2. REST homework

Dr. Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2017

In the following text use your own Neptun code with capital letters instead of the word “NEPTUN”.

1 Description of the task

The task is to create a system for storing information about movies and the publication of the system as a RESTful web service.

The data structure received or sent by the service is described by the following pseudo-code:

```
struct Movie
{
    string Title;
    int Year;
    string Director;
    string[] Actor; // list of actor names
}
```

2 The service

The service should be implemented in a Maven web application similar to the one in the tutorial. The pom.xml must be the attached **service/pom.xml**, but in this file the word NEPTUN should be replaced with your own Neptun code.

The name of the web application should be: **Rest_NEPTUN**

The service has to store data between calls. In a real world application the storage should be a database. In the homework, the storage should be static variables.

The base URL of the service must be:

`http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase`

The table below describes the operations of the service. The URLs inside the table are relative to the base URL given above.

HTTP method	Relative URL	Parameters	Operation
GET	**/movies**	(none)	returns the list of all Movie objects
GET	**/movies/{id}**	the **{id}** is an identifier of a Movie	returns the Movie object with the given identifier if no such object exists, it returns with HTTP 404 (not found) status code
POST	**/movies**	a Movie structure inside the HTTP body	inserts the received Movie object into the database (the server assigns and identifier to the Movie object) returns the identifier of the Movie object
PUT	**/movies/{id}**	the **{id}** is an identifier of a Movie, a Movie structure inside the HTTP body	inserts or updates the received Movie object (the client assigns the identifier to the Movie object) if no such Movie with the received identifier exists, it inserts the Movie to the database, otherwise it updates the Movie in the database
DELETE	**/movies/{id}**	the **{id}** is an identifier of a Movie	deletes the Movie with the given identifier from the database
GET	**/movies/find?year={year}&orderby={field}**	the **{year}** is a year, the value of **{field}** can be **Title** or **Director**	returns the list of identifiers of all the Movie objects which have the year number **{year}** the list should be sorted according to **{field}** (**Title** means order by title, **Director** means order by director)

The service must support requests and responses in XML and JSON format. The examples below give the exact format of the requests and responses.

Example for creating a new movie:

POST	**http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies**

Request HTTP body:	{
	"title": "Batman Begins",
	"year": 2005,
	"director": "Cristopher Nolan"
	}

Response HTTP body:	{"id":61}

Example for creating a new movie:

POST	**http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies**
Request HTTP body:	<movie> <title>Batman Begins</title> <year>2005</year> <director>Cristopher Nolan</director> </movie>
Response HTTP body:	<result> <id>61</id> </result>

Example for updating a movie:

PUT	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/61

Request HTTP body:	{
	"title": "Batman Begins",
	"year": 2005,
	"director": "Cristopher Nolan",
	"actor": ["Christian Bale", "Michael Caine"]
	}

Response HTTP body:	

Example for updating a movie:

PUT	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/61
Request HTTP body:	<movie> <title>Batman Begins</title> <year>2005</year> <director>Cristopher Nolan</director> <actor>Christian Bale</actor> <actor>Michael Caine</actor> </movie>
Response HTTP body:	

Example for deleting a movie:

DELETE	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/61
Request HTTP body:	
Response HTTP body:	

Example for retrieving a movie:

GET	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/61
Request HTTP body:	

Response HTTP body:	{
	"title": "Batman Begins",
	"year": 2005,
	"director": "Cristopher Nolan",
	"actor": ["Christian Bale", "Michael Caine"]
	}

Example for retrieving a movie:

GET	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/61
Request HTTP body:	
Response HTTP body:	<movie> <title>Batman Begins</title> <year>2005</year> <director>Cristopher Nolan</director> <actor>Christian Bale</actor> <actor>Michael Caine</actor> </movie>

Example for retrieving all the movies:

GET	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies
Request HTTP body:	

Response HTTP body:	{
	"movie": [
	{
	"title": "Batman Begins",
	"year": 2005,
	"director": "Cristopher Nolan",
	"actor": ["Christian Bale", "Michael Caine"]
	},
	{
	"title": "Pirates of the Caribbean: The Curse of the Black Pearl",
	"year": 2003,
	"director": "Gore Verbinski",
	"actor": ["Johnny Depp", "Geoffrey Rush"]
	}
	]
	}

Example for retrieving all the movies:

GET	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies
Request HTTP body:	
Response HTTP body:	<movies> <movie> <title>Batman Begins</title> <year>2005</year> <director>Cristopher Nolan</director> <actor>Christian Bale</actor> <actor>Michael Caine</actor> </movie> <movie> <title>Pirates of the Caribbean: The Curse of the Black Pearl</title> <year>2003</year> <director>Gore Verbinski</director> <actor>Johnny Depp</actor> <actor>Geoffrey Rush</actor> </movie> </movies>

Example for finding movies:

GET	http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/find?year=2007&orderby=Director
Request HTTP body:	
Response HTTP body:	{"id":[56,41]}

Example for finding movies:

GET	**http://localhost:8080/Rest_NEPTUN/resources/MovieDatabase/movies/find?year=2003&orderby=Title**
Request HTTP body:	
Response HTTP body:	<movies> <id>23</id> <id>37</id> </movies>

The server does not have to respond with XML to an XML request or with JSON to a JSON request. The server must respond to the client in the format the client asks through the HTTP Accept header.

3 The client

No client is required for this homework. However, creating a client for testing is strongly recommended, but do not upload this client.

4 Hints for the solution

To achieve the exact XML and JSON formats, the **name** attribute of the **@XmlElement** and **@XmlRootElement** annotations should be fine-tuned.

Sometimes it may be necessary to create wrapper classes to achieve the desired XML and JSON structure. One of the challenges of this task is finding the appropriate wrapper classes.

5 To be uploaded

A single ZIP file has to be uploaded. Other archive formats must not be used!

The root of the ZIP file must contain a single folder:

- **Rest_NEPTUN:** the Maven application of the service

The compiled class files (target folder) may be omitted from the ZIP file.

The service application must compile from the command line using the following command:

```
...\Rest_NEPTUN>mvn package
```

The service application must deploy to the server from the command line using the following command:

```
...\Rest_NEPTUN>mvn wildfly:deploy
```

The service must support the exact XML and JSON message formats described in the examples above.

The solution must compile and run in the given environment using the given pom.xml files. These pom.xml files must not be modified apart from replacing the word NEPTUN with your own Neptun code.

Important: the instructions and naming constraints described in this document must be precisely followed!

Again: use your own Neptun code with capital letters instead of the word “NEPTUN”.