# 7. Laborator: Client REST, Facelets – part1

## 7.1 Client Rest

Vom folosi serviciile web oferite de laboratorul 6. Deci trebuie sa aveti lab6-ear.ear instalat in Glassfish.

In continuare ne vom folosi de arhiva starter: laboratoree7_starter.zip.

Pentru consumarea serviciilor REST vom implementa un client REST caruia ii vom preciza URL-ul de unde sa obtina datele.

Ne folosim de aceeasi implementare a serviciilor REST, anume de libraria Jersey folosita si in laboratoarele anterioare.

Codul de baza pentru consumul serviciilor REST este:

```
        Response response = webTarget.
                request(MediaType.APPLICATION_JSON).
                get(Response.class); //fiecare post(entity)

        response.readEntity(…)
```

unde webTarget este o instanta `javax.ws.rs.client.WebTarget` si care efectueaza apelul REST si obtinerea rezultatului. Rezultatul apoi poate fi usor convertit in obiect java.

Pentru implementarea clientului REST, creati pachetul `ro.ulbs.ip.an3.frontend` si apoi clasa `DepartmentRest`:

```
package ro.ulbs.ip.an3.frontend;

import java.util.List;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.GenericType;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

@Stateless
@LocalBean
public class DepartmentRest {

    private static final String BASE_URI = "http://localhost:8080/restservices/departments";

    private final WebTarget webTarget;
    private final Client client;

    public DepartmentRest() {
        //instantierea unui client REST
        client = javax.ws.rs.client.ClientBuilder.newClient();

        //fiecare url va necesita o noua instanta WebTarget
        webTarget = client.target(BASE_URI);
    }

    public List<DepartmentDto> listAll() throws javax.ws.rs.ClientErrorException {
        //apel GET cu precizarea formatului dorit al datelor
        Response resp = webTarget.
                request(MediaType.APPLICATION_JSON).
                get(Response.class);

        //transformarea datelor in obiecte java
        List<DepartmentDto> ret = resp.readEntity(new GenericType<List<DepartmentDto>>() {
        });
        return ret;
    }

    public List<DepartmentDto> filterBy(String name) throws ClientErrorException {
        Response resp = webTarget.path("/search")
                .queryParam("filter", name)
                .request(MediaType.APPLICATION_JSON)
                .get(Response.class);
        return resp.readEntity(new GenericType<List<DepartmentDto>>() {
        });
    }
}
```

Pentru ca ne dorim datele obtinute de la serviciul web sa se materializeze in obiecte java, net trebuie o clasa care sa modeleze aceste obiecte. O vom numi DepartmentDto. Dto = Data Transfer Object.
Cu alte cuvinte, marcam acest tip de obiecte ca fiind elemente de transfer date.
Clasa va avea doua compuri (id, name) si metodele setter/getter aferente:

```
package ro.ulbs.ip.an3.frontend;

import java.io.Serializable;

public class DepartmentDto implements Serializable {
    private static final long serialVersionUID = 1111L;
    private Integer id;
    private String name;

    public DepartmentDto() {
    }

    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;
    }
}
```

Pentru a verifica daca acest client Rest functioneaza vom crea un servlet care apeleaza acest serviciu cand noi vom apel acest servlet in browser.
Pentru aceasta cream un servlet numit TestRestServlet cu continutul de mai jos:

```
package ro.ulbs.ip.an3.frontend;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```java
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "TestRestServlet", urlPatterns = {"/TestRestServlet"})
public class TestRestServlet extends HttpServlet {

    @EJB
    private DepartmentRest restClient;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html><html><body>");
            List<DepartmentDto> departamente = restClient.listAll();
            out.print("Au fost gasite ");
            out.print(departamente.size());
            out.println(" departamente. <br/>");
            out.println("<table border=\"1\" style=\"border-collapse: collapse\">");
            for (DepartmentDto dept: departamente) {
                out.print("<tr><td>");
                out.print(dept.getId());
                out.print("</td><td>");
                out.print(dept.getName());
                out.println("</td></tr>");
            }
            out.println("</table></body></html>");
        }
    }
}
```

Compilati proiectul (click dreapta > build) si apoi deploy-ati in glassfish (http://localhost:4848/ > application > deploy). Pentru deploy selectati fisierul *../target/frontend.war*

Daca deploy-mentul a fost cu succes introducem in browser url-ul:

http://localhost:8080/frontend/TestRestServlet

care ar trebui sa ne afiseze datele obtinute de la serviciul REST oferit de laboratorul 6.

## 7.2 Facelets

Afisarea datelor cu ajutorul servletilor (ca mai sus) este anevoioasa si impracticabila in proiectele mari. Pentru o structurare mai buna a codului avem posibilitatea sa folosim Facelets.

Facelets are doua componente: o clasa java cu rolul Managed Bean si o pagina xhtml ce permite definirea continutului web foarte apropiat de continutul html.

Creati o noua clasa numita DepartmentMBean pe care o consideram managed bean-ul pentru operatiile department. Aceasta clasa va fi injectata in context de application server (Glassfish). Contextul deci face disponibile instante ale claselor ce ne sunt necesare. Aceasta operatie este numita Context Dependecy Injection (CDI).

Codul pentru aceasta clasa este:

Lab 7

```
package ro.ulbs.ip.an3.frontend;

import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

@SessionScoped
@Named("deptMBean")
public class DepartmentMBean implements Serializable {

    private static final long serialVersionUID = 10112;

    private String filterText;

    @EJB
    private DepartmentRest restClient;

    public DepartmentMBean() {
    }

    public List<DepartmentDto> getDepartments() {
        if (filterText != null && filterText.length() > 0) {
            return restClient.filterBy(filterText);
        } else {
            return restClient.listAll();
        }
    }

    public String getFilterText() {
        return filterText;
    }

    public void setFilterText(String filterText) {
        this.filterText = filterText;
    }

    public void filter() {

    }
}
```

Selectati nodul „Web Pages” si creati un nou fisier. (new > other > JavaServer Faces > JSF Page). Pentru denumire folositi „index”. Ca rezultat se va crea fisierul index.xhtml in care adaugati codul:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      >
    <h:head>
        <title>Frontend</title>
    </h:head>
    <h:body>
        <f:view id="masterindexId">
            <h:form id="masterForm">
                <h:commandButton value="Action 1" action="/action1" type="submit"/>
            </h:form>
        </f:view>

        <f:view>
            <h:form id="deptFilterForm">
                Filter:
                <h:inputText value="#{deptMBean.filterText}"/>
                <h:commandLink value="Search" action="#{deptMBean.filter}" type="submit"/>
            </h:form>
            <br/>
            <h:form id="deptForm">
                <h:dataTable value="#{deptMBean.departments}"  var="dept">
                    <h:column>
                        <f:facet name="header">ID</f:facet>
                        <h:outputText value="#{dept.id}"/>
                    </h:column>
                    <h:column>
                        <f:facet name="header">Name</f:facet>
                        <h:outputText value="#{dept.name}"/>
                    </h:column>
                </h:dataTable>
            </h:form>
        </f:view>
    </h:body>
</html>
```

Creati inca o pagina JSF, cu numele action1, in care copiati:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      >
    <h:head>
        <title>Frontend</title>
    </h:head>
    <h:body>
        This is action 1
        <f:view id="goBack">
            <h:form id="goBackForm">
                <h:commandButton value="Go Back" action="/index" type="submit"/>
            </h:form>
        </f:view>

    </h:body>
</html>
```

Compilati proiectul si faceti un nou deployment (http://localhost:4848/ > application > deploy).

Accesati aceste pagini web la: **http://localhost:8080/frontend/**

## Sumar

Aplicatia de fata foloseste urmatoarele : Java EE, Maven, Facelets, client REST citire date (GET)
Serviciile WEB tip REST din laboratorul 6 sunt expuse la http://localhost:8080/restservices/departments
Paginile JSF sunt disponibile la http://localhost:8080/frontend/

Mai multe detalii pentru CDI:
https://www.javacodegeeks.com/cdi-tutorials
https://dzone.com/articles/cdi-di-p1
https://docs.oracle.com/javaee/6/tutorial/doc/giwhl.html

## 7.3 Probleme

7.3.1 Adaugati cod astfel incat sa avem in pagina index.xhtml un textbox si un buton pentru a cauta si afisa un singur departament

7.3.2 Adaugati cod similar pentru a afisa lista pentru datele din tabela Employee

### 7.4 Configurare Glassfish

In cazul in care aveti eroare ce se refera la moxy atunci trebuie sa actualizati libraria moxy in Glassfish.
Bug: https://bugs.eclipse.org/bugs/show_bug.cgi?id=463169.
Gasiti moxy-2.6.1.jar in lista de resurse.
Redenumiti libraria existenta, copiati in …\glassfish\modules\ noul jar si reporniti Glassfish-ul

```
if exist %GF_DIR%\glassfish\modules\org.eclipse.persistence.moxy.jar (
            ren %GF_DIR%\glassfish\modules\org.eclipse.persistence.moxy.jar
%GF_DIR%\glassfish\modules\org.eclipse.persistence.moxy.jar_disabled
)

copy .\org.eclipse.persistence.moxy-2.6.1.jar %GF_DIR%\glassfish\modules\
```