



Academia Tehnică Militară "Ferdinand I"
Facultatea de Sisteme Informaticе și Securitate Cibernetică

PARADIGME DE PROGRAMARE (ÎN JAVA)

CURS 9 - SERVICII WEB REST

COL(R) TRAIAN NICULA

TEMATICĂ DE CURS

- Introducere în paradigme de programare (1)
- Programare orientată pe obiecte (OOP) (3)
- Programare funcțională și asincronă (3)
- Programare reactivă (1)
- **Servicii web REST** (1/2)
- Quarkus Framework (4)

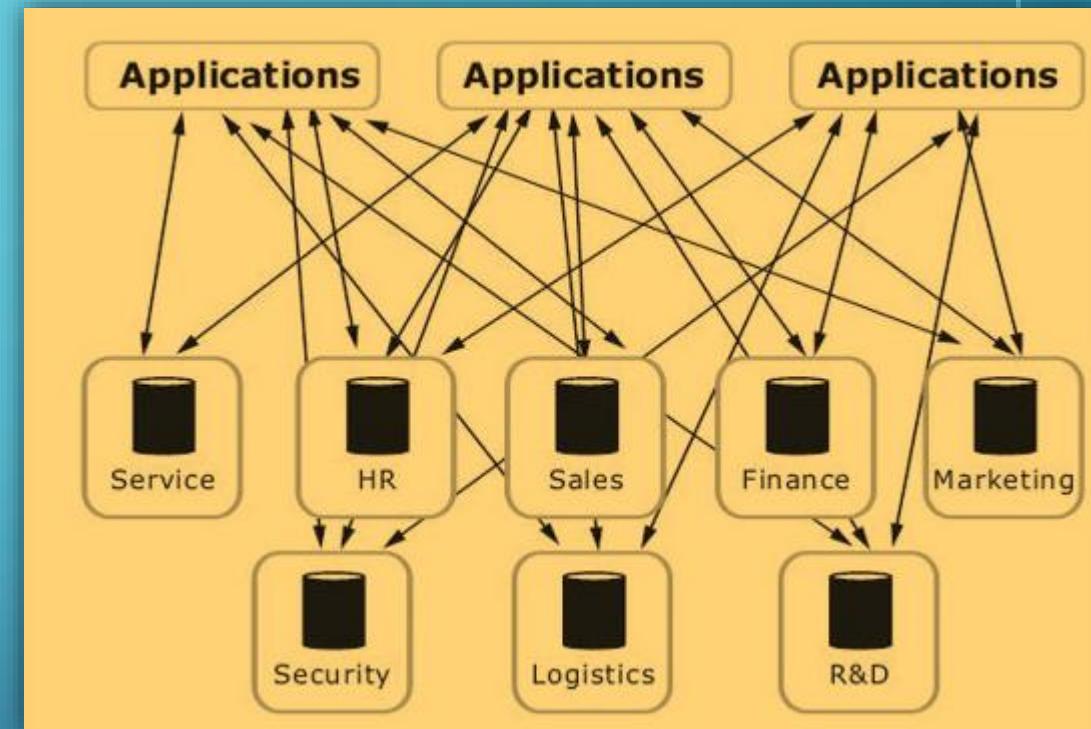
CUPRINS CURS

- Stil arhitectural REST
- Introducere în HTTP
- Procesare JSON în Java
- Prezentare JAX-RS
- JAX-RS – Adnotări
- JAX-RS – Response
- Implementare serviciu REST
- Bibliografie



STIL ARHITECTURAL REST

- Stilul arhitectural ***Representational State Transfer (REST)*** a fost introdus pentru prima dată de Roy Fielding în lucrarea lui de doctorat
- **REST nu este o arhitectură ci mai curând un set de constrângeri care pot fi folosite pentru a crea aplicații distribuite**
- **Aplicațiile enterprise au nevoie de acces simplificat și posibilitate de modificare a datelor localizate pe diferite sisteme**



STIL ARHITECTURAL REST

Fielding a identificat următoarele constrângeri care **definesc un sistem REST**:

- **Stateless** – serviciul nu trebuie să păstreze sesiunea utilizator. **Cererile HTTP trebuie să fie independente** una de alta
- **Cacheable** – se impune implementarea unui **sistem de caching la diferite niveluri** pentru a evita cererile repetitive pentru aceeași resursă
- **Uniform interface** – **interacțiunile** dintre client și server **se desfășoară printr-o interfață generică** folosind un set generic de metode. **Fiecare resursă expusă trebuie să aibă o adresă unică** accesibilă printr-o interfață generică
- **Layered system** – Serverul poate avea mai multe straturi care îmbunătățesc performanța prin **caching, load balancing**. **Stratul cel mai de sus impune și politicile de securitate**
- **Client-server** – Clientul și serverul sunt slab cuplați, dar există o **interfață comună** care facilitează comunicarea între client și server



INTRODUCERE ÎN HTTP

- Protocolul **Hypertext Transfer Protocol (HTTP)** stă la baza **comunicării pe web**
- **HTTP este un protocol de nivel aplicație care definește modul de formatare, transmitere și procesare a mesajelor hipertext (hyperlinks) peste Internet**

Versiuni HTTP:

- **HTTP/0.9** apărută în 1991
- **HTTP/1.0** apărută în 1996
- **HTTP/1.1** apărută în 1999, introduce metode GET, HEAD și POST. Este **versiunea cea mai răspândită** și astăzi
- **HTTP/2** apărută în 2015, suportată deja de majoritatea browser-elor. Se ocupă mai mult cu modul de transport al datelor între client și server



INTRODUCERE ÎN HTTP

MODELUL CERERE-RĂSPUNS

Cuprins

- HTTP/1.1 are un model cerere-răspuns de comunicare cu serverul

Cerere	Răspuns
<p>GET /restful/ HTTP/1.1</p> <p>Host: localhost:8080</p> <p>User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:104.0) Gecko/20100101 Firefox/104.0</p> <p>Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8</p> <p>Accept-Language: en-US,en;q=0.5</p> <p>Accept-Encoding: gzip, deflate, br</p> <p>Connection: keep-alive</p>	<p>HTTP/1.1 200 OK</p> <p>Content-Type: text/html;charset=ISO-8859-1</p> <p>Content-Length: 247</p> <p>Date: Tue, 20 Sep 2022 08:00:30 GMT</p> <p>Keep-Alive: timeout=20</p> <p>Connection: keep-alive</p>
<p>Exemplu de conținut HTML (body) în răspuns.</p>	<pre><html> <body> <h2>Jersey RESTful Web Application!</h2> <p>Jersey resource <p>Visit Project Jersey websitefor more information on Jersey! </body> </html></pre>



INTRODUCERE ÎN HTTP

MODELUL CERERE-RĂSPUNS

- Utilizatorul introduce în browser adresa <http://localhost:8080/restful/> și trimite cererea
- Browser-ul stabilește conexiunea cu serverul (TCP) și trimite o cerere alcătuită din resursa *URI (Uniform Resource Identifier)*, versiunea de protocol, modificatori de cerere (*headers*), informații client și eventual conținut (*body*)
- La primirea cererii serverul o procesează și returnează un răspuns
- Prima linie a răspunsului este linia de stare. Codul de stare poate semnala *succesul cererii, eroare la server sau la client, redirecționarea cererii sau coduri de informare*
- Urmează header-e de răspuns, care pot contine informații despre resursa solicitată, informații pentru client precum **tipul de conținut, lungimea acestuia, rata de actualizare** etc.
- Ultima parte reprezintă **corpu răspunsului** care poate fi HTML, imagine, video, date binare, XML, JSON etc.



INTRODUCERE ÎN HTTP

METODE PENTRU CERERILE HTTP

- **Uniform Resource Identifier (URI)** este un **text care identifică orice resursă sau nume de pe Internet**
- **URI este Uniform Resource Locator (URL)** dacă conține mijloacele de accesare a resursei, **protocolul și numele serverului**, în exemplul anterior <http://localhost:8080/restful/>
- Prezentăm **metodele suportate de HTTP și acțiunile specifice executate de acestea pe resursa țintă**
- **Serviciile REST** folosesc unele dintre aceste metode precum: **GET, POST, PUT și DELETE**

INTRODUCERE ÎN HTTP

METODE PENTRU CERERILE HTTP

Metodă	Descriere
GET	Obține resurse de pe server pe baza unui URI dat
HEAD	La fel ca GET dar fără corp de răspuns
POST	Trimite date la server. Serverul stochează datele (entitatea) ca o resursă subordonată la URI
PUT	Modifică o resursă pe baza unui URI. Dacă resursa nu există o creează
DELETE	Șterge resursa indicată de URI
TRACE	Trimite înapoi conținutul cererii primite. Se folosește pentru depanare (debug)
OPTIONS	Returnează metodele HTTP pe care serverul le suportă
CONNECT	Stabilește o conexiune la server
PATCH	Aplică modificări parțiale unei resurse identificate prin URI

INTRODUCERE ÎN HTTP

REPREZENTAREA TIPULUI DE CONȚINUT

- Atât **cererea** cât și **răspunsul** conțin **header-e care descriu tipul de conținut** prezent în **corful mesajului**
- **Content-Type** – descrie **tipul de conținut din corful mesajului** atât pentru cere cât și pentru răspuns
- **Accept** – **clientul anunță serverul ce tip așteaptă** în **corful mesajului de răspuns**
- **Conținutul permis** este standardizat prin **tipurile media de Internet (MIME type)**
- **Formatul** tipului de conținut este **tip primar/subtip**

Content-Type: `text/html`

INTRODUCERE ÎN HTTP

REPREZENTAREA TIPULUI DE CONȚINUT

Tip conținut	Descriere
text	Conținutul este text și nu necesită software special pentru citire. Content-Type: <i>text/html</i> anunță că mesajul este în format html și trebuie afișat corespunzător
multipart	Conține mai multe părți de date independente. Content-Type: <i>multipart/form-data</i> este folosit pentru forme care conțin fișiere
application	Date de aplicație sau binare. Content-Type: <i>application/json; charset=utf-8</i> desemnează tipul JavaScript Object Notation (JSON) cu codificare UTF-8
image	Date de tip imagine. Content-Type: <i>image/png</i> indică o imagine în format png în corpul răspunsului
video	Date de tip video
audio	Date de tip audio
...	Mai multe pe site-ul Internet Assigned Numbers Authority (IANA)

INTRODUCERE ÎN HTTP

CODURI DE STARE

Code de stare	Descriere
1xx Informational	
100 Continue	Serverul a primit header-ele și cere clientului să trimită corpul
101 Switching Protocols	Serverul este gata pentru comutare de protocol
102 Processing	Cod informațional trimis de server în cazul procesării de lungă durată
2xx Success	Coduri de succes
200 OK	Cererea este succes și conținutul răspunsului este returnat
201 Created	Cererea este succes și noua resursă este creată
204 No Content	Cererea este succes dar nu se returnează nimic. Apare la ștergerea unei resurse
3xx Redirection:	Clientul trebuie să efectueze alte acțiuni
304 Not Modified	Resursa nu a fost modificată
4xx Client Error	
400 Bad Request	Serverul a eșuat să proceseze cererea din cauza sintaxei greșite
401 Unauthorized	Resursa necesită autentificare



INTRODUCERE ÎN HTTP

CODURI DE STARE

Code de stare	Descriere
403 Forbidden	Serverul refuză să răspundă chiar dacă cererea este valabilă. Motivul este listat în corpul răspunsului
404 Not Found	Resursa cerută nu există în locația specificată în cerere.
405 Method Not Allowed	Metoda specificată în cerere nu este pe resursa specificată prin URI
408 Request Timeout	Clientul nu a răspuns în timpul stabilit de server
5xx Server Error	Cod ce indică eșec la server în procesarea cereri
500 Internal Server Error	Mesaj generic de eroare care indică o eroare neașteptată la server
...	IANA

ELEMENTELE ARHITECTURALE REST

RESURSELE

- O resursă REST este orice este adresabil peste web
- Reprezentarea resurselor este ceea ce este trimis între clienți și server
- O reprezentare este o stare la un moment dat a datelor stocate pe server la momentul cererii
- O reprezentare poate lua diferite forme precum XML, JSON, text în funcție de cererile clientilor. Toți clientii vor accesa același URI cu valorile corespunzătoare ale header-ului Accept
- În forma cea mai simplă serviciile web REST sunt aplicații de rețea care manipulează starea resurselor (creare, citire, actualizare și ștergere)

Acțiune	HTTP
Creare	POST sau PUT
Citire	GET
Actualizare	PUT
Ștergere	DELETE

PROCESARE JSON ÎN JAVA

- Cele mai populare **formate utilizate de serviciile web REST** sunt **JSON** și **XML**
- **JSON (JavaScript Object Notation)** este un format de schimb de date de tip text în care **obiectele sunt reprezentate ca perechi atribut valoare**. Provine din JavaScript
- **JSON** este reprezentat de **2 tipuri de structuri**:
- **Colecție neordonată de perechi nume-valoare**. Numele și valoarea sunt separate prin :, numele este de tip String iar valorile un tip JSON (număr, string, boolean, vector, obiect sau null). **Perechile sunt separate prin virgulă**, iar **obiectul este înconjurat de acolade**.

```
{ "departmentId":10, "departmentName":"IT", "manager": "John Chen" }
```



PROCESARE JSON ÎN JAVA

- O **colecție ordonată de valori** (vector) – **Vectorul este închis în paranteze pătrate [].**
Valoare sunt separate cu virgulă.

```
{"departmentName": "IT",  
"employees": [  
    {"firstName": "John", "lastName": "Chen"},  
    {"firstName": "Ameya", "lastName": "Job"},  
    {"firstName": "Pat", "lastName": "Fay"}  
],  
"location": ["New York", "New Delhi"]  
}
```

- **JSON poate fi stocat în fișiere text cu extensia .json**
- **Implementările JAX-RS, respectiv serviciile web REST** în Java, fac automat **serializarea și deserializarea** de la JSON la obiecte Java



PROCESARE JSON ÎN JAVA

JACKSON

- **Jackson** este o **bibliotecă Java cu suport pentru procesarea JSON**. Are module și pentru alte formate precum JSON binar, XML, YAML, CSV
- Suportă **3 modele de procesare JSON**:
 - **Tree model API** – pentru **reprezentare de tip arbore** a documentului JSON
 - **Data binding API** – pentru **convertirea obiectului JSON în și de la obiecte Java**
 - **Streaming API** – pentru **citirea și scrierea unui document JSON**
- **Tree model API**

Clasă	Descriere
com.fasterxml.jackson.databind.ObjectMapper	Face conversia între obiectele Java și reprezentarea lor în format JSON
com.fasterxml.jackson.databind.JsonNode	Clasă de bază pentru nodurile JSON, care alcătuiesc modelul de tip arbore Jackson



PROCESARE JSON ÎN JAVA JACKSON

Vezi proiectul course9/jackson

```
...
public static void findAndUpdateNode(String file) throws IOException{
    InputStream input = ReaderExample.class.getResourceAsStream(file);
    // create ObjectMapper instance
    ObjectMapper objMapper = new ObjectMapper();
    // read JSON like DOM Parser
    JsonNode rootNode = objMapper.readTree(input);
    if (rootNode.isArray()){
        for (JsonNode node : rootNode){
            System.out.println(node);
            JsonNode id = node.path("employeeId");
            logger.log(Level.INFO, "id-{0}", id.asInt());
            JsonNode email = node.path("email");
            logger.log(Level.INFO, "email = {0}", email.textValue());
            if (email.textValue() == null){
                ((ObjectNode)node).put("email", "unknown");
            }
        }
        objMapper.writeValue(new File("emp-array-modified.json"), rootNode);
    }
}
```

Exemplul prezintă modul de citire al unui fișier JSON folosind **Tree API**. Metoda **readTree()** creează o reprezentare de tip arbore pentru fișier având ca rădăcină nodul **rootNode**. Se parcurge iterativ vectorul de noduri și se modifică valorile nule pentru email. Se salvează rezultatul într-un alt fișier JSON (metoda **writeValue()**).

PROCESARE JSON ÎN JAVA JACKSON

Vezi proiectul course9/jackson

```
package atm.paradigms;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;

public class ReaderExample {
    private static final Logger logger =
        Logger.getLogger(ReaderExample.class.getName());
    public static void main(String[] args)
        throws IOException {
        logger.setLevel(Level.INFO);
        findAndUpdateNode("emp-array.json");
    }
    ...
}
```



PROCESARE JSON ÎN JAVA

JACKSON - JSONNODE

Metodă	Descriere
readTree()	Citește JSON într-un obiect JsonNode. Se aplică pe o instanță ObjectMapper
	<pre>JsonNode jsonNode = objectMapper.readTree(json);</pre>
get()	Accesează valoarea unui câmp al unui obiect JsonNode și returnează tot JsonNode
	<pre>JsonNode field1 = jsonNode.get("field1");</pre>
path()	La fel ca get() dar returnează eroare dacă nu există câmpul respectiv
	<pre>JsonNode field1 = jsonNode.path("field1");</pre>
asText(), asDouble(), asInt(), aslong()	Metode pentru a converti JsonNode la alte tipuri de date
	<pre>String f2Str = jsonNode.get("f2").asText(); double f2Dbl = jsonNode.get("f2").asDouble(); int f2Int = jsonNode.get("f2").asInt(); long f2Lng = jsonNode.get("f2").asLong();</pre>
textValue()	Metodă pentru a accesa valoarea String a unui JsonNode
	<pre>String email = node.path("email").textValue();</pre>



File Edit Selection View Go Run Terminal Help ReaderExample.java - jackson - Visual Studio Code

JACKSON

- .vscode
- src
 - main\java\atm\paradigms
 - App.java
 - emp-array.json
 - ReaderExample.java
 - test
 - target
 - .editorconfig
 - emp-array-modified.json
 - pom.xml

Fișierul original este **emp-array.json** (de tip resursă). Fișierul rezultat în urma execuției este **emp-array-modified.json**. Se salvează în calea proiectului.

```

src > main > java > atm > paradigms > ReaderExample.java > ReaderExample > findAndUpdateNode(String)
5   import java.util.logging.Level;
6   import java.util.logging.Logger;
7
8   import com.fasterxml.jackson.databind.JsonNode;
9   import com.fasterxml.jackson.databind.ObjectMapper;
10  import com.fasterxml.jackson.databind.node.ObjectNode;
11
12  public class ReaderExample {
13      private static final Logger logger =
14          Logger.getLogger(ReaderExample.class.getName());
15      Run | Debug
16      public static void main(String[] args) throws IOException {
17          logger.setLevel(Level.INFO);
18          findAndUpdateNode(file: "emp-array.json");
19
20
21      public static void findAndUpdateNode(String file) throws IOException{
22          InputStream input = ReaderExample.class.getResourceAsStream(file);
23          // create ObjectMapper instance
24          ObjectMapper objMapper = new ObjectMapper();
25          //read JSON like DOM Parser
26          JsonNode rootNode = objMapper.readTree(input);
27          if (rootNode.isArray()){
28              for (JsonNode node : rootNode){
                  System.out.println(node);
              }
          }
      }
  
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

Run: ReaderExample

```

{"employeeId":102,"firstName":"Pat","lastName":"Fay","email":"pat.fey@xxx.com","hireDate":"2001-03-06"
"}
Sep 21, 2022 11:23:43 AM atm.paradigms.ReaderExample findAndUpdateNode
INFO: id-102
Sep 21, 2022 11:23:43 AM atm.paradigms.ReaderExample findAndUpdateNode
INFO: email = pat.fey@xxx.com
PS D:\dev\ATM\Paradigms\code\course9\jackson>
  
```

x 0 ▲ 3 ↗ ✓

Ln 26, Col 28 Spaces: 4 UTF-8 CRLF {} Java Prettier ↗ ↘

22 din 63



PROCESARE JSON ÎN JAVA JACKSON

- Uneori este necesară **conversia dinamică a conținutului JSON la obiecte** specifice Java precum Map, liste, String, numere., Boolean etc.

```
public static void main(String[] args) throws IOException {
    logger.setLevel(Level.INFO);
    String jsonString = "{\"employeeId\":100, \"firstName\":\"John\", \"lastName\":\"Chen\"}";
    System.out.println(getSimpleBinding(jsonString));
}

public static Map<String, String> getSimpleBinding(String json)
        throws JsonMappingException, JsonProcessingException {
    ObjectMapper om = new ObjectMapper();
    Map<String, String> result = om.readValue(json,
        new TypeReference<Map<String, String>>(){});
    return result;
}
```

Rezultat:

{employeeId=100, firstName=John, lastName=Chen}

Metoda **readValue()** primește ca argumente un String conținând obiectul JSON și o instanță a clasei generice abstracte **TypeReference** parametrizată cu tipul așteptat la deserializare.



PROCESARE JSON ÎN JAVA JACKSON

Exemplul demonstrează cum se poate crea o colecție conținând obiecte Employee dintr-un vector JSON de angajați

```
public static void main(String[] args)
    throws StreamReadException, DatabindException, IOException {
    logger.setLevel(Level.INFO);
    List<Employee> staff = getEmployeeList("emp-array.json");
    System.out.println(staff);
}

public static List<Employee> getEmployeeList(String file)
    throws StreamReadException, DatabindException, IOException{
    InputStream input = ReaderExample.class.getResourceAsStream(file);
    // create ObjectMapper instance
    ObjectMapper objMapper = new ObjectMapper();
    CollectionType collection = objMapper.getTypeFactory()
        .constructCollectionType(List.class, Employee.class);
    List<Employee> result = objMapper.readValue(input, collection);
    return result;
}
```

Metoda **readValue()** primește ca argumente un **InputStream** conținând obiectul JSON și o instanță a clasei **CollectionType** construită dintr-o listă de instanțe ale clasei Employee.



PROCESARE JSON ÎN JAVA

JACKSON

- Salvarea datelor într-un fișier JSON extern

```
public static void main(String[] args)
    throws StreamReadException, DatabindException, IOException {
    logger.setLevel(Level.INFO);
    String fileName = "emp-array-modified.json";
    writeEmployeeList(fileName);
    logger.log(Level.INFO, "{0} written successfully.",fileName);
}
public static void writeEmployeeList(String file)
    throws StreamReadException, DatabindException, IOException{
    List<Employee> employees = getEmployeeList();
    ObjectMapper om = new ObjectMapper();
    try(OutputStream os = new FileOutputStream(file)){
        om.writeValue(os, employees);
    }
}
private static List<Employee> getEmployeeList()
    throws StreamReadException, DatabindException, IOException{
    return BindingExample.getEmployeeList("emp-array.json");
}
```

Metoda **writeValue()** primește ca argumente un **OutputStream** pentru fișierul JSON și o listă de instanțe ale clasei Employee, care este serializată la un obiect JSON. Folosim try-with-resources.



PROCESARE JSON ÎN JAVA JACKSON

- Exemplul folosește metoda `readValue(InputStream src, Class<T> valueType)` a clasei **ObjectMapper** pentru a converti reprezentarea JSON într-o instanță a clasei **Employee**

```
public static void main(String[] args)
    throws StreamReadException, DatabindException, IOException {
    logger.setLevel(Level.INFO);
    Employee emp = getEmployee();
    logger.log(Level.INFO,"Employee: {0} read successfully.", emp);
}

public static Employee getEmployee()
    throws StreamReadException, DatabindException, IOException{
    InputStream input = ReaderExample.class.getResourceAsStream("emp.json");
    // create ObjectMapper instance
    ObjectMapper objMapper = new ObjectMapper();
    return objMapper.readValue(input, Employee.class);
}
```

Dacă obiectul JSON este o reprezentare a clasei Employee, al doilea argument în `readValue()` va fi clasa respectivă.

```
public static void main(String[] args) throws FileNotFoundException, IOException {  
    JsonNode user = generate();  
    ObjectMapper om = new ObjectMapper();  
    try(OutputStream os = new FileOutputStream("createdNode.json")){  
        om.writeValue(os, user);  
    }  
}  
  
public static JsonNode generate(){  
    ObjectMapper mapper = new ObjectMapper();  
    // create a root JSON object  
    ObjectNode user = mapper.createObjectNode();  
    user.put("id", 1);  
    user.put("name", "John Doe");  
    user.put("email", "john.doe@example.com");  
    user.put("salary", 3545.99);  
    user.put("role", "QA Engineer");  
    user.put("admin", false);  
    // create a child JSON object  
    ObjectNode address = mapper.createObjectNode();  
    address.put("street", "2389 Radford Street");  
    address.put("city", "Horton");  
    address.put("state", "KS");  
    address.put("zipCode", 66439);  
    // append address to user  
    user.set("address", address);  
    return user;  
}
```

CREAREA PROGRAMATICĂ A UNUI OBIECT JSON

Clasa **JsonNode** este imutabilă. Pentru a modifica sau adăuga un nod folosim subclasa acestuia **ObjectNode**, care pune la depozitie metodele **put** și **set**.



PROCESARE JSON ÎN JAVA

JACKSON - OBJECTNODE

ObjectNode este o subclasă a clasei abstracte **JsonNode** și se folosește pentru a construi obiecte JSON

Metodă	Descriere
createObjectNode()	Se aplică pe o instanță ObjectMapper pentru a crea un ObjectNode
	<code>ObjectNode parentNode = objectMapper.createObjectNode();</code>
set()	Setează valoarea unui câmp JSON cu valoarea unui obiect JsonNode
	<code>parentNode.set("child1", childNode);</code>
put()	Setează valori primitive sau text pentru câmpurile unui obiect JSON
	<code>user.put("id", 1); user.put("name", "John Doe");</code>
remove()	Șterge un câmp dintr-un obiect JSON
	<code>objectNode.remove("fieldName");</code>

PREZENTARE JAX-RS

- Java EE 6 a introdus pentru prima dată specificația pentru **Java API for RESTful web services (JAX-RS)**
- **JAX-RS** (versiunea curentă 2.1) este parte integrantă platformei Java EE și asigură **portabilitatea codului REST în serverele de aplicație compatibile Java EE** (Jakarta EE) precum JBoss, Glassfish, IBM Websphere, Tomee
- *Apache Tomcat* este un **container web** care este **parțial compatibil cu specificațiile Java EE**, dar suportă **JAX-RS**. *Va fi folosit pe parcursul cursurilor de servicii web REST*
- Cele mai populare **implementări open-source ale JAX-RS** sunt: **Jersey, Apache CXF, RESTEasy, Restlet**
- **Jersey** – va fi utilizată în codul REST



JAX-RS - ADNOTĂRI

- **Folosirea adnotărilor** permit **crearea de servicii web REST la fel de ușor** precum o clasă POJO
- **Adnotările furnizează metadate claselor Java** ce vor fi folosite la **compilare, la instalare (deployment) și la rulare** pentru executarea sarcinilor atribuite

HTTP Request Methods	Resource	Request-Response Media Types	Request Parameters
<ul style="list-style-type: none">• @GET• @POST• @PUT• @DELETE• @HEAD• @OPTIONS	<ul style="list-style-type: none">• @Path	<ul style="list-style-type: none">• @Produces• @Consumes	<ul style="list-style-type: none">• @PathParam• @QueryParam• @MatrixParam• @HeaderParam• @CookieParam• @DefaultValue• @Context• @BeanParam• @Encoded

JAX-RS - ADNOTĂRI

- Resursele REST sunt **clase POJO** care folosesc adnotări pentru a implementa un serviciu web
- Fiecare resursă trebuie să aibă cel puțin o metodă adnotată cu **@Path** și cu una din adnotările **@HttpMethod**
- Doar **metodele publice** pot fi expuse ca metode de resurse
- **@Path** - indică URL-ul pentru resursa clasă sau metoda clasei care răspunde la cererea HTTP. Valoarea specificată în **@Path** este relativă la **URI-ul serverului** pe care resursa este găzduită
- Adnotarea poate fi aplicată la **nivel clasă și/sau la nivel metodă**
- Nu este necesar să se introducă / (slash)



JAX-RS - ADNOTĂRI

Vezi proiectul course9/restful

```
package atm.paradigms;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

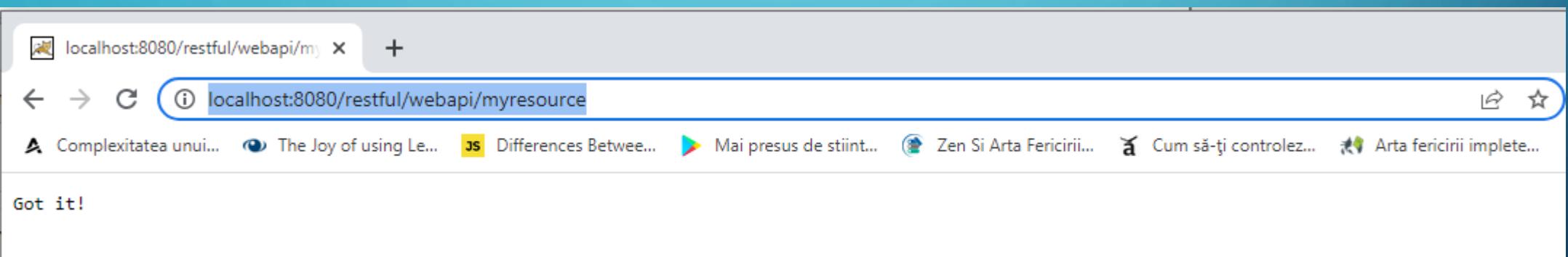
/**
 * Root resource (exposed at "myresource" path)
 */
@Path("myresource")
public class MyResource {
    /**
     * @return String that will be returned as a text/plain response.
     */
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

Implementarea unei clase resursă **MyResource** marcată ca resursă cu adnotarea **@Path**. Metoda resursă **getIt()** răspunde la cererile HTTP de tip GET și produce un conținut de tip **text/plain**.



JAX-RS - ADNOTĂRI

- URI-ul pentru resursă este <http://localhost:8080/restful/webapi/myresource> care are forma **http://host:port/<context-root>/<application-path>**
- < application-path > este în cazul nostru **myresource**

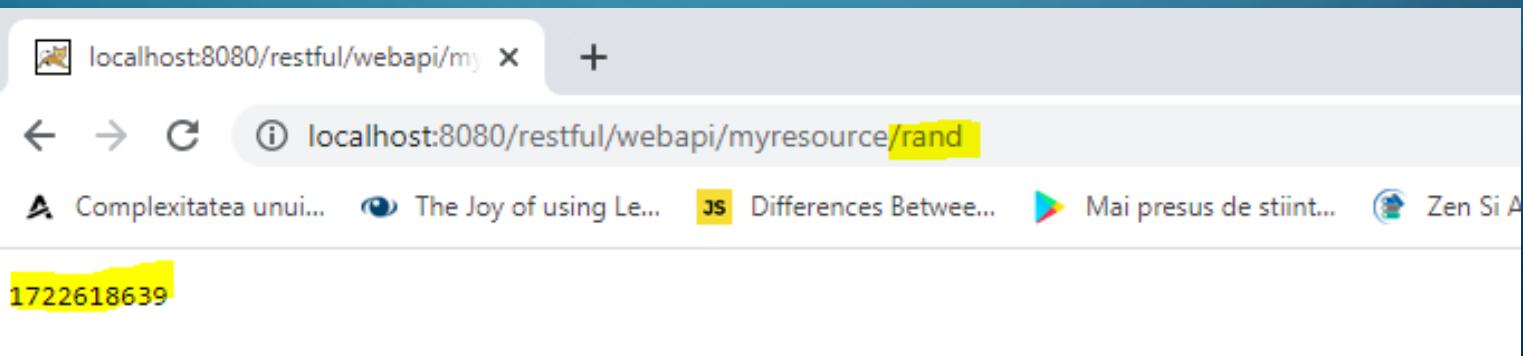


- Adăugam o metodă **getRand** pentru resursa cu URI **rand**
- **Calea către resursă** va fi
<http://localhost:8080/restful/webapi/myresource/rand>

JAX-RS - ADNOTĂRI

```
@Path("myresource")
public class MyResource {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }

    @GET
    @Path("rand")
    @Produces("text/plain")
    public Integer getRand(){
        return ThreadLocalRandom.current().nextInt();
    }
}
```





JAX-RS - ADNOTĂRI

- **Şablonul de cale URI** permite **definirea de variabile care apar ca substituent** în URI. **Variabilele sunt înlocuite la rulare cu valorile furnizate** de client
- Exemplu de **şablon de cale** URI `/myresource/{id}`
- Vom folosi utilitara `curl` pentru testarea serviciilor web (detalii la laborator)

```
package atm.paradigms;
import javax.ws.rs.*;
@Path("myresource")
public class MyResource {
    // . . .

    @DELETE
    @Path("object/{id}")
    @Produces("text/plain")
    public String getId(@PathParam("id") int id){
        return "You wanted me to delete ID: " + id;
    }
}
```

Metoda `getId()` are parametrul de cale `{id}` definit de adnotarea `@Path`. Pentru ca valoarea parametrului din URI să fie injectată metodei, se utilizează adnotarea `@PathParam` înaintea parametrului metodei.



JAX-RS - ADNOTĂRI

- Folosim **curl** pentru a testa cu metoda **DELETE** URI-ul

<http://localhost:8080/restful/webapi/myresource/object/120>

```
c:\ Command Prompt

0:\dev\tools\curl\bin>curl.exe -X "DELETE" http://localhost:8080/restful/webapi/myresource/object/120
You wanted me to delete ID: 120
0:\dev\tools\curl\bin>
```

- Nu funcționează în browser (suportă doar GET prin câmpul de adresă)





JAX-RS - ADNOTĂRI

- Un **serviciu REST** folosește header-ul **Content-type** care indică tipul conținutului din corpul cererii sau al răspunsului
- **@Produces** - definește **tipul media** Internet pe care metoda clasei îl returnează clientului

Tipurile produse sunt:

- *application/atom+xml*
- *application/json*
- *application/octet-stream*
- *application/svg+xml*
- *application/xhtml+xml*
- *application/xml*
- *text/html*
- *text/plain*
- *text/xml*



JAX-RS - ADNOTĂRI

```
package atm.paradigms.model;

public class Student {
    private int id;
    private String name;
    private String faculty;
    private int yearOfStudy;
    // important
    public Student() {
    }

    public Student(int id, String name,
                  String faculty, int yearOfStudy) {
        this.id = id;
        this.name = name;
        this.faculty = faculty;
        this.yearOfStudy = yearOfStudy;
    }

    // getters and setters methods
}
```

```
package atm.paradigms;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import atm.paradigms.model.Student;

@Path("atm")
public class StudentResource {
    @GET
    @Path("student/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Student getStudent(@PathParam("id") int id){
        return new Student(id, "Ion Popescu",
                           "FSISC", 3);
    }
}
```

Metoda **getStudent()** primește **id** ca parametru prin URI și returnează un obiect JSON serializat din instanța clasei **Student**.

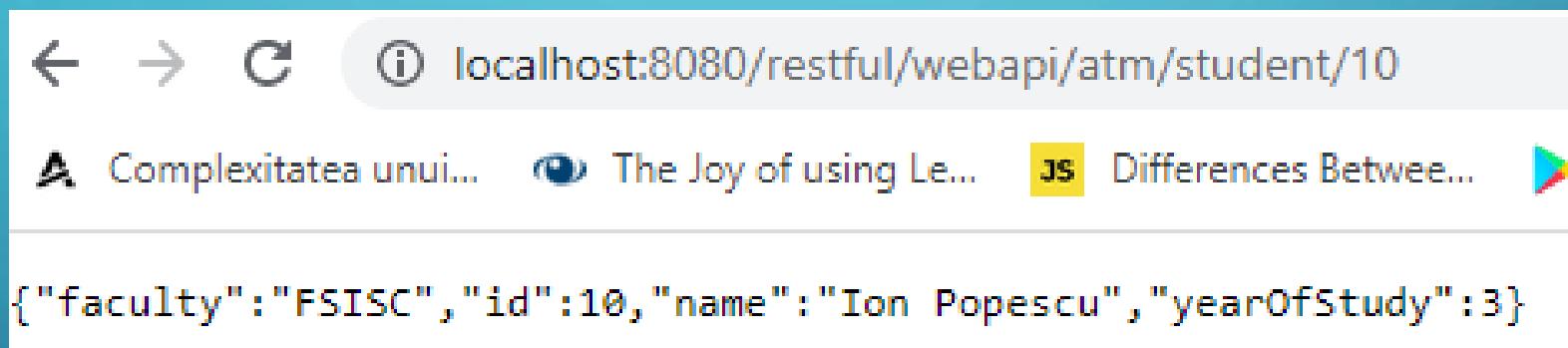


JAX-RS - ADNOTĂRI

- URL-ul pentru accesarea serviciului este

<http://localhost:8080/restful/webapi/atm/student/10>

- Rezultat returnat



```
Command Prompt

D:\dev\tools\curl\bin>curl.exe -X "GET" http://localhost:8080/restful/webapi/atm/student/10
{"faculty": "FSISC", "id": 10, "name": "Ion Popescu", "yearOfStudy": 3}
D:\dev\tools\curl\bin>
```



JAX-RS - ADNOTĂRI

- **@Consumes** - definește **tipul media Internet metoda clasei resursă îl poate accepta**. Poate fi folosit la nivel clasă sau metodă, dar **nivelul metodă suprascrie nivelul clasă**
- REST API poate consuma următoarele tipuri:
 - **application/atom+xml**
 - **application/json**
 - **application/octet-stream**
 - **application/svg+xml**
 - **application/xhtml+xml**
 - **application/xml**
 - **text/html**
 - **text/plain**
 - **text/xml**
 - **multipart/form-data**
 - **application/x-www-form-urlencoded**

JAX-RS - ADNOTĂRI

```
@Path("atm")
public class StudentResource {
    @GET
    @Path("student/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    // @Produces(MediaType.APPLICATION_XML)
    public Student getStudent(@PathParam("id") int id){
        return new Student(id, "Ion Popescu", "FSISC", 3);
    }

    @POST
    @Path("student")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Student echoRequest(Student student){
        student.setId(999);
        return student;
    }
}
```

- Avem o **implementare de metodă HTTP POST cu URL**
<http://localhost:8080/restful/webapi/atm/student>



JAX-RS - ADNOTĂRI

- Folosim **curl** pentru testare cu comanda:

```
curl -i -X "POST" -H "Content-Type: application/json" -d
"{"faculty": "FSISC", "id": 10, "name": "Ion
Popescu", "yearOfStudy": 3}"
http://localhost:8080/restful/webapi/atm/student
```

- Răspuns (**nu funcționează fără header-ul Content-Type**)

```
D:\dev\tools\curl\bin>curl -i -X "POST" -H "Content-Type: application/json" -d
"{"faculty": "FSISC", "id": 10, "name": "Ion Popescu", "yearOfStudy": 3}"
http://localhost:8080/restful/webapi/atm/student
HTTP/1.1 200
Content-Type: application/json
Content-Length: 65
Date: Fri, 23 Sep 2022 07:28:41 GMT

{"faculty": "FSISC", "id": 999, "name": "Ion Popescu", "yearOfStudy": 3}
D:\dev\tools\curl\bin>
```



JAX-RS - ADNOTĂRI

- Implementăm o metodă GET care returnează o listă de studenți. Poate fi accesată la URL <http://localhost:8080/restful/webapi/atm/student>

```
@Path("atm")
public class StudentResource {
    @GET
    @Path("student")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Student> getStudents()
        throws StreamReadException, DatabindException, IOException{
        List<Student> students = Tools.getStudents();
        return students;
    }
    // other methods
}
```

The screenshot shows a Java IDE interface with the following details:

- EXPLORER** panel: Shows a project structure with a **RESTFUL** folder containing **.vscode**, **src\main** (with **java** and **resources** subfolders), and a **students.json** file.
- POM.XML**: pom.xml
- CODE EDITOR**: StudentResource.java (selected tab)
- TOOLS**: Tools.java
- RESOURCES**: students.json (highlighted)
- STUDENT RESOURCE**: Student.java

The **students.json** file content is:

```
[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Irina Paraschiv", "yearOfStudy": 3}]
```

JAX-RS - ADNOTĀRI

```
package atm.paradigms.utils;
import java.io.*;
import java.util.List;
import com.fasterxml.jackson.core.exc.StreamReadException;
import com.fasterxml.jackson.databind.*;
import com.fasterxml.jackson.databind.type.CollectionType;

import atm.paradigms.model.Student;

public class Tools {
    public static List<Student> getStudents()
            throws StreamReadException, DatabindException, IOException{
        InputStream input = Tools.class.getResourceAsStream("/students.json");
        // create ObjectMapper instance
        ObjectMapper objMapper = new ObjectMapper();
        CollectionType collection = objMapper.getTypeFactory()
                .constructCollectionType(List.class, Student.class);
        List<Student> result = objMapper.readValue(input, collection);
        return result;
    }
}
```



JAX-RS - ADNOTĂRI

- Rezultatul afișat în *curl* și browser

```
D:\dev\tools\curl\bin>curl -i -X "GET" http://localhost:8080/restful/webapi/atm/student
HTTP/1.1 200
Content-Type: application/json
Content-Length: 200
Date: Fri, 23 Sep 2022 08:12:44 GMT

[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Irina Paraschiv", "yearOfStudy": 3}]
D:\dev\tools\curl\bin>
```

The screenshot shows a browser window with the address bar set to `localhost:8080/restful/webapi/atm/student`. The page content displays the same JSON array as the curl command output:

```
[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Irina Paraschiv", "yearOfStudy": 3}]
```



JAX-RS - ADNOTĀRI

[Cuprins](#)

```
@PUT
@Path("student/{id}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public List<Student> addStudent(@PathParam("id") int id, Student student)
    throws StreamReadException, DatabindException, IOException {
    List<Student> students = Tools.getStudents();
    long count = students.stream().filter(s -> s.getId() == id).count();
    if (count == 0L) {
        student.setId(id);
        List<Student> copyStudents = students.stream()
            .collect(toList());
        copyStudents.add(student);
        return copyStudents;
    } else {
        return students.stream()
            .map(s -> {
                if (s.getId() == id) {
                    s.setName(student.getName());
                    s.setFaculty(student.getFaculty());
                    s.setYearOfStudy(student.getYearOfStudy());
                }
                return s;
            }).collect(toList());
    }
}
```



JAX-RS - ADNOTĂRI

- Testare metodă @PUT

<http://localhost:8080/restful/webapi/atm/student/3>, pentru **ID existent**, cu comanda:

```
curl -i -X "PUT" -H "Content-Type: application/json" -d
```

```
"{\"faculty\":\"FSISC\",\"id\":10,\"name\":\"Vasile Vasilache\",\"yearOfStudy\":3}"
```

```
http://localhost:8080/restful/webapi/atm/student/3
```

- Rezultat

```
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

D:\dev\tools\curl\bin>curl -i -X "PUT" -H "Content-Type: application/json" -d "{\"faculty\":\"FSISC\",\"id\":10,\"name\":\"Vasile Vasilache\",\"yearOfStudy\":3}" http://localhost:8080/restful/webapi/atm/student/3
HTTP/1.1 200
Content-Type: application/json
Content-Length: 201
Date: Mon, 09 Jan 2023 08:51:43 GMT

[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Vasile Vasilache", "yearOfStudy": 3}]
D:\dev\tools\curl\bin>
```

Metoda PUT va modifica resursa existentă cu id = 3 cu datele trimise în conținutul cererii.



JAX-RS - ADNOTĂRI

- **Testare metodă @PUT**

<http://localhost:8080/restful/webapi/atm/student/4>, pentru **ID nonexistent**, cu comanda:

```
curl -i -X "PUT" -H "Content-Type: application/json" -d
"{"faculty": "FSISC", "id": 10, "name": "Vasile Vasilache", "yearOfStudy": 3}"
http://localhost:8080/restful/webapi/atm/student/4
```

```
D:\dev\tools\curl\bin>curl -i -X "PUT" -H "Content-Type: application/json" -d "{\"faculty\": \"FSISC\", \"id\": 10, \"name\": \"Vasile Vasilache\", \"yearOfStudy\": 3}" http://localhost:8080/restful/webapi/atm/student/4
HTTP/1.1 200
Content-Type: application/json
Content-Length: 269
Date: Fri, 23 Sep 2022 10:30:40 GMT

[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Irina Paraschiv", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 4, "name": "Vasile Vasilache", "yearOfStudy": 3}]
D:\dev\tools\curl\bin>
```

Metoda PUT va crea o nouă resursă cu id=4, folosind datele trimise în conținutul cererii.



JAX-RS - ADNOTĂRI

- Implementare și testare metodă `@DELETE`

<http://localhost:8080/restful/webapi/atm/student/3>:

```
@DELETE  
@Path("student/{id}")  
@Produces(MediaType.APPLICATION_JSON)  
public List<Student> deleteStudent(@PathParam("id") int id)  
    throws StreamReadException, DatabindException, IOException {  
    List<Student> students = Tools.getStudents();  
    return students.stream()  
        .filter(s -> s.getId() != id)  
        .collect(toList());  
}
```

```
D:\dev\tools\curl\bin>curl -i -X "DELETE" http://localhost:8080/restful/webapi/atm/student/3  
HTTP/1.1 200  
Content-Type: application/json  
Content-Length: 132  
Date: Fri, 23 Sep 2022 10:39:36 GMT  
  
[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}]  
D:\dev\tools\curl\bin>
```



JAX-RS - ADNOTĂRI

- **Şablonul de cale URI are o parte care indică resursa dar poate primi și parametrii** (așa cum am văzut deja)
- **Adnotarea @PathParam injectează valoarea parametrului din cale într-un parametru al metodei sau un câmp al clasei**
- **Adnotarea este folosită cu metodele HTTP @GET, @POST, @PUT și @DELETE**

```
@GET  
@Path("test/{param1}/{param2}")  
@Produces(MediaType.APPLICATION_JSON)  
public String getParams(@PathParam("param1") String p1,  
                        @PathParam("param2") String p2){  
    ObjectMapper objMapper = new ObjectMapper();  
    ObjectNode root = objMapper.createObjectNode();  
    root.put("Param1", p1);  
    root.put("Param2", p2);  
    return root.toString();  
}
```



JAX-RS - ADNOTĂRI

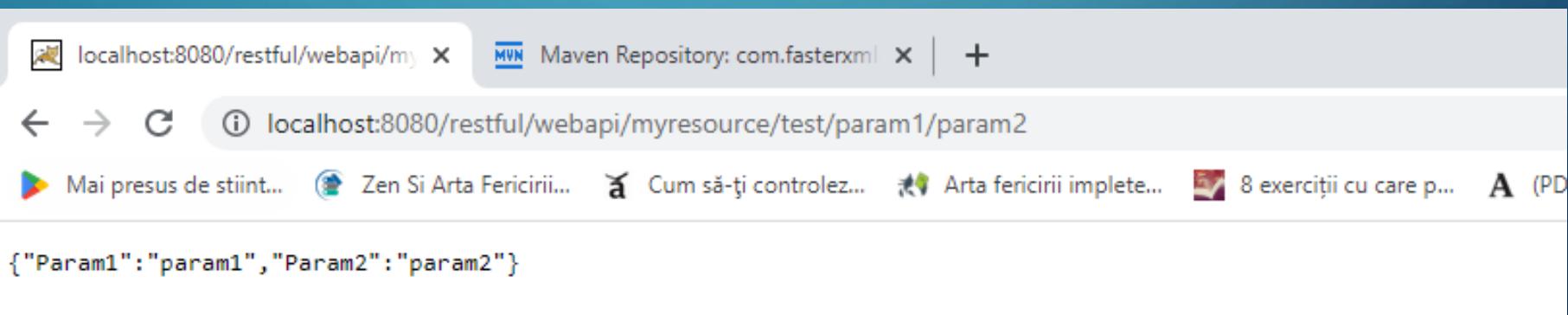
- Resursa se poate testa cu **curl** sau în browser pe calea

<http://localhost:8080/restful/webapi/myresource/test/param1/param2>

```
Command Prompt

D:\dev\tools\curl\bin>curl.exe -i -X "GET" http://localhost:8080/restful/webapi/myresource/test/param1/param2
HTTP/1.1 200
Content-Type: application/json
Content-Length: 37
Date: Mon, 26 Sep 2022 09:53:17 GMT

>{"Param1":"param1","Param2":"param2"}
D:\dev\tools\curl\bin>
```





JAX-RS - ADNOTĂRI

- Adnotarea `@QueryParam` injectează valoarea parametrului HTTP query într-un parametru al metodei sau într-un câmp al clasei
- Resursa se poate testa cu `curl` sau în browser pe calea
<http://localhost:8080/restful/webapi/myresource/test?param1=test>

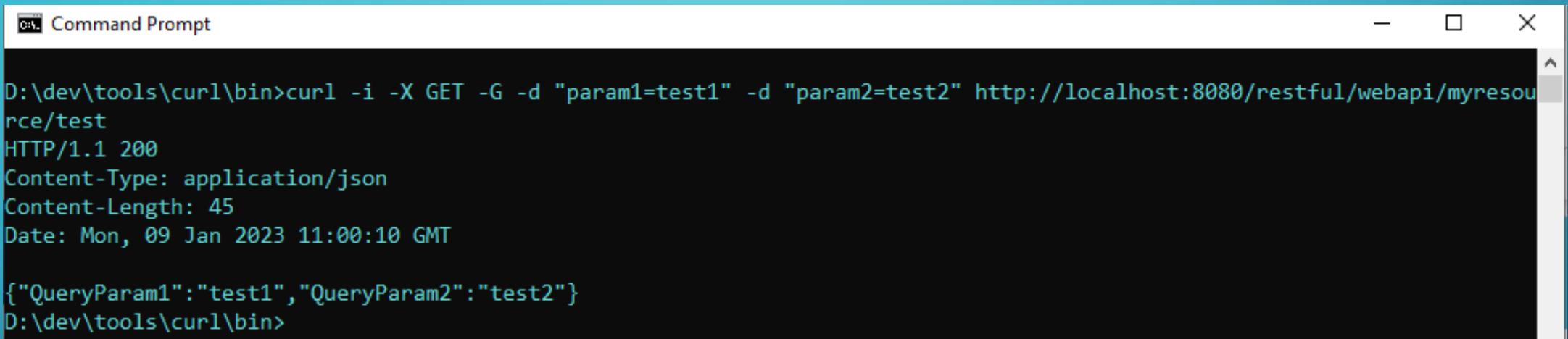
```
@GET  
@Path("test")  
@Produces(MediaType.APPLICATION_JSON)  
public String getQueryParams(@QueryParam("param1") String param){  
    ObjectMapper objMapper = new ObjectMapper();  
    ObjectNode root = objMapper.createObjectNode();  
    root.put("QueryParam", param);  
    return root.toString();  
}
```

JAX-RS - ADNOTĂRI

```
@GET  
@Path("test")  
@Produces(MediaType.APPLICATION_JSON)  
public String getQueryParams(@QueryParam("param1") String p1,  
                             @QueryParam("param2") String p2){  
    ObjectMapper objMapper = new ObjectMapper();  
    ObjectNode root = objMapper.createObjectNode();  
    root.put("QueryParam1", p1);  
    root.put("QueryParam2", p2);  
    return root.toString();  
}
```

JAX-RS - ADNOTĂRI

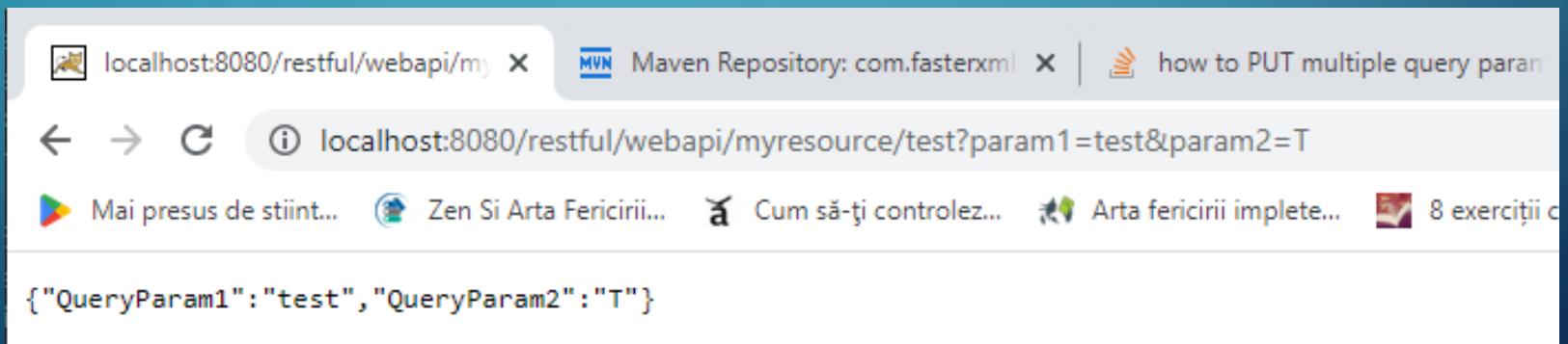
```
curl -i -X GET -G -d "param1=test1" -d "param2=test2" http://localhost:8080/restful/webapi/myresource/test
```



```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X GET -G -d "param1=test1" -d "param2=test2" http://localhost:8080/restful/webapi/myresource/test
HTTP/1.1 200
Content-Type: application/json
Content-Length: 45
Date: Mon, 09 Jan 2023 11:00:10 GMT

>{"QueryParam1":"test1","QueryParam2":"test2"}
D:\dev\tools\curl\bin>
```



JAX-RS - ADNOTĂRI

- **Adnotarea `@HeaderParam` injectează valorile din header-ele HTTP în parametrul unei metode sau în câmpul unei clase**
- **Adnotarea `@CookieParam` injectează valoarea parametrilor cookie prezenți din header-ele HTTP în parametrul unei metode sau în câmpul unei clase**



JAX-RS - ADNOTĂRI

- Adnotarea `@FormParam` injectează valoare unui parametru prezent în forma HTML într-un parametru al metodei sau într-un câmp al clasei

```
@POST  
@Path("test")  
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)  
@Produces(MediaType.APPLICATION_JSON)  
public String getFormParams(@FormParam("param1") String p1,  
                           @FormParam("param2") String p2){  
    ObjectMapper objMapper = new ObjectMapper();  
    ObjectNode root = objMapper.createObjectNode();  
    root.put("FormParam1", p1);  
    root.put("FormParam2", p2);  
    return root.toString();  
}
```

```
D:\dev\tools\curl\bin>curl -i -X POST -H "Content-Type: application/x-www-form-urlencoded" -d "param1=value1&param2=value2" http://localhost:8080/restful/webapi/myresource/test  
HTTP/1.1 200  
Content-Type: application/json  
Content-Length: 45  
Date: Mon, 26 Sep 2022 10:44:46 GMT  
  
{"FormParam1": "value1", "FormParam2": "value2"}  
D:\dev\tools\curl\bin>
```



JAX-RS - ADNOTĂRI

- Adnotarea **@DefaultValue** specifică **valoare implicită** când **parametrii corespunzători din cerere lipsesc** pentru următoarele adnotări **PathParam**, **QueryParam**, **MatrixParam**, **CookieParam**, **FormParam** și **HeaderParam**

```
@GET
```

```
@Produces(MediaType.APPLICATION_JSON)  
public List<Department> findAllDepartmentsInRange  
(@DefaultValue("0") @QueryParam("from") Integer from,
```

- Adnotarea **@Context** poate fi folosită pentru **accesarea diferitelor obiecte de context** precum **calea URI**, **cererea HTTP**, **header HTTP**, **securitate** etc.
- JAX-RS permite referențierea obiectelor de context în cod folosind **dependency injection**



JAX-RS - ADNOTĂRI

```
@GET  
@Path("test/{name}")  
@Produces(MediaType.APPLICATION_JSON)  
public String getContext(@Context UriInfo uriInfo){  
    String name = uriInfo.getPathParameters().getFirst("name");  
    ObjectMapper objMapper = new ObjectMapper();  
    ObjectNode node = objMapper.createObjectNode();  
    node.put("NameParam", name);  
    return node.toString();  
}
```

Se folosește adnotarea `@Context` pentru accesarea parametrului `name` primit prin URI.

```
curl -i -X GET http://localhost:8080/restful/webapi/myresource/test/Ionel
```

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/restful/webapi/myresource/test/Ionel

HTTP/1.1 200

Content-Type: application/json

Content-Length: 21

Date: Mon, 26 Sep 2022 12:15:44 GMT

[{"NameParam": "Ionel"}]

D:\dev\tools\curl\bin>

JAX-RS - RESPONSE

- Resursele implementate până acum au returnat obiecte Java ca răspuns la cereri
- Ce se întâmplă dacă vrem să returnăm header-re adiționale, cookies, coduri de stare?
- JAX-RS permite returnarea de metadate suplimentare folosind clasa ***javax.ws.rs.core.Response***
- Reimplementăm metoda `getStudents` folosind `Response` și adăugam header-ul `Cache-Control`
- Metoda poate fi accesată pe calea
<http://localhost:8080/restful/webapi/atm/student>

JAX-RS - RESPONSE

```

@GET
@Path("student")
@Produces(MediaType.APPLICATION_JSON)
public Response getStudents() throws StreamReadException, DatabindException, IOException {
    List<Student> students = Tools.getStudents();
    // Sets cache control directive to the response
    CacheControl cacheControl = new CacheControl();
    // Cache the result for a day
    cacheControl.setMaxAge(86400);
    return Response.ok().cacheControl(cacheControl)
        .entity(students).build();
}
  
```

Command Prompt

```

D:\dev\tools\curl\bin>curl -i -X "GET" http://localhost:8080/restful/webapi/atm/student
HTTP/1.1 200
Cache-Control: no-transform, max-age=86400
Content-Type: application/json
Content-Length: 200
Date: Mon, 26 Sep 2022 12:29:40 GMT

[{"faculty": "FSISC", "id": 1, "name": "Ion Popescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 2, "name": "Razvan Ionescu", "yearOfStudy": 3}, {"faculty": "FSISC", "id": 3, "name": "Irina Paraschiv", "yearOfStudy": 3}]
D:\dev\tools\curl\bin>
  
```

Folosirea clasei **Response** pentru a crea un răspuns customizat. Se observă o înlățuirea de metode statice precum: **ok()**, **status()**, **entity()**, **cacheControl()** etc. care returnează **ResponseBuilder**. Ultima metodă este **build()** care returnează **Response**.

JAX-RS - RESPONSE

- JAX-RS folosește clasa *javax.ws.rs.ext.MessageBodyReader* pentru **a mapa conținutul cereri HTTP la tipul Java corespunzător**
- *javax.ws.rs.ext.MessageBodyWriter* pentru **maparea tipurilor Java returnate de clasa resursă la conținutul mesajului de răspuns HTTP corespunzător** precum JSON, XML și text
- **MessageBodyReader și MessageBodyWriter generează excepția *javax.ws.rs.WebApplicationException* dacă nu știe cum să convertească datele de intrare**

IMPLEMENTARE SERVICIU REST

- Pe baza cunoștințelor JAX-RS acumulate vom dezvolta un serviciu REST cu un strat de persistență într-o bază de date MySQL
- Resursa va conține câte un endpoint (URI) pentru fiecare operație **CRUD** (*create, read, update, and delete*) pe baza de date
- Deoarece Apache Tomcat nu implementează **Java (Jakarta) Persistence API (JPA)** nu putem folosi soluții Object-Relational Mapping (ORM) precum *Hibernate* sau *EclipseLink*
- Vom utiliza soluția de persistență **MyBatis** care simplifică mult codul comparativ cu **Java Database Connectivity (JDBC)**
- Detalii privind configurarea proiectului folosind Maven vor fi prezentate la laborator

BIBLIOGRAFIE

- **RESTful Java Web Services, Third Edition**, Bogunuva Mohanram Balachandar, Packt Publishing, 2017
- **RESTful Java with JAX-RS 2.0, SECOND EDITION**, Bill Burke, O'Reilly, 2014
- **RESTful Java Web Services**, Jose Sandoval, Packt Publishing, 2009