Laborator 11: Servicii Rest (Representational State Transfer)

Rest (Representational State Transfer) este un model arhitectural pentru crearea de web-servicii, care cuprinde o serie de recomandări și constângeri pentru transferul informațiilor via HTTP.

Formatul datelor transmise poate fi xml, json (JavaScriptObject Notation), html etc.
MIME content type = Multipurpose Internet Mail Extension: text/html, text/plain, application etc.

Caracteristici arhitectură REST (constrângeri):
1. **arhitectură client-server**: Aplicația server și aplicațiile client sunt separate și independente și interacționează doar prin mesaje *request* sau *response*.

Mesajele de tip response sunt însoțite de un status code care se regăsesc într-unul din următoarele intervale:
100 – 199 info
200 – 299 succes
300 – 399 redirectare
400 – 499 eroare client
500 – 599 eroare server

2. **statelessness:** totate informațiile necesare pentru procesarea unui *request* sunt trimise în request, nu sunt memorate sesiuni la nivel de server.

3. **resurse cacheable:** resursele cacheable vor fi însoțite de informații privind versiunea, clientul va fi infomat dacă o resursă este sau nu *cacheable*.

4. **code on demand:** (opțional) mesajele respone pot conține cod executabil (JavaScript)

5. **layered system:** pot exista servere intermediare, acest fapt nu va modifica mesajele *request* sau *response*.

6. **interfață uniformă:** Fiecare resursă este identificată unic de un singur URI.

Operații disponibile într-un API Rest: **GET** (SELECT), **POST** (INSERT), **PUT** (UPDATE), **DELETE**.

JSON JavaScriptObject Notation:
Obietele json conțin perechi **key/value** separate prin „**:**”
Identificatorii *Key* sunt incluși obligatoriu între ghilimele.
Valorile pot fi numere, șiruri de caractere, valori booleene, vectori (*array*) incluși între paranteze drepte, obiecte json imbricate și **null.**

Jackson este librăria folosită de spring pentru pentru serializarea, deserializarea obietelor JSON (maparea obiectelor JSON în POJO (Plain Old Java Objects)). Orice obiect JSON trimis unui controller Rest este automat convertit in POJO.

Pentru a utiliza **Jackson** în proiecte care nu utilizează spring se adaugă în pom:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.0</version>
</dependency>
```

Obiectele de tip ObjectMapper pot citi obiecte de tip json (metoda readValue) dintr-un fisier, string etc (sursa este specificată în primul argument al metodei readValue) si le pot converti, utilizând metode de tip setter, într-un obiect POJO (tipul POJO este specificat în al doilea argument al metodei readValue).

1. Importați proiectul laborator_11.

2. Adăugați testul MapJson care va converti un string de tip Json, denumit jsonObject într-un obiect de tip Film:

```
public static final String jsonObject =
"{\"titlu\":\"titluTest\"," +
            "\"cast\":[{\"rol\":\"rolTest\"}]}";

@Test
public void MapJson() throws IOException {

        ObjectMapper objectMapper = new ObjectMapper();
        log.info(jsonObject);
        Film result = objectMapper.readValue(jsonObject,
Film.class);
        log.info(result.toString());

        String jsonFilm = "";
        objectMapper.enable(SerializationFeature.INDENT_OUTPUT);
        jsonFilm = objectMapper.writeValueAsString(result);
        log.info(jsonFilm);
}
```

com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException Dacă obiectul JSON conține atribute care nu sunt mapate în obiectul POJO, pentru care nu sunt implementate metode de tip setter, se aruncă excepția com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException. Dacă se dorește ignorarea atributelor pentru care nu sunt implementate metode de tip *setter* se va folosi adnotarea @JsonIgnoreProperties.

Exemple:
@JsonIgnoreProperties(values=”regizor”). Campul regizor nu va fi serializare/deserializat
@JsonIgnoreProperties(ignoreUnknown = true). Nu vor fi serializate câmpurile necunoscute.
@JsonIgnoreProperties(value = "regizor", allowGetters = true) campurile specificate la *value* vor fi de tip read only.

3. Adăugați în obiectul JSON *jsonObject* atributul regizor. Rulați testele. Adnotați clasa Film @JsonIgnoreProperties(ignoreUnknown = true)

```
public static final String jsonObject =
"{\"titlu\":\"titluTest\"," +
            "\"cast\":[{\"rol\":\"rolTest\"}]," +
            "\"regizor\":\"regTest\"}";
```

```
@JsonIgnoreProperties(ignoreUnknown = true)
@Entity
public class Film {
```

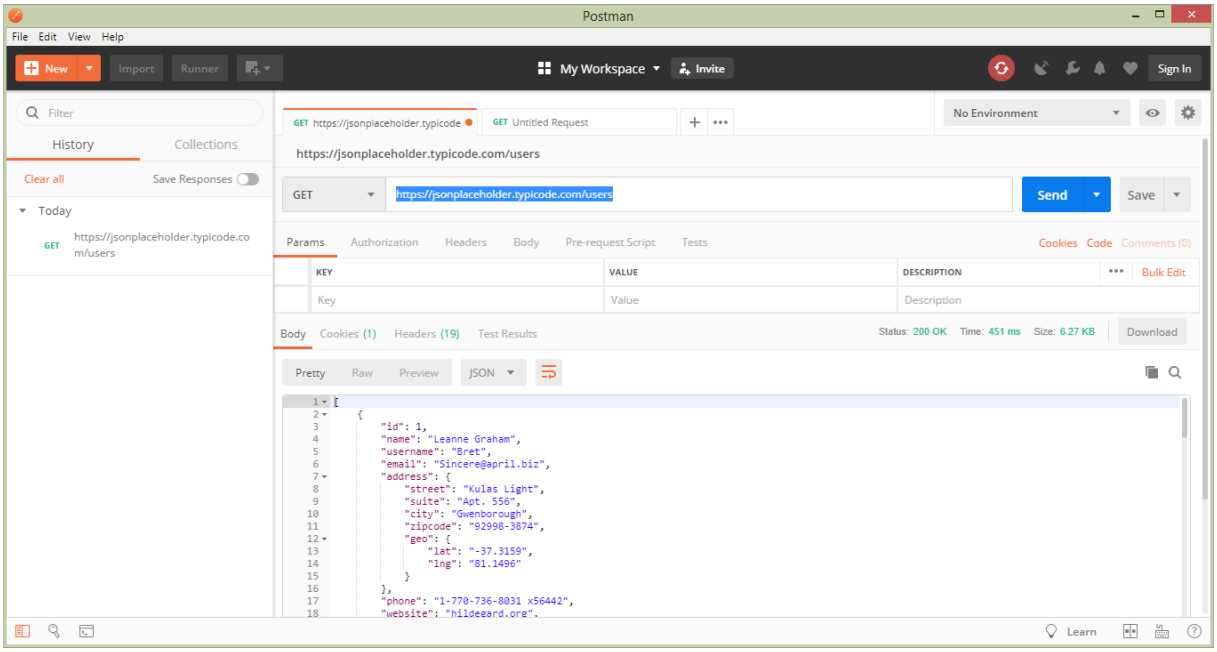
4. Adnotați entitatea Film astfel încât să nu fie serializate atributele null

```
@JsonInclude(Include.NON_NULL)
```

5. Schimbați ordinea de afișare a atributelor cast și categorii:

```
@JsonPropertyOrder
```

6. În Postman testați request-ul https://jsonplaceholder.typicode.com/users.



7. Adăugați în TestRestController metoda:

```
@GetMapping("filme")
public List<Film> getFilme(){
      List<Film> filme  = new LinkedList<Film>();
      filme.addAll(filmService.getFilme());
      return filme;
}

return
restTemplate.getForObject("https://jsonplaceholder.typicode.com/t
odos/1", TestJson.class);
```

8. Creați o clasa de tip @RestController. Testați în Postman http://localhost:8080/api/v1/hello

```
@RestController
@RequestMapping("/api/v1")
public class TestRestController {
    @GetMapping("hello")
    public String helloWord(){
        return "Hello Word!";
    }
}
```

9. Testați:

```
http://localhost:8080/api/v1/hello
```

10. Adăugați adnotările@JsonManagedReference și @JsonBackReference necesare.

```
@JsonBackReference
private Set<Film> filme = new HashSet<Film>();

@JsonManagedReference
private Info info;
```

11. Adăugați în header-ul atașat obiectului response perechea Example-Header = Value-Header.

```
@GetMapping("/film/{id}")
    public ResponseEntity<Film> getFilmById(@PathVariable String
id){
        HttpHeaders responseHeaders = new HttpHeaders();
        responseHeaders.set("Example-Header", "Value-Header");
        ResponseEntity<Film> response = new
ResponseEntity<Film>(HttpStatus.OK);
        return
response.ok().headers(responseHeaders).body(filmService.findById(
new Long(id)));
    }
```

12. Adăugați o metodă pentru crearea unui nou rol și testați în Postman:

```
@PostMapping("cast")
public ResponseEntity<Cast> saveCast(@RequestBody Cast cast){
        return new
ResponseEntity<Cast>(castService.saveCast(cast),
                HttpStatus.CREATED);
}
```

```
{
"rol":"rol test 2",
"film":{
    "id": 3,
    "titlu": "Aquaman"
    },
 "infoActor":{"id":1}
}
```

13. În pachetul config creați un template Rest:

```
@Configuration
public class RestTemplateConfig {
    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder
builder){
        return builder.build();
    }
}
```

14. În **MySqlWorkbench** deschideți o conexiune pentru user root și executați:

```
CREATE DATABASE filme;

CREATE USER 'user_filme'@'localhost' IDENTIFIED BY 'user_filme';

GRANT SELECT ON filme.* TO 'user_filme'@'localhost';
GRANT INSERT ON filme.* TO 'user_filme'@'localhost';
GRANT DELETE ON filme.* TO 'user_filme'@'localhost';
GRANT UPDATE ON filme.* TO 'user_filme'@'localhost';
GRANT CREATE ON filme.* TO 'user_filme'@'localhost';
GRANT ALTER  ON filme.* TO 'user_filme'@'localhost';
GRANT DROP   ON filme.* TO 'user_filme'@'localhost';
```

15. Adaugați în fișierul pom al proiectului dependența:

```
<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
</dependency>
```

16. În directorul resources adaugați fișierul: **application-mysql.properties**:

```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:mysql://localhost:3306/filme
spring.datasource.username=user_filme
spring.datasource.password=user_filme
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.hbm2ddl.import_files=data.sql
```

17. Adaugați fișierul **application-h2.properties**

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.hibernate.ddl-auto=create-drop
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.show-sql=true
```

18. Testați aplicația:

```
spring-boot:run -Dspring-boot.run.profiles=mysql
spring-boot:run -Dspring-boot.run.profiles=h2
```