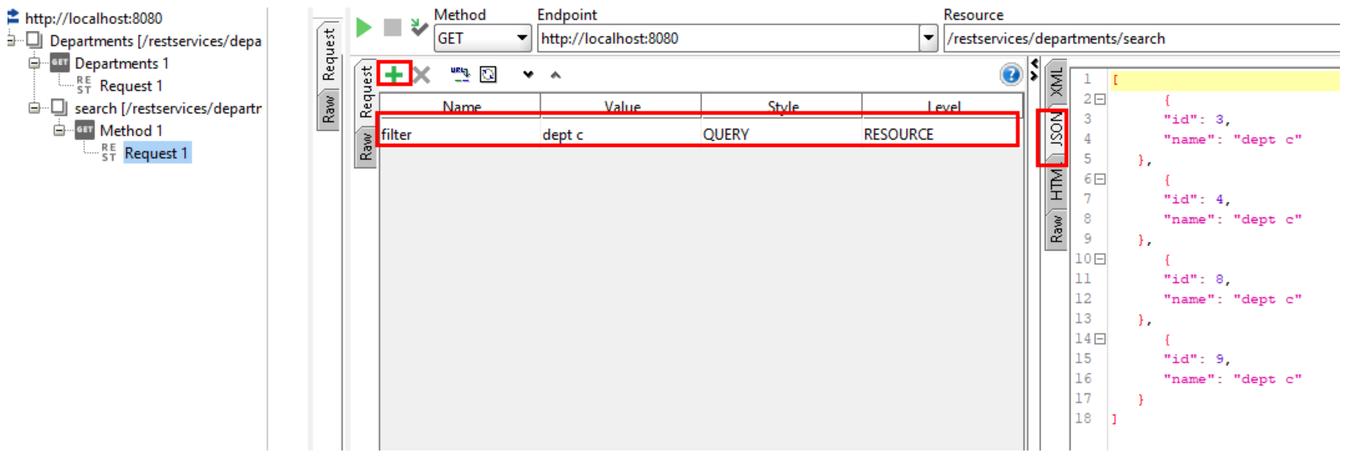
Resource Path: http://localhost:8080/restservices/departments/search

Apoi adaugati un parametru de tip ‘QUERY’,

Name : filter

Value : dept c (sau orice valoare vreti sa cautati)

La executie (triunghiul verde) ar trebui sa primiti inapoi raspunsul json cu elementele filtrate)



http://localhost:8080
Departments [/restservices/depa
Departments 1
Request 1
search [/restservices/departr
Method 1
Request 1

Method: GET | Endpoint: http://localhost:8080 | Resource: /restservices/departments/search

Name	Value	Style	Level
filter	dept c	QUERY	RESOURCE

```
[
    {
        "id": 3,
        "name": "dept c"
    },
    {
        "id": 4,
        "name": "dept c"
    },
    {
        "id": 8,
        "name": "dept c"
    },
    {
        "id": 9,
        "name": "dept c"
    }
]
```

5.3.3.c) Request de tip PUT (creare entitate) :

URL : http://localhost:8080/restservices/departments/

Method : PUT

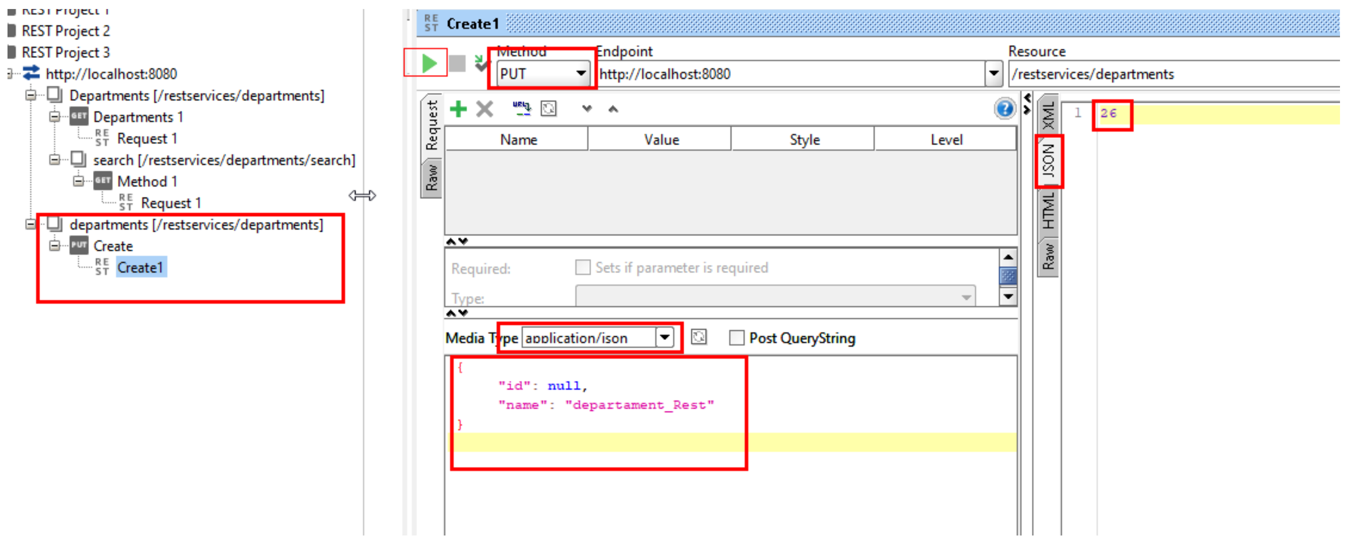
Media Type : Application/JSON

Body : {

"id": null,

"name": "un nou departament"

}



REST Project 1
REST Project 2
REST Project 3
http://localhost:8080
Departments [/restservices/departments]
Departments 1
Request 1
search [/restservices/departments/search]
Method 1
Request 1
departments [/restservices/departments]
Create
Create1

Create1

Method: PUT | Endpoint: http://localhost:8080 | Resource: /restservices/departments

Name	Value	Style	Level

Required: Sets if parameter is required
Type:

Media Type: application/json | Post QueryString

```
{
    "id": null,
    "name": "departament_Rest"
}
```

```
26
```

Salvati proiectul (File > Save as)

Sumar

Aplicatia de fata foloseste urmatoarele : Java EE, Maven, JPA, EJB, REST
Serviciile WEB tip REST sunt expuse la http://localhost:8080/restservices/departments
Browserul web poate accesa url-ul de mai sus si face cerere de tip GET. Pentru alte tipuri de operatii (PUT, POST, DELETE) ne vom folosi de unelte specializate (SoapUI, PostMan, RestAssured).

5.4 Probleme

5.4.1. Extindeti codul de mai sus pentru a oferi servicii de cautare entitate Departament dupa ID.
5.4.2. Adaugati un nou serviciu REST pentru update. Folositi metoda POST.
5.4.3 Adaugati o noua clasa RestEmployee care sa ofere servicii REST pentru entitatile Employee. Trebuie sa oferiti support pentru operatii CRUD (Create – PUT, Read-GET, Update – POST, Delete – DELETE)

Mai multe informatii:

What is REST
https://restfulapi.net/
https://www.tutorialspoint.com/restful/index.htm

What is EJB
https://stackoverflow.com/questions/12872683/what-is-an-ejb-and-what-does-it-do
https://stackify.com/enterprise-java-beans/

When your Dependecy Injection is not working
https://www.adam-bien.com/roller/abien/entry/when_your_di_breaks_bean