



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

PARADIGME DE PROGRAMARE (ÎN JAVA)

CURS 10 - 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** (2/2)
- Quarkus Framework (4)

CUPRINS CURS

- Prezentare MyBatis
- Implementare serviciu REST
- Client REST
- Securizarea serviciilor REST
- Bibliografie



PREZENTARE MYBATIS

- **MyBatis este o soluție open source de persistență Java care leagă obiectele Java de comenzi SQL sau proceduri stocate folosind descriptori XML sau adnotări**
- Spre deosebire de **soluțiile ORM** (Object-relational mapping), **MyBatis nu mapează obiectele la tabele** din baza de date
- **MyBatis mapează metode la comenzi SQL și suportă întreaga funcționalitate a bazei de date (proceduri stocate, view-uri, interogări complexe)**
- **MyBatis pune la dispoziție suportul pentru maparea rezultatelor comenziilor SQL la o structură de obiecte**
- **Mult mai simplu decât JDBC** (Java Database Connectivity)
- Proiectul folosit va fi pus la dispoziție ca material didactic (project course 10/mybatis)



PREZENTARE MYBATIS

- Pentru **conectarea la baza de date MySQL**, MyBatis folosește un **fișier de configurare XML** (se poate face și din cod fără XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/testdb"/>
        <property name="username" value="user"/>
        <property name="password" value="qaz123QAZ!@#"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="mymapper.xml"/>
  </mappers>
</configuration>
```



PREZENTARE MYBATIS

- Prin **intermediul** interfeței **MyMapper** se mapează o metodă la o comandă SQL specificată în adnotare

```
package atm.paradigms;

import java.util.List;
import org.apache.ibatis.annotations.Select;

public interface MyMapper {

    @Select("SELECT VERSION()")
    public String getMySQLVersion();

    // ...
}
```

- Adnotarea **@Select** leagă metoda **getMySQLVersion()** de comanda SQL **SELECT VERSION()**

PREZENTARE MYBATIS

- Ne conectăm la baza de date și citim versiunea. Pentru acesta **citim fișierul de configurare și adăugăm interfața MyMapper. Deschidem sesiunea și citim datele**

```

package atm.paradigms;
import java.io.IOException;
import java.io.Reader;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.*;

public class SQLVersionTest {
    private static SqlSessionFactory factory = null;
    public static void main(String[] args) throws IOException {
        String resource = "mybatis-config.xml";
        Reader reader = Resources.getResourceAsReader(resource);
        factory = new SqlSessionFactoryBuilder().build(reader);
        factory.getConfiguration().addMapper(MyMapper.class);
        reader.close();
        try (SqlSession session = factory.openSession();){
            // method with annotation from interface
            String version = session.selectOne("getMySQLVersion");
            System.out.println(version);
        }
    }
}

```

Rezultat:
8.0.30

Se creează o instanță **SqlSessionFactory** la care adăugăm interfața **MyMapper**. Se deschide o sesiune la baza de date (**openSessions()**) cu *try-with-resources*. Se invocă declarativ pe sesiune, cu **selectOne()**, metoda **getMySQLVersion**.



PREZENTARE MYBATIS

- Același lucru **folosind descriptor XML** în loc de adnotări
- **Se creează un fișier XML numit mymapper.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="atm.paradigms">
    <select id="mysqlVersion" resultType="String">
        SELECT VERSION()
    </select>
</mapper>
```

- În **fișierul de configurare** se marchează acest fișier cu tag-ul **mappers**

```
    ...
<mappers>
    <mapper resource="mymapper.xml"/>
</mappers>
</configuration>
```

PREZENTARE MYBATIS

```
package atm.paradigms;
import java.io.IOException;
import java.io.Reader;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.*;

public class SQLVersionTest {
    private static SqlSessionFactory factory = null;
    public static void main(String[] args) throws IOException {
        String resource = "mybatis-config.xml";
        Reader reader = null;
        reader = Resources.getResourceAsReader(resource);
        factory = new SqlSessionFactoryBuilder().build(reader);
        reader.close();
        try (SqlSession session = factory.openSession();) {
            // mapping with xml file
            String version1 = session.selectOne("mysqlVersion");
            System.out.println(version1);
        }
    }
}
```

Rezultat:
8.0.30



PREZENTARE MYBATIS

- Folosind **descriptori XML** se pot **crea interogări XML dinamice** folosind tag-uri precum **<if>**, **<where>**, **<foreach>** etc.

```
<mapper namespace="atm.paradigms">
    <select id = "getBook" resultType = "atm.paradigms.model.Book">
        SELECT * FROM MyBooks
        <where>
            <if test = "_parameter != null">
                Id = #{id}
            </if>
        </where>
    </select>
</mapper>
```

- Tag-ul **<where>** este inclus doar dacă parametrul **Id** nu este null
- Expresia returnează o carte identificată de **Id** sau toate cărțile dacă **Id** este null

PREZENTARE MYBATIS

```

package atm.paradigms;
import java.io.IOException;
import java.util.List;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import atm.paradigms.model.Book;
import atm.paradigms.utils.Tools;

public class DynamicSqlTest {
    public static void main(String[] args) throws IOException {
        SqlSessionFactory factory = Tools.getSqlSessionFactory();
        try (SqlSession session = factory.openSession();) {
            Book book = session.selectOne("getBook", 1);
            System.out.println(book);
            List<Book> books = session.selectList("getBook");
            books.forEach(System.out::println);
        }
    }
}

```

Mod de utilizare a metodei **getBook()**, definită prin descriptorul XML (**mymapper.xml**), prezentată anterior. Se folosește cu parametrul **ID** pentru a obține o carte particulară (**selectOne()**). Se folosește **fără parametru** pentru a obține lista cărților (**selectList()**).

*Book [author=Leo Tolstoy, id=1, published=1869, remark=Napoleonic wars, title=War and Peace]
 Book [author=Leo Tolstoy, id=1, published=1869, remark=Napoleonic wars, title=War and Peace]
 ...*

PREZENTARE MYBATIS

- Introducem clasa **Book** sau **bean** (constructor cu zero argumente, getter-e și setter-e)

```
package atm.paradigms.model;

public class Book {
    private Long id;
    private String author;
    private String title;
    private int published;
    private String remark;

    public Book() {
    }
    public Book(String author, String title, int published, String remark) {
        this.author = author;
        this.title = title;
        this.published = published;
        this.remark = remark;
    }
    // getters, setters and toString
}
```

PREZENTARE MYBATIS

- În exemplele următoare **vom folosi modul de mapare bazat pe anotări**
- Având în vedere că obiectul **SqlSessionFactory** este **thread safe** implementăm o **metodă statică de tip Singleton** care returnează instanța clasei

```
package atm.paradigms.utils;
import java.io.IOException;
import java.io.Reader;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.*;
import atm.paradigms.MyMapper;

public class Tools {
    private static SqlSessionFactory factory = null;
    // thread safe method
    public static SqlSessionFactory getSqlSessionFactory() throws IOException{
        if (factory == null){
            String resource = "mybatis-config.xml";
            Reader reader = Resources.getResourceAsReader(resource);
            factory = new SqlSessionFactoryBuilder().build(reader);
            factory.getConfiguration().addMapper(MyMapper.class);
            reader.close();
        }
        return factory;
    }
}
```

Metoda **getSqlSessionFactory()** returnează valoarea câmpului **factory**, de tip **SqlSessionFactory**, dacă este nenulă. Dacă este nulă creează o instanță **SqlSessionFactory**, pe baza fișierului de configurare, și adaugă la ea interfața **MyMapper**.



PREZENTARE MYBATIS

- **Script SQL** utilizat pentru crearea schemei și popularea bazei de date **testdb**

```
CREATE TABLE MyBooks(Id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
Author VARCHAR(30), Title VARCHAR(60), Published INTEGER, Remark VARCHAR(150));  
  
INSERT INTO MyBooks(Author, Title, Published, Remark) VALUES ('Leo Tolstoy',  
'War and Peace', 1869, 'Napoleonic wars');  
INSERT INTO MyBooks(Author, Title, Published, Remark) VALUES ('Leo Tolstoy',  
'Anna Karenina', 1878, 'Greatest book of love');  
INSERT INTO MyBooks(Author, Title, Published, Remark) VALUES ('Jeff Prosise',  
'Programming Windows with MFC', 1999, 'Classic book about MFC');  
INSERT INTO MyBooks(Author, Title, Published, Remark) VALUES ('Tom Marrs',  
'JBoss at Work', 2005, 'JBoss practical guide');  
INSERT INTO MyBooks(Author, Title, Published, Remark) VALUES ('Debu Panda',  
'EJB3 in Action', 2007, 'Introduction to Enterprise Java Beans');
```

PREZENTARE MYBATIS

```
package atm.paradigms;
import java.util.List;
import org.apache.ibatis.annotations.*;
import atm.paradigms.model.Book;

public interface MyMapper {
    @Select("SELECT * FROM MyBooks WHERE Id = #{id}")
    public Book getBookById(Long id);

    @Select("SELECT * FROM MyBooks WHERE Author = #{author}")
    public List<Book> getBooksByAuthor(String author);

    @Insert("INSERT INTO MyBooks(Author, Title, Published, Remark) "
            + "VALUES(#{author}, #{title}, #{published}, #{remark})")
    public void insertBook(String author, String title, int published,
                          String remark);

    @Delete("DELETE FROM MyBooks WHERE Id = #{id}")
    public void deleteBookById(Long id);

    @Update("UPDATE MyBooks SET Author=#{author}, Title=#{title}, "
            + "Published=#{published}, Remark=#{remark} WHERE Id = #{id}")
    public void updateBook(Book book);
}
```

Maparea diferitelor comenzi
SQL (CRUD) la metode



PREZENTARE MYBATIS

```
package atm.paradigms;
// imports ...
public class MyBatisBooks {
    public static void main(String[] args) throws IOException {
        SqlSessionFactory factory = Tools.getSqlSessionFactory();
        try (SqlSession session = factory.openSession();) {
            Book book = session.selectOne("getBookById", 4L);
            System.out.println(book);
            List<Book> books = session.selectList("getBooksByAuthor", "Leo Tolstoy");
            books.forEach(System.out::println);
            Book newBook = new Book("Miguel de Cervantes", "Don Quixote", 1605, "First modern novel");
            session.insert("insertBook", newBook);
            // delete book by Id
            session.delete("deleteBookById", 8L);
            // update book by Id
            book.setAuthor("Lev Tolstoi");
            session.update("updateBook", book);
            session.commit();
        }
    }
}
```

Se testează diferitele metode mapate la comenzi SQL. Se observă că pentru **insertBook()** putem trece direct obiectul și nu câmpurile acestuia.
În cazul modificărilor (insert, update, delete) se invocă metoda **commit()** pe sesiune.

IMPLEMENTARE SERVICIU REST

- Serviciul REST **BooksService** cu resursa REST **book** asociată va fi implementat folosind **exemplele MyBatis anterioare și cunoștințele acumulate în cursul 9**
- Proiectul este generat folosind arhetipul Maven **jersey-quickstart-webapp** din grupul **org.glassfish.jersey.archetypes**, care se adaugă **suportul pentru Jackson** și **MyBatis** (detalii la laborator)
- Se observă în proiect elementele dezvoltate la prezentarea MyBatis: fișierul XML de configurare **mybatis-config.xml**, **Book** bean și clasa **Tools**
- Interfața **Mapper** conține metodele și adnotările de mapare la operații pe baza de date
- Structura proiectului este prezentată în slide-ul următor



The screenshot shows a Java project structure in the Explorer view, with a file named `BooksService.java` selected. The code implements a REST endpoint for retrieving books using MyBatis and Jackson. In the Servers view, a Tomcat 9 instance is running. A green tooltip box highlights the deployment path to the Tomcat container.

```
src > main > java > atm > paradigms > BooksService.java > BooksService
17 import javax.ws.rs.core.Response.Status;
18
19 import org.apache.ibatis.session.SqlSession;
20 import org.apache.ibatis.session.SqlSessionFactory;
21
22 import com.fasterxml.jackson.databind.ObjectMapper;
23 import com.fasterxml.jackson.databind.node.ObjectNode;
24
25 import atm.paradigms.model.Book;
26 import atm.paradigms.utils.Tools;
27
28 @Path("book")
29 public class BooksService {
30     private static final Logger logger = Logger.getLogger(BooksService.class.getName());
31
32     @GET
33     @Produces(MediaType.APPLICATION_JSON)
34     public Response getAllBooks(){
35         try {
36             SqlSessionFactory factory = Tools.getSqlSessionFactory();
37             try(SqlSession session = factory.openSession()){
38                 List<Book> books = session.selectList("getBooks");
39                 return Response.ok().entity(books).build();
40             }
41         } catch (Exception e) {
42             logger.log(Level.SEVERE, e.toString());
43         }
44     }
45 }
```

Server: apache-tomcat-9.0.41 (Started) (Un)

Proiectul se instalează ca aplicație web și se rulează din containerul web **Tomcat 9**.
Detalii la laborator

Proiectul se instalează ca aplicație web și se rulează din containerul web **Tomcat 9**.
Detalii la laborator



IMPLEMENTARE SERVICIU REST

GET ALL BOOKS

```

@Path("book")
public class BooksService {
    private static final Logger logger = Logger.getLogger(BooksService.class.getName());

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getAllBooks(){
        try {
            SqlSessionFactory factory = Tools.getSqlSessionFactory();
            try(SqlSession session = factory.openSession()){
                List<Book> books = session.selectList("getBooks");
                return Response.ok().entity(books).build();
            }
        } catch (Exception e) {
            logger.log(Level.SEVERE, e.toString());
            ObjectMapper om = new ObjectMapper();
            ObjectNode node = om.createObjectNode();
            node.put("message", "An error occurred. Check on server.");
            return Response.status(Status.INTERNAL_SERVER_ERROR).entity(node.toString()).build();
        }
    }
}

```

```

public interface Mapper {
    @Select("SELECT * FROM MyBooks ORDER BY Id")
    public List<Book> getBooks();
}

```

Se utilizează **Response** pentru generarea și trimiterea răspunsului HTTP reprezentat ca vector JSON.

În caz de eroare va trimite un obiect JSON cu detalii.



IMPLEMENTARE SERVICIU REST

GET ALL BOOKS

- **Accesarea cu metoda GET a resursei cu URL <http://localhost:8080/restful-service/webapi/book> returnează lista cărților reprezentată ca vector JSON.**
- Cod: 200(OK)

```
PS C:\Select Command Prompt
D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/restful-service/webapi/book
HTTP/1.1 200
Content-Type: application/json
Content-Length: 777
Date: Thu, 29 Sep 2022 06:40:19 GMT

[{"author": "Leo Tolstoy", "id": 1, "published": 1869, "remark": "Napoleonic wars", "title": "War and Peace"}, {"author": "Leo Tolstoy", "id": 2, "published": 1878, "remark": "Greatest book of love", "title": "Anna Karenina"}, {"author": "Jeff Prosise", "id": 3, "published": 1999, "remark": "Classic book about MFC", "title": "Programming Windows with MFC"}, {"author": "Lev Tolstoi", "id": 4, "published": 2005, "remark": "JBoss practical guide", "title": "JBoss at Work"}, {"author": "Debu Panda", "id": 5, "published": 2007, "remark": "Introduction to Enterprise Java Beans", "title": "EJB3 in Action"}, {"author": "Miguel de Cervantes", "id": 9, "published": 1605, "remark": "First modern novel", "title": "Don Quixote"}, {"author": "Miguel de Cervantes", "id": 10, "published": 1605, "remark": "First modern novel", "title": "Don Quixote"}]
D:\dev\tools\curl\bin>
```

IMPLEMENTARE SERVICIU REST

GET BOOK BY ID

```
@Select("SELECT * FROM MyBooks WHERE Id = #{id}")
public Book getBookById(Long id);

@GET
@Path("{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response getABooksById(@PathParam("id") long id){
    try {
        SqlSessionFactory factory = Tools.getSqlSessionFactory();
        try(SqlSession session = factory.openSession()){
            Book book = session.selectOne("getBookById", id);
            return Response.ok().entity(book).build();
        }
    } catch (Exception e) {
        logger.log(Level.SEVERE, e.toString());
        ObjectMapper om = new ObjectMapper();
        ObjectNode node = om.createObjectNode();
        node.put("message", "An error occurred. Check on server.");
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(node.toString()).build();
    }
}
...
...
```

IMPLEMENTARE SERVICIU REST

GET BOOK BY ID

- Accesarea cu metoda **GET** a endpoint-ului cu URL

<http://localhost:8080/restful-service/webapi/book/1> returnează cartea cu
Id egal cu 1 în reprezentare JSON. Cod: 200(OK)

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/restful-service/webapi/book/1
HTTP/1.1 200
Content-Type: application/json
Content-Length: 99
Date: Thu, 29 Sep 2022 06:46:16 GMT

{"author": "Leo Tolstoy", "id": 1, "published": 1869, "remark": "Napoleonic wars", "title": "War and Peace"}
D:\dev\tools\curl\bin>
```



IMPLEMENTARE SERVICIU REST

POST NEW BOOK

```

@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response addBook(Book book){
    ObjectMapper om = new ObjectMapper();
    try {
        SqlSessionFactory factory = Tools.getSqlSessionFactory();
        try(SqlSession session = factory.openSession()){
            session.insert("insertBook", book);
            session.commit();
            ObjectNode node = om.createObjectNode();
            node.put("message", "Book successfully inserted.");
            return Response.status(Status.CREATED).entity(node.toPrettyString()).build();
        }
    } catch (Exception e) {
        logger.log(Level.SEVERE, e.toString());
        ObjectNode node = om.createObjectNode();
        node.put("message", "An error occurred. Check on server.");
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(node.toString()).build();
    }
}

```

```

@Insert("INSERT INTO MyBooks(Author, Title, Published, Remark) "
+ "VALUES(#{author}, #{title}, #{published}, #{remark})")
public void insertBook(Book book);

```

Cartea trimisă în reprezentare JSON de client, cu HTTP POST, este deserializată și inserată în baza de date.
Este necesar ca tranzacția să fie comisă explicit cu metoda **commit()**.
Folosind **Response**, se trimit un mesaj de succes sau eroare în reprezentare JSON.



IMPLEMENTARE SERVICIU REST

POST NEW BOOK

- Accesarea și trimitera unei cărți în reprezentare JSON cu metoda POST la endpoint-ul cu URL <http://localhost:8080/restful-service/webapi/book/>, în vederea salvării în baza de date. Returnează un mesaj în format JSON cu succesul operațiunii. Coduri: 200(OK), 201 (Created), 204 (No Content)

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X "POST" -H "Content-Type: application/json" -d "{\"author\":\"Marin Preda\",\"id\":10,\"published\":1980,\"remark\":\"Romania after 1945\",\"title\":\"Cel mai iubit dintre pamanteni\"}" http://localhost:8080/restful-service/webapi/book
HTTP/1.1 201
Content-Type: application/json
Content-Length: 49
Date: Thu, 29 Sep 2022 06:56:03 GMT

{
  "message" : "Book successfully inserted."
}
D:\dev\tools\curl\bin>
```

IMPLEMENTARE SERVICIU REST

POST NEW BOOK

The screenshot shows the MySQL Workbench interface with the following details:

- Title Bar:** MySQL Workbench, Local instance MySQL80.
- File Menu:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Includes icons for SQL, XML, CSV, PDF, and various export options.
- Navigator:** Shows the database schema with **testdb** selected, containing **Tables** like **mybooks**, **Views**, **Stored Procedures**, and **Functions**.
- Query Editor:** Title **Query 1**, tab **mybooks**. The query is:

```
1 • use testdb;
2 • SELECT * FROM MyBooks;
```
- Result Grid:** Displays the results of the query. The columns are **Id**, **Author**, **Title**, **Published**, and **Remark**. The data includes:

| | Id | Author | Title | Published | Remark |
|---|----|---------------------|--------------------------------|-----------|---------------------------------------|
| ▶ | 1 | Leo Tolstoy | War and Peace | 1869 | Napoleonic wars |
| ▶ | 2 | Leo Tolstoy | Anna Karenina | 1878 | Greatest book of love |
| ▶ | 3 | Jeff Prosise | Programming Windows with MFC | 1999 | Classic book about MFC |
| ▶ | 4 | Lev Tolstoi | JBoss at Work | 2005 | JBoss practical guide |
| ▶ | 5 | Debu Panda | EJB3 in Action | 2007 | Introduction to Enterprise Java Beans |
| ▶ | 9 | Miguel de Cervantes | Don Quixote | 1605 | First modern novel |
| ▶ | 10 | Miguel de Cervantes | Don Quixote | 1605 | First modern novel |
| ▶ | 14 | Marin Preda | Cel mai iubit dintre pamanteni | 1980 | Romania after 1945 |
- Result Grid Panel:** Shows the current view is **Result Grid**.

IMPLEMENTARE SERVICIU REST

PUT BOOK UPDATE

```
@PUT  
@Path("{id}")  
@Consumes(MediaType.APPLICATION_JSON)  
@Produces(MediaType.APPLICATION_JSON)  
public Response modifyBook(@PathParam("id") long id, Book book){  
    ObjectMapper om = new ObjectMapper();  
    try {  
        SqlSessionFactory factory = Tools.getSqlSessionFactory();  
        try(SqlSession session = factory.openSession()){  
            book.setId(id);  
            session.update("updateBook", book);  
            session.commit();  
            ObjectNode node = om.createObjectNode();  
            node.put("message", "Book successfully updated.");  
            return Response.status(Status.CREATED).entity(node.toPrettyString()).build();  
        }  
    } catch (Exception e) {  
        logger.log(Level.SEVERE, e.toString());  
        ObjectNode node = om.createObjectNode();  
        node.put("message", "An error occurred. Check on server.");  
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(node.toString()).build();  
    }  
}
```



IMPLEMENTARE SERVICIU REST

PUT BOOK UPDATE

- Accesarea și trimiterea unei cărți modificate în reprezentare JSON, cu metoda **PUT** la endpoint-ul cu URL <http://localhost:8080/restful-service/webapi/book/14>, în vederea salvării modificărilor în baza de date. Returnează un mesaj în JSON cu succesul operațiunii. Coduri: **200(OK)**, **201(Created)**, **204 (No Content)**

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X PUT -H "Content-Type: application/json" -d "{\"author\":\"Marin Preda\",\"id\":10,\"published\":1955,\"remark\":\"Romania after 1945\",\"title\":\"Morometii\"}" http://localhost:8080/restful-service/webapi/book/14
HTTP/1.1 201
Content-Type: application/json
Content-Length: 48
Date: Thu, 29 Sep 2022 07:07:20 GMT

{
  "message" : "Book successfully updated."
}
D:\dev\tools\curl\bin>
```



IMPLEMENTARE SERVICIU REST

DELETE BOOK BY ID

```
@DELETE  
@Path("{id}")  
@Produces(MediaType.APPLICATION_JSON)  
public Response deleteBook(@PathParam("id") long id){  
    ObjectMapper om = new ObjectMapper();  
    try {  
        SqlSessionFactory factory = Tools.getSqlSessionFactory();  
        try(SqlSession session = factory.openSession()){  
            session.delete("deleteBookById", id);  
            session.commit();  
            ObjectNode node = om.createObjectNode();  
            node.put("message", "Book successfully deleted.");  
            return Response.status(Status.NO_CONTENT).entity(node.toPrettyString()).build();  
        }  
    } catch (Exception e) {  
        logger.log(Level.SEVERE, e.toString());  
        ObjectNode node = om.createObjectNode();  
        node.put("message", "An error occurred. Check on server.");  
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(node.toString()).build();  
    }  
}
```



IMPLEMENTARE SERVICIU REST

DELETE BOOK BY ID

- Accesarea cu metoda **DELETE** a endpoint-ului cu URL

<http://localhost:8080/restful-service/webapi/book/14> în vederea ștergerii înregistrării din baza de date. Returnează un mesaj în JSON cu succesul operațiunii. Coduri: **200(OK)**, **202 (Accepted)**, **204 (No Content)**

```
c:\ Command Prompt

D:\dev\tools\curl\bin>curl -i -X "DELETE" http://localhost:8080/restful-service/webapi/book/16
HTTP/1.1 200
Content-Type: application/json
Content-Length: 48
Date: Thu, 29 Sep 2022 07:26:17 GMT

{
  "message" : "Book successfully deleted."
}
D:\dev\tools\curl\bin>
```

CLIENT REST

- Specificația **JAX-RS** standardizează API client pentru o integrare coerentă cu serviciile web REST
- API client implementează constrângerile de interfață uniformă, care reprezintă o componentă cheie a stilului arhitectural REST, și oferă modalități convenabile pentru accesarea resurselor web
- **Punctul de plecare** în dezvoltarea unui client îl constituie clasa `javax.ws.rs.client.ClientBuilder` care creează o instanță a clasei `javax.ws.rs.client.Client`
- Clasa `javax.ws.rs.client.WebTarget` încapsulează URI și permite editarea acestuia cu metoda `path(String path)`
- Folosind clasa `javax.ws.rs.client.Invocation.Builder` și resursa creată se implementează o cerere folosind metoda `request()`
- Ultimul pas constă în invocarea și obținerea unui răspuns

CLIENT REST

```
package atm.paradigms;
import java.util.List;
import javax.ws.rs.client.*;
import javax.ws.rs.client.Invocation.Builder;
import javax.ws.rs.core.*;
import atm.paradigms.model.Book;

public class ClientTest {
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        String BASE_URI = "http://localhost:8080/restful-service/webapi";
        WebTarget webTarget = client.target(BASE_URI);
        // append book URI to BASE_URI
        WebTarget resource = webTarget.path("book");
        // build the request
        Builder builder = resource.request(MediaType.APPLICATION_JSON);
        // build a GET request
        Invocation invocation = builder.buildGet();
        // specify format type
        GenericType<List<Book>> responseType = new GenericType<List<Book>>(){ };
        List<Book> books = invocation.invoke(responseType);
        System.out.println(books);
    }
}
```

CLIENT REST

- Se poate folosi o **variantă simplificată API client** prin care se pot invoca direct metodele **head()**, **get()**, **post()**, **update()**, **put()**, sau **delete()** pe obiectul **Invocation.Builder**
- Vom implementa un client REST cu metode care să consume fiecare endpoint al serviciul REST BookService

The screenshot shows the Visual Studio Code interface with the following details:

- Project Structure:** The Explorer sidebar shows the project structure under "RESTFUL-CLIENT". It includes ".vscode", "src" (containing "main\java\atm\paradigms" with files Book.java, ClientTest.java, RestClient.java, and RestClientTest.java), "test", "target", and ".editorconfig". A file named "pom.xml" is selected in the Explorer.
- pom.xml Content:** The main editor window displays the following XML code:

```
35 | </dependency>
36 | <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.core/jersey-client/2.37 -->
37 | <dependency>
38 |   <groupId>org.glassfish.jersey.core</groupId>
39 |   <artifactId>jersey-client</artifactId>
40 |   <version>2.37</version>
41 | </dependency>
42 | <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.inject/jersey-hk2/2.37 -->
43 | <dependency>
44 |   <groupId>org.glassfish.jersey.inject</groupId>
45 |   <artifactId>jersey-hk2</artifactId>
46 |   <version>2.37</version>
47 | </dependency>
48 | <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/jersey-media-json-binding/2.37 -->
49 | <dependency>
```
- Annotations:** A green callout box highlights the pom.xml file in the Explorer and the Jersey dependencies in the code editor, with the text "Vezi proiectul course10/restful-client" and "Sunt prezentate dependențele necesare".

CLIENT REST

```
package atm.paradigms;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.core.GenericType;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import atm.paradigms.model.Book;

public class RestClient {
    private static final Logger logger = Logger.getLogger(RestClient.class.getName());
    private final String BASE_URI = "http://localhost:8080/restful-service/webapi";
    private Client client;

    public RestClient() {
        client = ClientBuilder.newClient();
    }
    ...
}
```

Clasa **RestClient** implementează metodele necesare pentru testarea fiecărui endpoint aparținând serviciului **BookService**



CLIENT REST GET ALL BOOKS

```
public List<Book> getAllBooks() {  
    String url = BASE_URI + "/book";  
    Response response = client.target(url).request()  
        .accept(MediaType.APPLICATION_JSON)  
        .get();  
    GenericType<List<Book>> responseType = new GenericType<List<Book>>() { };  
    return response.readEntity(responseType);  
}
```

Mod de utilizare API simplificat

- Se testează cu

```
// get all books  
List<Book> books = rc.getAllBooks();  
System.out.println(books);
```

- Rezultat:

```
[Book [author=Leo Tolstoy, id=1, published=1869, remark=Napoleonic wars, title=War and Peace],  
Book [author=Leo Tolstoy, id=2, published=1878, remark=Greatest book of love, title=Anna Karenina],  
...]
```

CLIENT REST GET BOOK BY ID

```
public Book getBookById(long id) {  
    String url = BASE_URI + "/book/" + id;  
    Response response = client.target(url).request()  
        .accept(MediaType.APPLICATION_JSON)  
        .get();  
    return response.readEntity(Book.class);  
}
```

- Se testează cu

```
//get book by Id  
Book book = rc.getBookById(1);  
System.out.println(book);
```

- Rezultat:

*Book [author=Leo Tolstoy, id=1, published=1869, remark=Napoleonic wars,
title=War and Peace]*

CLIENT REST POST NEW BOOK

```
public void addBook(Book book) {  
    String url = BASE_URI + "/book";  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .post(Entity.entity(book, MediaType.APPLICATION_JSON));  
    if (response.getStatus() == Status.CREATED.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully saved.");  
    }  
}
```

- Se testează cu

```
// add new book  
Book book = new Book("Marin Preda", "Cel mai iubit dintre pamanteni",  
                     1980, "Romania after 1945");  
rc.addBook(book);
```

- Rezultat:

Sep 30, 2022 10:12:54 AM atm.paradigms.RestClient addBook

INFO: Book successfully saved.

CLIENT REST

PUT BOOK UPDATE

```
public void updateBook(long id, Book book) {  
    String url = BASE_URI + "/book/" + id;  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .put(Entity.entity(book, MediaType.APPLICATION_JSON));  
    if (response.getStatus() == Status.CREATED.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully updated.");  
    }  
}
```

- Se testează cu

```
// update book  
Book book = new Book("Marin Preda", "Morometii",  
                     1955, "Romania before and after 1945");  
rc.updateBook(18, book);
```

- Rezultat:

Sep 30, 2022 10:15:47 AM atm.paradigms.RestClient updateBook

INFO: Book successfully updated.

CLIENT REST

DELETE BOOK BY ID

```
public void deleteBookById(long id) {  
    String url = BASE_URI + "/book/" + id;  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .delete();  
    if (response.getStatus() == Status.OK.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully deleted.");  
    }  
}
```

- Se testează cu

```
// delete one book by Id  
rc.deleteBookById(19);
```

- Rezultat:

Sep 30, 2022 10:18:25 AM atm.paradigms.RestClient deleteBookById

INFO: Book successfully deleted.

SECURIZAREA SERVICIILOR REST

- Există mai multe modalități de securizare a serviciilor REST
- *HTTP basic authentication*
- *HTTP digest authentication*
- *JWT authentication*
- *OAuth authentication*
- În acest curs vom discuta prima opțiune *HTTP basic authentication*
- Constă în trimiterea perechi utilizator/parolă, codificat cu Base64, într-un header de autorizare

Authorization: Basic ZGVtbzpwQDU1dzByZA==

- Header-ul de autorizare se trimită pentru fiecare cerere făcută de client
- Dacă cererea nu conține header-ul de autorizare serverul răspunde cu codul 401 **Unauthorized**



SECURIZAREA SERVICIILOR REST

- Serviciile REST fiind bazate pe protocolul de transport HTTP **sunt vulnerabile la riscuri de securitate** ca orice altă aplicație web
- Din perspectiva securizării serviciilor REST **contează securizarea accesului la servicii și accesarea acestora doar de utilizatorii autorizați**

Securitatea are 2 elemente esențiale:

- **Autentificarea** – care presupune **verificarea identității utilizatorului** prin obținerea datelor de logare (utilizator și parola) și validarea cu utilizatorii configurați pe server
- **Autorizarea** – care constă în **verificarea dacă utilizatorul autentificat are acces la resursa** pe care o accesează

SECURIZAREA SERVICIILOR REST

- Fiind **codificate cu Base64**, datele de logare pot fi **interceptate și decodificate** ușor dacă se folosește protocolul de transport HTTP
- Această vulnerabilitate **se rezolvă prin folosirea HTTPS** (TLS transport layer security) care **utilizează certificate digitale** pentru a securiza comunicația între client și server
- **Header-ul de autorizare se setează la client** indiferent de limbajul în care este dezvoltat (JavaScript, Java, C# etc.)
- Când un client accesează o resursă securizată, serverul identifică și validează utilizatorul și îi permite accesul la aplicație
- În timpul acestui proces **Jersey generează** obiectul **javax.ws.rs.core.SecurityContext** care conține **informații relevante de securitate** privind utilizatorul
- **SecurityContext** poate fi injectat cu anotarea **@Context**

```
public Response getSystemInfo(@Context SecurityContext securityContext) {  
... }
```

SECURIZAREA SERVICIILOR REST

- Pe server se implementează interfața `javax.ws.rs.container.ContainerRequestFilter` cu o singură metodă `filter` care interceptează cererile înainte de invocarea resurselor REST securizate
- Se implementează interfața `ContainerRequestFilter` pentru efectuarea autorizării
- Există 2 tipuri de filtre `ContainerRequestFilter`:
- **Postmatching** – filtrele sunt **executate după identificarea metodei clasei** resursă care corespunde URL-ului din cerere. Este varianta implicită
- **Prematching** - filtrele sunt **executate înainte de identificarea metodei clasei** resursă care corespunde URI –ului din cerere. Se specifică cu adnotarea `@PreMatching`

SECURIZAREA SERVICIILOR REST

FILTRU DE AUTORIZARE

```

@Provider
public class AuthenticationFilter implements ContainerRequestFilter {
    @Context
    private ResourceInfo resopuInfo;
    private static final String AUTHORIZATION_PROPERTY = "Authorization";
    private static final String AUTHENTICATION_SCHEME = "Basic";
    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        Method method = resopuInfo.getResourceMethod();
        // access not allowed for all
        if (!method.isAnnotationPresent(PermitAll.class)) {
            // access deny for all
            if (method.isAnnotationPresent(DenyAll.class)) {
                requestContext.abortWith(Response
                    .status(Response.Status.FORBIDDEN)
                    .entity("Access blocked for all users!!").build());
            }
            return;
        }
        // get request headers
        final MultivaluedMap<String, String> headers = requestContext.getHeaders();
        // get authorization header
        final List<String> authorization = headers.get(AUTHORIZATION_PROPERTY);
        ...
    }
}

```

Se verifică dacă adnotarea de securitate pe endpoint este alta decât **@PermitAll**

Dacă adnotarea de securitate este **@DenyAll** trimite un mesaj JSON clientului cu cod 403 FORBIDDEN

Obține header-ul de autorizare trimis de cererea client

SECURIZAREA SERVICIILOR REST

FILTRU DE AUTORIZARE

```
...  
    // block access if no authorization  
    if (authorization == null || authorization.isEmpty()) {  
        requestContext.abortWith(Response  
            .status(Response.Status.FORBIDDEN)  
            .entity("You cannot access this resource!!").build());  
        return;  
    }  
    // get encoded credentials  
    final String encodedCredentials = authorization.get(0)  
        .replaceFirst(AUTHENTICATION_SCHEME + " ", "");  
    // decode credentials  
    final String credentials = new String(Base64.getDecoder().decode(encodedCredentials));  
    // split username and password  
    final StringTokenizer tokenizer = new StringTokenizer(credentials, ":");  
    final String username = tokenizer.nextToken();  
    final String password = tokenizer.nextToken();  
...
```

Dacă header-ul de autorizare lipsește sau este gol, trimite un mesaj JSON cu cod de stare 403 FORBIDDEN

Se extrag datele de logare din header-ul de autorizare



SECURIZAREA SERVICIILOR REST

```

    ...
    // verify user access
    if (method.isAnnotationPresent(RolesAllowed.class)) {
        RolesAllowed rolesAnnotation = method.getAnnotation(RolesAllowed.class);
        Set<String> rolesSet = new HashSet<>(Arrays.asList(rolesAnnotation.value()));
        // user not valid
        if (!isUserAllowed(username, password, rolesSet)) {
            requestContext.abortWith(Response
                .status(Response.Status.FORBIDDEN)
                .entity("You cannot access this resource!!").build());
        }
        return;
    }
}

private boolean isUserAllowed(final String username, final String password, final Set<String> rolesSet) {
    boolean isAllowed = false;
    if (username.endsWith("user") && password.equals("password")) {
        // mock role
        String role = "ADMIN";
        if (rolesSet.contains(role)) {
            isAllowed = true;
        }
    }
    return isAllowed;
}

```

Se verifică dacă adnotarea de securitate pe endpoint este `@RolesAllowed` și dacă este se extrag rilurile din adnotare

Metoda `isUserAllowed()` implementează logica de autorizare. Se trimit 403 FORBIDDEN dacă datele de logare nu corespund

Metoda `isUserAllowed()` este de test. Așteaptă valorile doar `user`, `password` și rolul **ADMIN**.

SECURIZAREA SERVICIILOR REST

ÎNREGISTRAREA FILTRULUI

- Pentru a **înregistra filtrul de autorizare** creat **cu containerul** se creează clasa **CustomApplication** care extinde **org.glassfish.jersey.server.ResourceConfig**

```
package atm.paradigms;

import org.glassfish.jersey.server.ResourceConfig;
import atm.paradigms.provider.AuthenticationFilter;

public class CustomApplication extends ResourceConfig {

    public CustomApplication() {
        packages("atm.paradigms");
        register(AuthenticationFilter.class);
    }
}
```



SECURIZAREA SERVICIILOR REST

ÎNREGISTRAREA FILTRULUI

- Este necesar să se modifice fișierul **web.xml** care este **descriptorul pentru instalarea aplicației web**. Se găsește în proiect la **/webapp/WEB-INF/web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>restful-service</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>javax.ws.rs.Application</param-name>
            <param-value>atm.paradigms.CustomApplication</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>restful-service</servlet-name>
        <url-pattern>/webapi/*</url-pattern>
    </servlet-mapping>
</web-app>
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- JAX-RS permite controlul accesului folosind adnotările *javax.annotation.security* pe clasa resursă sau metodele clasei resursă:
- *DenyAll* – nici un rol nu poate accesa resursa
- *PermitAll* – toate rolurile de securitate au permisiunea de a accesa resursa
- *RolesAllowed* – specifică lista de roluri care au permisiunea de a accesa resursa
- Vom aplica adnotări de securitate pe toate endpoint-urile aparținând serviciului REST *BooksService*

```
@Path("book")
public class BooksService {
    @PermitAll
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getAllBooks(){
        . . .
    }

    @PermitAll
    @GET
    @Path("{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getABooksById(@PathParam("id") long id){
        . . .
    }

    @RolesAllowed("ADMIN")
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response addBook(Book book){
        . . .
    }
    . . .
}
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

```
    . . .
    @RolesAllowed("ADMIN")
    @PUT
    @Path("{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response modifyBook(@PathParam("id") long id, Book book){
        . . .
    }

    @RolesAllowed("ADMIN")
    @DELETE
    @Path("{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response deleteBook(@PathParam("id") long id){
        . . .
    }
}
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- Testăm cu **curl**, fără autentificare, resursa **book** cu URL

<http://localhost:8080/restful-service/webapi/book>

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/restful-service/webapi/book
HTTP/1.1 200
Content-Type: application/json
Content-Length: 777
Date: Mon, 03 Oct 2022 11:06:23 GMT

[{"author": "Leo Tolstoy", "id": 1, "published": 1869, "remark": "Napoleonic wars", "title": "War and Peace"}, {"author": "Leo Tolstoy", "id": 2, "published": 1878, "remark": "Greatest book of love", "title": "Anna Karenina"}, {"author": "Jeff Prosise", "id": 3, "published": 1999, "remark": "Classic book about MFC", "title": "Programming Windows with MFC"}, {"author": "Lev Tolstoi", "id": 4, "published": 2005, "remark": "JBoss practical guide", "title": "JBoss at Work"}, {"author": "Debu Panda", "id": 5, "published": 2007, "remark": "Introduction to Enterprise Java Beans", "title": "EJB3 in Action"}, {"author": "Miguel de Cervantes", "id": 9, "published": 1605, "remark": "First modern novel", "title": "Don Quixote"}, {"author": "Miguel de Cervantes", "id": 10, "published": 1605, "remark": "First modern novel", "title": "Don Quixote"}]
D:\dev\tools\curl\bin>
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- Testăm cu **curl**, fără autentificare, resursa **book/{id}** cu URL
<http://localhost:8080/restful-service/webapi/book/1>

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/restful-service/webapi/book/1
HTTP/1.1 200
Content-Type: application/json
Content-Length: 99
Date: Tue, 04 Oct 2022 06:51:44 GMT

{"author": "Leo Tolstoy", "id": 1, "published": 1869, "remark": "Napoleonic wars", "title": "War and Peace"}
D:\dev\tools\curl\bin>
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- Același lucru **extensia VSC Rest Client**:

The screenshot shows the 'TestRest.rest' file in VS Code. It contains a single line of code: 'GET http://localhost:8080/restful-service/webapi/book/1'. Below the code, the 'Response(10ms)' tab is selected, displaying the following JSON response:

```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Content-Length: 99
4 Date: Tue, 04 Oct 2022 06:52:39 GMT
5 Connection: close
6
7 {
8     "author": "Leo Tolstoy",
9     "id": 1,
10    "published": 1869,
11    "remark": "Napoleonic wars",
12    "title": "War and Peace"
13 }
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- Testăm cu **curl**, fără autentificare, resursa **book** cu URL

<http://localhost:8080/restful-service/webapi/book> folosind metoda **POST**
pentru a adăuga o carte în format JSON

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X "POST" -H "Content-Type: application/json" -d "{\"author\":\"Marin Preda\",\"id\":10,\"published\":1980,\"remark\":\"Romania after 1945\",\"title\":\"Cel mai iubit dintre pamanteni\"}" http://localhost:8080/restful-service/webapi/book
HTTP/1.1 403
Content-Type: application/json
Content-Length: 33
Date: Tue, 04 Oct 2022 06:59:16 GMT

You cannot access this resource!!
D:\dev\tools\curl\bin>
```

SECURIZAREA SERVICIILOR REST

CONTROLUL ACCESULUI

- Testăm cu **curl**, cu autentificare, resursa **book** cu URL

<http://localhost:8080/restful-service/webapi/book> folosind metoda **POST**
pentru a adăuga o carte în format JSON

```
D:\dev\tools\curl\bin>curl -i -X "POST" -u "user:password" -H "Content-Type: application/json" -d "{\"author\":\"Marin Preda\",\"id\":10,\"published\":1980,\"remark\":\"Romania after 1945\",\"title\":\"Cel mai iubit dintre pamanteni\"}" http://localhost:8080/restful-service/webapi/book
HTTP/1.1 201
Content-Type: application/json
Content-Length: 49
Date: Tue, 04 Oct 2022 07:01:03 GMT

{
  "message" : "Book successfully inserted."
}
D:\dev\tools\curl\bin>
```



SECURIZAREA SERVICIILOR REST

CLIENT REST

- **Modificările clasei RestClient pentru a trimite header-ul Authorization**
- Metodele `getAllBooks()` și `getBookById(long id)` nu se modifică deoarece endpoint-ul accesat are anotarea `@PermitAll`
- **Se adaugă un constructor care primește ca parametrii numele utilizatorului și parola**

```
public class RestClient {  
    private static final Logger logger = Logger.getLogger(RestClient.class.getName());  
    private final String BASE_URI = "http://localhost:8080/restful-service/webapi";  
    private Client client;  
    private String authEnc;  
    ...  
    public RestClient(String username, String password) {  
        client = ClientBuilder.newClient();  
        String credentials = username + ":" + password;  
        authEnc = new String(Base64.getEncoder().encode(credentials.getBytes()));  
    }  
    ...  
}
```

SECURIZAREA SERVICIILOR REST

CLIENT REST

```
public void addBook(Book book) {  
    String url = BASE_URI + "/book";  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .header("Authorization", "Basic " + authEnc)  
        .post(Entity.entity(book, MediaType.APPLICATION_JSON));  
    if (response.getStatus() == Status.CREATED.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully saved.");  
    }  
}
```

O variantă o reprezintă folosirea metodei **header()** cu parametrii corespunzători.

```
RestClient rc = new RestClient("user", "password");  
// add new book  
Book book = new Book("Marin Preda", "Cel mai iubit dintre pamanteni",  
                     1980, "Romania after 1945");  
rc.addBook(book);
```

Rezultat:

Oct 04, 2022 10:27:21 AM atm.paradigms.RestClient addBook
INFO: Book successfully saved.

SECURIZAREA SERVICIILOR REST

CLIENT REST

```
public void updateBook(long id, Book book) {  
    String url = BASE_URI + "/book/" + id;  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .header("Authorization", "Basic " + authEnc)  
        .put(Entity.entity(book, MediaType.APPLICATION_JSON));  
    if (response.getStatus() == Status.CREATED.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully updated.");  
    }  
}
```

```
RestClient rc = new RestClient("user", "password");  
// update book  
Book book = new Book("Marin Preda", "Morometii",  
                     1955, "Romania before and after 1945");  
rc.updateBook(22, book);
```

Rezultat:

Oct 04, 2022 10:30:06 AM atm.paradigms.RestClient updateBook
INFO: Book successfully updated.

SECURIZAREA SERVICIILOR REST

CLIENT REST

```
public void deleteeBookById(long id) {  
    String url = BASE_URI + "/book/" + id;  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        .header("Authorization", "Basic " + authEnc)  
        .delete();  
    if (response.getStatus() == Status.OK.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully deleted.");  
    }  
}
```

```
RestClient rc = new RestClient("user", "password");  
// delete one book by Id  
rc.deleteeBookById(23);
```

Rezultat:

Oct 04, 2022 10:33:03 AM atm.paradigms.RestClient deleteeBookById
INFO: Book successfully deleted.

SECURIZAREA SERVICIILOR REST

CLIENT REST

- **Utilizarea clasei *HttpAuthenticationFeature* care furnizează capabilități de autentificare BASIC și DIGEST:**
- **BASIC** – mod preventiv de autentificare. Trimit datele de autentificare cu fiecare cerere HTTP
- **BASIC NON-PREEMPTIVE** – trimit datele de autentificare doar dacă serverul răspunde cu codul 401
- Se modifică constructorul `RestClient(String username, String password)` prin **adăugarea unei instanțe *HttpAuthenticationFeature* și înregistrarea acestuia cu clientul**
- Metodele pentru **POST, PUT și DELETE nu mai au nevoie de header-ul de autorizare**

SECURIZAREA SERVICIILOR REST

CLIENT REST

```
public RestClient(String username, String password) {  
    client = ClientBuilder.newClient();  
    // String credentials = username + ":" + password;  
    // authEnc = new String(Base64.getEncoder().encode(credentials.getBytes()));  
    HttpAuthenticationFeature feature = HttpAuthenticationFeature.basic(username, password);  
    client.register(feature);  
}  
...  
public void addBook(Book book) {  
    String url = BASE_URI + "/book";  
    Response response = client.target(url).request(MediaType.APPLICATION_JSON)  
        .accept(MediaType.APPLICATION_JSON)  
        // .header("Authorization", "Basic " + authEnc)  
        .post(Entity.entity(book, MediaType.APPLICATION_JSON));  
    if (response.getStatus() == Status.CREATED.getStatusCode()) {  
        logger.log(Level.INFO, "Book successfully saved.");  
    }  
}  
...
```

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