University of Cyprus
Computer Science Department

Homework 2: RESTful API for serving a Leave Management System

EPL425: Internet Technologies
Lab instructor: Pavlos Antoniou
Spring 2024

Announced Date: Friday, 05/04/2024
Submission Date: Friday, 19/04/2024 (**23:59**)

1. Introduction

The goal of this exercise is to develop a RESTful API to serve a Leave Management System (LMS) for an organization. An LMS is the process within an organization that determines how leave is requested by employees and approved by managers, as well as how it is tracked for payroll, balance, and other purposes. A modern LMS should be digitalized, automated, and cloud-based.

You are about to develop a RESTful API to enable a full set of CRUD (**C**reate, **R**etrieve, **U**pdate, **D**elete) operations on the entities involved in the LMS.

2. Description

The REST API will provide access into 2 roles, 3 (in memory) users, and will manage 2 entities.

Roles: EMPLOYEE, MANAGER

MANAGER role will be able to:

- Create employee
- Retrieve employee information
- Update employee information
- Delete employee
- Create leave
- Retrieve leave information
- Update leave information
- Delete leave

EMPLOYEE role will be able to:

- Retrieve employee information

- Create leave
- Retrieve leave information
- Update leave information

In-memory Users:

- Username: jsmith, password: epl42$, role: EMPLOYEE
- Username: atrevor, password: letmein, role: EMPLOYEE, MANAGER
- Username: dalves, password: secure, role: MANAGER

Entities: employee, leave

Employee entity consists of the following attributes:

- id (int, primary key)
- firstname (varchar, not null)
- lastname (varchar, not null)
- department (varchar, not null)
- date_of_birth (date, not null)

Leave entity consists of the following attributes:

- id (int, primary key)
- employee_id (int, foreign key)
- description (text, not null)
- start_date (date, not null)
- end_date (date, not null)
- approved (tinyint(1), not null)

3. Spring Boot project

Use the information provided below to create a new Spring Boot project using Spring Initializr:

- Project: Maven
- Language: Java
- Spring Boot: The latest version (not a snapshot repository, snapshot means that this version has not been released yet) – leave selected
- Project Metadata
  - Group: cy.ac.ucy.cs.epl425
  - Artifact: LMS
  - Name: LMS
  - Description: Leave Management System
  - Package name: cy.ac.ucy.cs.epl425.LMS
- Packaging: Jar
- Java (version): leave selected

- Dependencies:
    - Spring Web
    - Spring Data JDBC
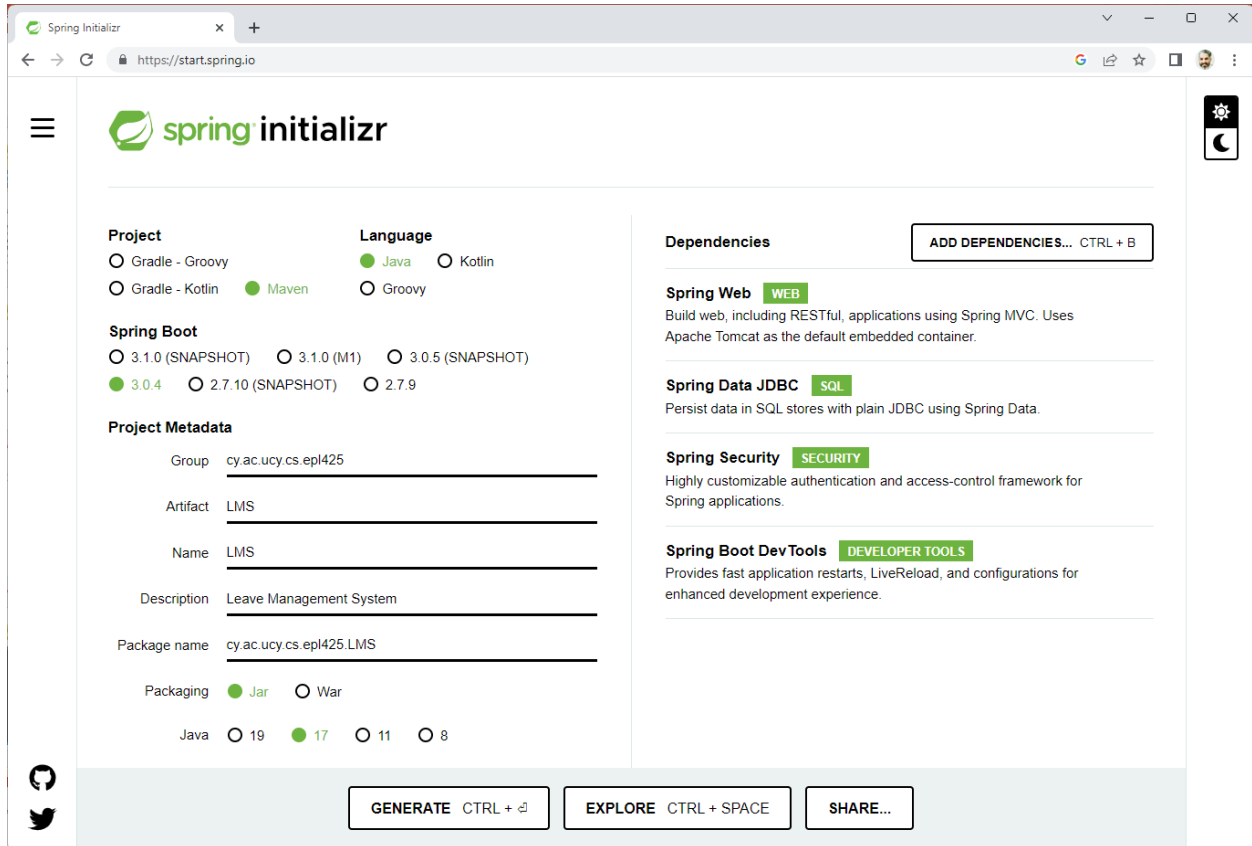   - Spring Security
    - Spring Boot Dev Tools

Figure 1: Recommended Spring Initialiser project settings.

4. Database

You need to launch a database server to store all employees and leaves. You can use the MySQL server that comes with XAMPP (see figure below). In addition, you need to launch Apache Web server in order to enable the phpMyAdmin dashboard (http://localhost/phpmyadmin/).
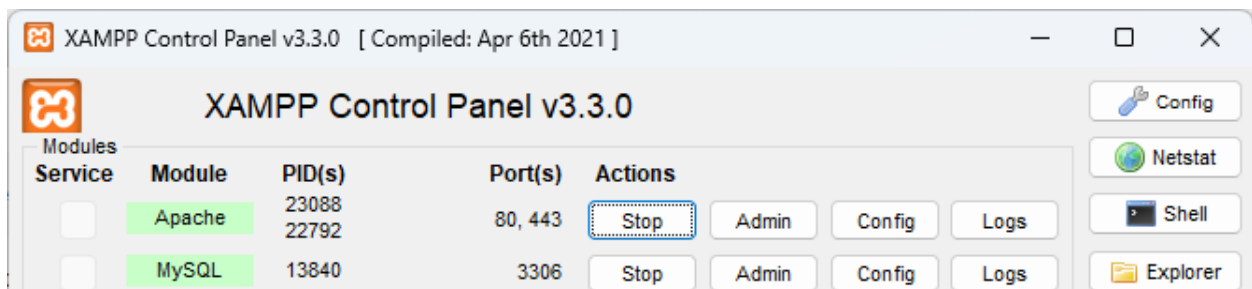
Before implementing your API, you need to create the database and the tables to store your data. Navigate via browser to phpMyAdmin to gain access to MySQL database server and create a database, namely **lms with two tables, employees and leaves**[1]. The following screenshots were taken from phpMyAdmin and display the structure of each of the aforementioned tables. <u>You must create the same database name, table names along with their attributes on your machine.</u>

Figure 2: employees table structure.

Figure 3: leaves table structure.

---

[1] The last slides of Lab 9 can guide you on how to use phpMyAdmin dashboard to create the given database and tables as well as define the structure of each table.

5. API Endpoints

The table shown below, displays all API endpoints concerning the Employee entity.

Method	API Endpoint (URL)	Description
GET	/api/employees	retrieve a list of all Employees (*)
GET	/api/employees/:id	retrieve an Employee by :id
POST	/api/employees	create new Employee
PUT	/api/employees/:id	update an Employee by :id
DELETE	/api/employees	delete all Employees
DELETE	/api/ employees /:id	delete an Employee by :id

(*) The first endpoint can accept the following <u>optional</u> request parameter:

Name	Type	Description
department	String	Retrieves a list of all Employees belonging to a specific department (all departments **containing** the given string will be taken in account). Example: /api/employees?department=it

The table shown below, displays all API endpoints concerning the Leave entity.

Method	API Endpoint (URL)	Description
GET	/api/leaves	retrieve a list of all Leaves (**)
GET	/api/leaves/:id	retrieve a Leave by :id
POST	/api/leaves/employees/:eid	create new Leave for the Employee by :eid
PUT	/api/leaves/:id	update the Leave by :id
DELETE	/api/leaves	delete all Leaves
DELETE	/api/leaves/:id	delete the Leave by :id

(**) The first endpoint can accept the following <u>optional</u> request parameters:

Name	Type	Description
start_date	date (ISO 8061)	YYYY-MM-DD (ISO 8601/RFC 3339). The oldest date from which the Leaves will be provided. Date is in day granularity and is inclusive (for example, 2023-03-01 includes the first day of March 2023). If not used with end_date Leaves from start_date to today will be returned.
end_date	date (ISO 8601)	YYYY-MM-DD (ISO 8601/RFC 3339). The newest, most recent date to which the Leaves will be provided. Date is in day granularity and is inclusive (for example, 2023-03-10 includes the 10th day of March 2023). If not used with start_date, all Leaves to the end_date will be returned.
approved	Boolean	Indicates if the approved or not approved Leaves will be returned. Default value (if not approved is used) is null. If not set, all leaves are returned.

GET messages will return in JSON format all attributes of each employee/leave as stored in the corresponding table. POST and PUT messages will accept a JSON string with all employee/leave attributes.

Response Codes

GET responses:

- If no employees/leaves are found in database, GET returns 204 NO CONTENT.
- On success, GET returns 200 OK
- When the requested employee/leave is not found (when retrieving by id), GET returns 404 NOT FOUND
- On failure, GET returns 500 SERVER ERROR

POST responses

- On success, POST returns 201 CREATED
- On failure, POST returns 500 SERVER ERROR

PUT responses

- On success, PUT returns 200 OK

- When the requested employee/leave to be edited is not found, PUT returns 404 NOT FOUND
- On failure, PUT returns 500 SERVER ERROR

DELETE responses

- DELETE returns 204 NO CONTENT
- On failure, DELETE returns 500 SERVER ERROR

6. Examples

In order to test you API endpoints you can use Postman as well as the proprietary LMS dashboard which can be downloaded from here. We provide a set of example API calls via Postman and the LMS dashboard.

6.1. Postman

Create new Employee

Below, we create a new employee using a POST message. The body of the message contains a JSON string describing the new employee. The employee id (primary key) is not provided in the JSON string as it will be automatically initiated by the database during INSERT query.
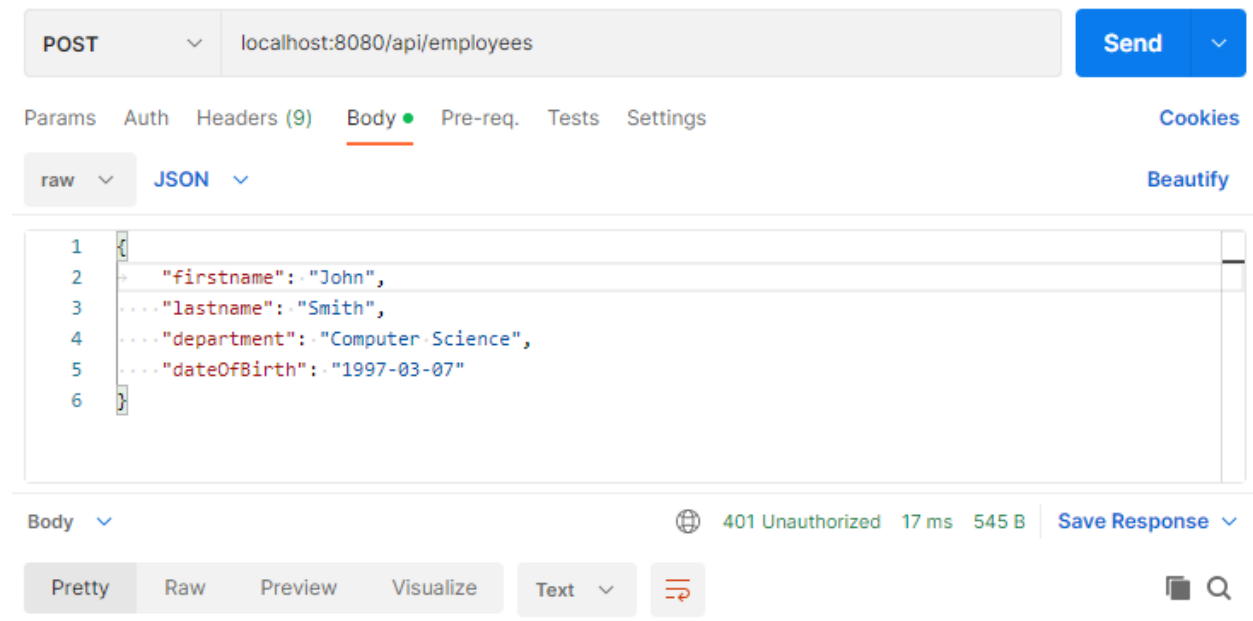
Figure 4: Create employee using POST message without user credentials.

As can be seen, the 401 Unauthorized response is returned. This message is automatically issued by the Spring Boot Security mechanism. We need to provide the credentials of a user possessing a managerial role since only managers are authorized to send POST messages.

If we provide the credentials of an employee (not a manager), the 403 Forbidden message is returned as shown in the next screenshot. This message is also issued by the Spring Boot Security mechanism.

Figure 5: Create employee using POST message with wrong (non-managerial) user credentials.

Finally, if we provide the proper user credentials, the message 201 Created is returned along with the JSON string of the newly created employee.

Figure 6: Create employee using POST message with managerial user credentials.

Retrieve Employee with a given id

Figure 7: Retrieve an employee with a given id using a GET message.

Retrieve All Employees

Figure 8: Retrieve all employees using a GET message.

Edit Employee

Figure 9: Edit an employee with a specified id using a PUT message.
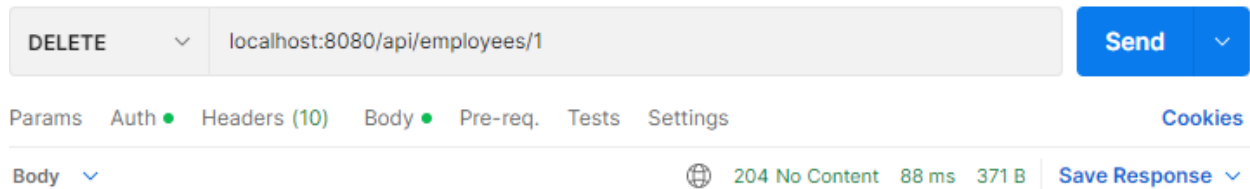
Delete Employee

Figure 10: Delete an employee with a specified id using a DELETE message.
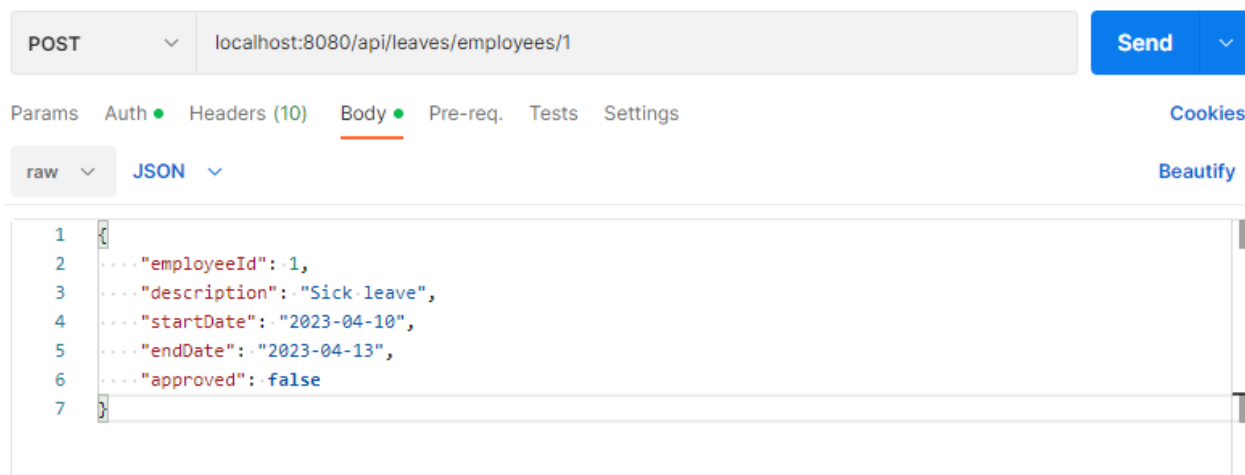
Create new Leave

Figure 11: Create leave using POST message with managerial user credentials.
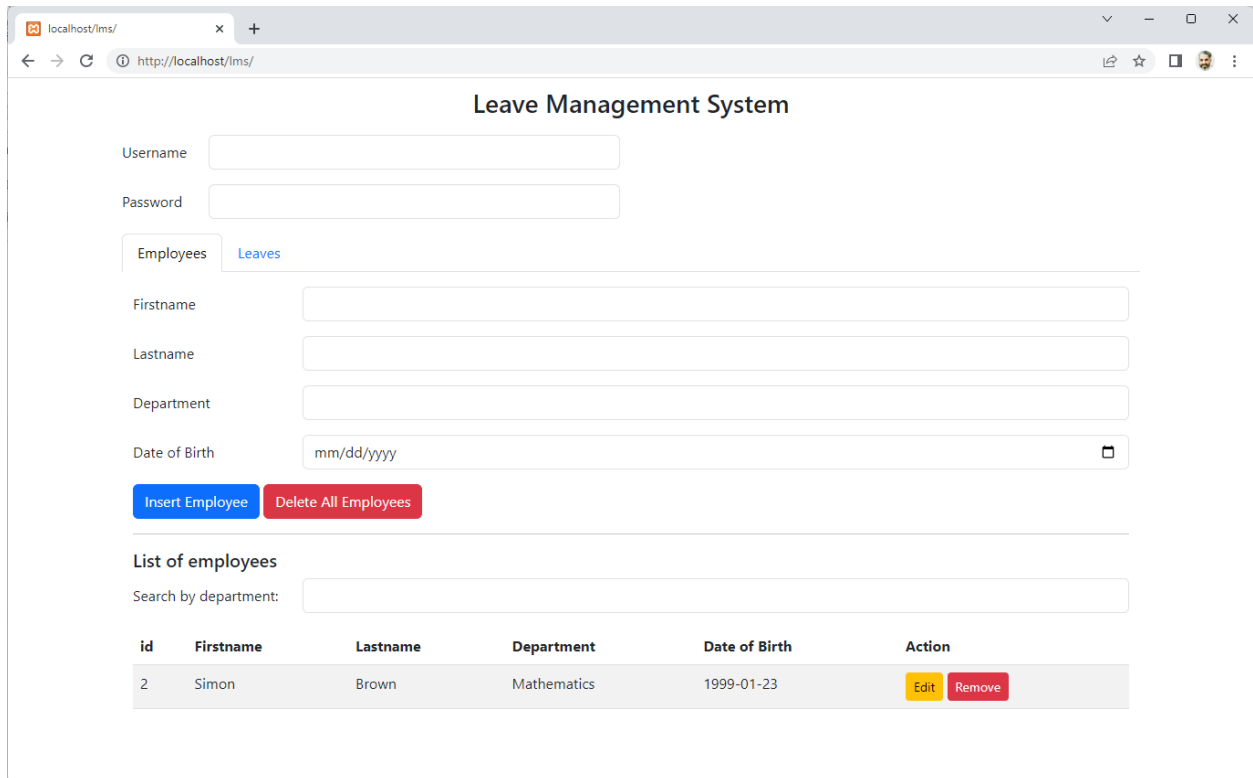
6.2. LMS Dashboard

The LMS dashboard can be used to test all API endpoints in a more visually appealing way as well as for you to know whether you implemented all API endpoints properly.

Download the LMSDashboard.zip and extract it preferably within the web server document root directory (e.g C:\xampp\htdocs\lms). Then open the script.js and modify (a) the hostname variable with the domain name or the IP address of the machine hosting the API as well as (b) the port variable if applicable. If the API is running on your localhost through the default Apache Tomcat port 8080, you do not need to modify the aforementioned variables.

If you placed the files as instructed above, you can access the dashboard via http://localhost/lms

Important: In case you face a CORS error when testing your API using the LMS Dashboard from the localhost, then follow the guidelines shown in the last slide of Lab11 in bold and highlighted text in order to disable CORS.
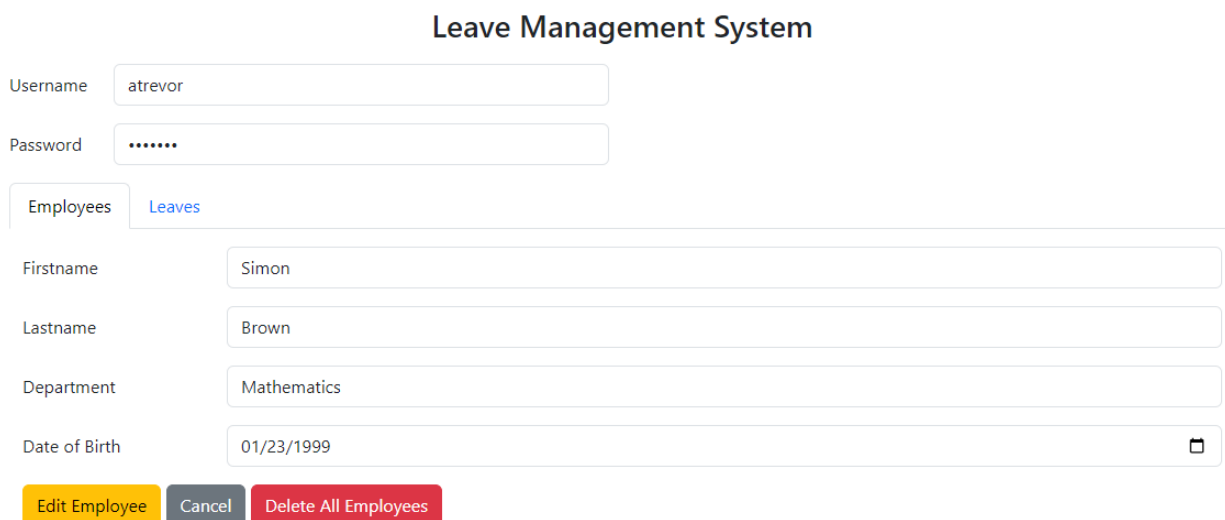
As soon as the dashboard is up and running, the list of all employees is loaded from the database (GET /api/employees).

Each employee can be edited or deleted from the system. When you click the Edit button, the involved employee is retrieved (GET /api/employees/{id}) and the form fields are filled. At the same time, the Edit Employee button replaces the Insert Employee. By clicking the Edit Employee button, the modified information of the employee is submitted (PUT /api/employees/{id})

The Remove button deletes the involved employee (DELETE /api/employees/{id}) from the system.

You can also insert an employee by filling all the necessary information in the form and by clicking the Insert Employee button (POST /api/employees). There is also an option to delete all employees (DELETE /api/employees). Finally, there is an option to filter the list of employees based on the department name (GET /api/employees?department={xxx}).

In the Leave tab, the list of leaves is loaded from the database (GET /api/leaves).

Each leave can be edited or deleted from the system. When you click the Edit button, the involved leave is retrieved (GET /api/leaves/{id}) and the form fields are filled. At the same time, the Edit Leave button replaces the Insert Leave. By clicking the Edit Leave button, the modified

information of the leave is submitted (PUT /api/leaves/{id}). The Remove button deletes the involved leave (DELETE /api/leaves/{id}) from the system.

You can also insert a leave by filling all the necessary information in the form and by clicking the Insert Leave button (POST /api/leaves). You can also delete all leaves (DELETE /api/leaves). Another option is to filter the list of leaves based on the start date, end date and/or approved state (GET /api/leaves?startDate={xxx}&endDate={yyy}&approved={true/false}).

All actions (except those involving GET messages) can be performed when the username and password of a user (managerial role) are given in the dedicated fields.

In case you do not provide any credentials, the 401 User Unauthorized message is displayed if you try to perform an action involving a POST/PUT/DELETE message.

In case you provide the credentials of a non-Manager user the 403 Forbidden message is displayed.

| Username | jsmith |
| Password | •••••• |

<div style="color:red">Access to Resource Forbidden (403)</div>

After submission (Edit or Insert) or after clicking the Cancel button, the form fields are cleared.

Important notice: <u>Do not modify</u> the LMS dashboard in order to be seamlessly connected to your RESTful API but try to follow all the aforementioned guidelines so as to make your API fully compatible with the given dashboard.

7. Submission

Your RESTful API will get full marks if it fully compatible with the LMS dashboard.

In case you don't implement all requested functionalities, provide a readme.txt file to document them accordingly.

Finally, compress the folder of your Spring Boot application as a .zip file and submit it to Moodle.
.