EE4216 Lab 3 – Full Stack Web App Design

(all checkpoints require verification in the lab session)

Objectives:

- Learn to build RESTful web service
- Learn to query H2 database with JdbcTemplate
- Learn to integrate Vue2 frontend with Spring RESTful API

RESTful web service is an architecture-neutral design for systems to communicate with each other across the Internet. In this lab, we will build an interactive web UI to browse and edit a movie database. The system composes of a Vue frontend and a Spring RESTful backend.

Task 1 – Creating Spring Boot Maven Project

Create a new Maven project using Spring Initializr (https://start.spring.io/) or Netbeans Spring Boot plugin. We are using JDK 17 and Spring Boot version 2.7.x. You will need the following dependencies:

- spring-boot-starter-web
- spring-boot-starter-jdbc
- h2

After setting up the new project, you should try to build it to let Maven download all the dependencies required. Make sure the project is built without error before you proceed to the next task.

Task 2 – Configuring H2 Database and Importing Data from IMDB

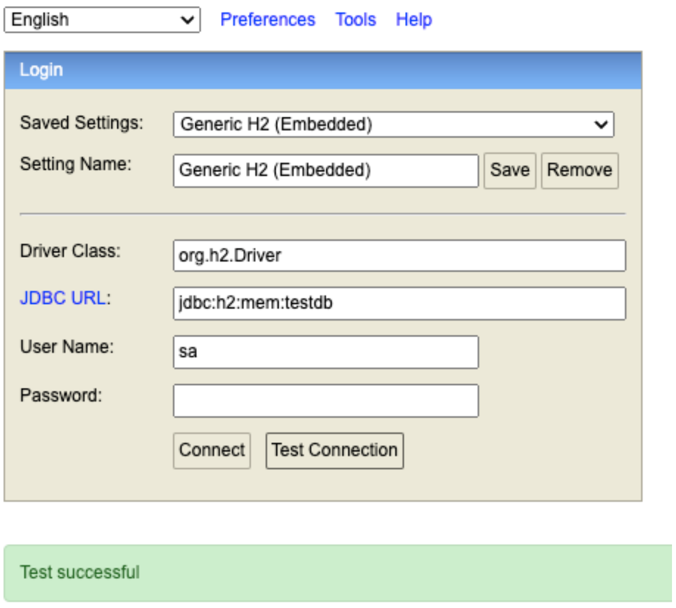
Now, we need to configure an embedded in-memory H2 database. You should add the following settings to your spring configuration file **/resources/application.properties**.

- spring.datasource.url=jdbc:h2:mem:testdb
- spring.datasource.driverClassName=org.h2.Driver
- spring.datasource.username=sa

- spring.h2.console.enabled=true
- spring.h2.console.path=/h2-console

The above settings will enable the built-in H2 web console, and you can perform administrative tasks on your H2 database using the web console. You can access the console with a localhost URL - http://localhost[:port]/h2-console/.

To test the database settings, you can click the button [Test Connection] to check. A successful message will be shown if the settings are correct.



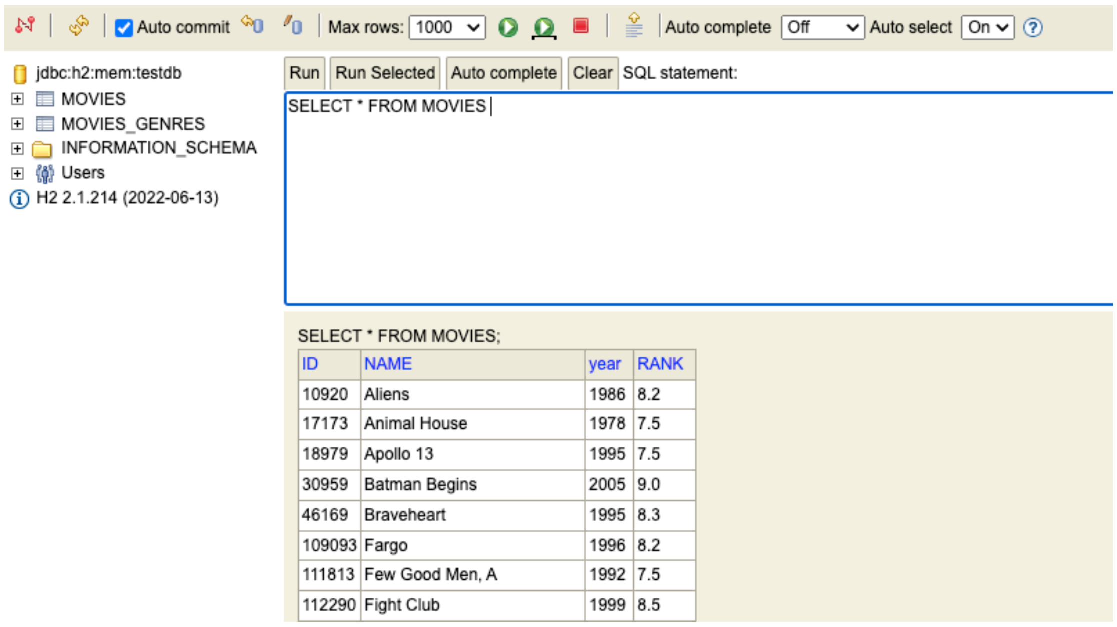
Saved Settings: Generic H2 (Embedded)
Setting Name: Generic H2 (Embedded)
Driver Class: org.h2.Driver
JDBC URL: jdbc:h2:mem:testdb
User Name: sa
Password:

Test successful

You are given a SQL file which contains the IMDB movie data. You can download it from Canvas and put it under the **/resources** folder. The file name must be **data.sql** so that it will be automatically executed by Spring and import the data.

You can restart your Spring project and log in to the H2 web console to check if the table **movies** is created and populated with some movie data.

Checkpoint 1:
Show the movie data in your movies table using the web console as below.



```
SELECT * FROM MOVIES
```

SELECT * FROM MOVIES;

ID	NAME	year	RANK
10920	Aliens	1986	8.2
17173	Animal House	1978	7.5
18979	Apollo 13	1995	7.5
30959	Batman Begins	2005	9.0
46169	Braveheart	1995	8.3
109093	Fargo	1996	8.2
111813	Few Good Men, A	1992	7.5
112290	Fight Club	1999	8.5

Task 3 – Creating RESTful API Endpoints

We will create three RESTful API endpoints:

- GET - /api/movies
- PUT - /api/movies/{id}
- DELETE - /api/movies/{id}

The first one is used to retrieve all movie data in JSON format. The other two endpoints are used to update or delete a movie with a specific id.

You should create three Java classes in your Spring project:

- A REST controller, **MovieController**, for mapping the above paths to proper handlers.
- An entity class, **Movie**, represents a row of movie data from the database table.
- A DAO class, **MovieDao**, interacts with the database and performs CRUD operations on the movies table using **JdbcTemplate**.

Checkpoint 2:

Use **cURL** to test your endpoints. You can enter the following commands in a terminal to verify the output.

- `curl -i http://localhost/api/movies`

-
  ```
  curl -i -X PUT -H "Content-Type: application/json"
  -d '{"id":10920,"name":"Aliens 2","year":1986,"rank":9.8}'
  http://localhost/api/movies/10920
  ```

-
  ```
  curl -i -X DELETE http://localhost/api/movies/10920
  ```

Show the outputs of the commands to the instructor. Demonstrate that the movie data can be updated or deleted using the RESTful endpoints.

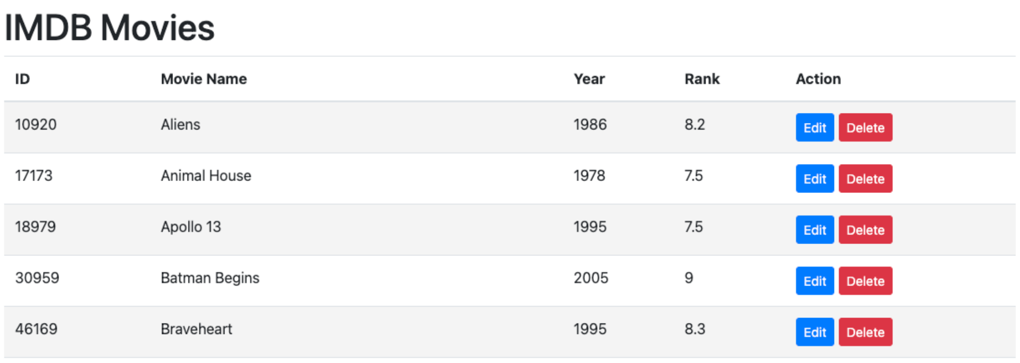
Task 3 – Integrating with Vue2 Frontend

The final task is to build a Vue client to allow a user to browse, edit and delete the movies from the database. You will create two files:

- /resources/static/index.html
- /resources/static/js/script.js

The first HTML file is a Vue template provided to you (see below). The second JS file contains your Vue client code. Your Vue client must communicate with the Spring backend using the RESTful API to perform read, update and delete operations.

The UI of the Vue client is shown below. It displays all movies in a table format and provides two buttons on the right side.

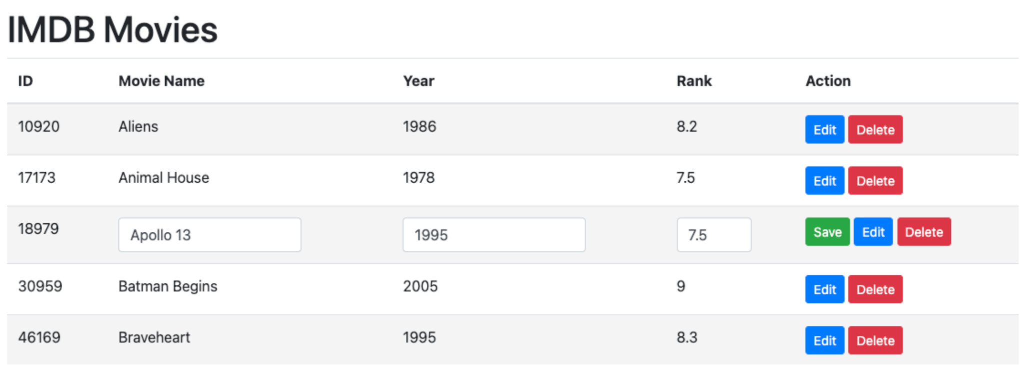


IMDB Movies

ID	Movie Name	Year	Rank	Action
10920	Aliens	1986	8.2	Edit Delete
17173	Animal House	1978	7.5	Edit Delete
18979	Apollo 13	1995	7.5	Edit Delete
30959	Batman Begins	2005	9	Edit Delete
46169	Braveheart	1995	8.3	Edit Delete

To edit a row, the user can click the [Edit] button, and the input fields will be shown in place. The user can directly edit the data and click the [Save] button to save the result to the database.

Similarly, the [Delete] button deletes a row from the database. The data table should immediately show the changes made by these operations.

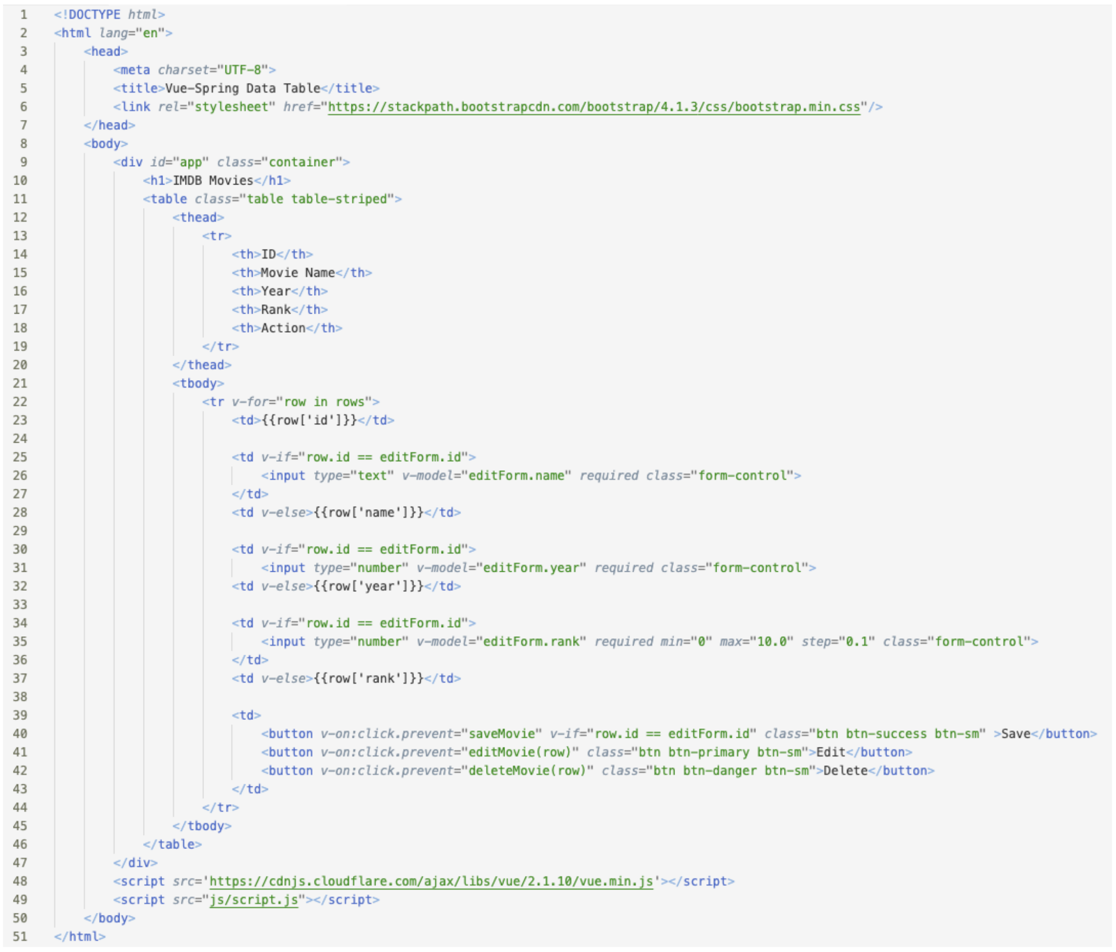


IMDB Movies

ID	Movie Name	Year	Rank	Action
10920	Aliens	1986	8.2	Edit Delete
17173	Animal House	1978	7.5	Edit Delete
18979	Apollo 13	1995	7.5	Save Edit Delete
30959	Batman Begins	2005	9	Edit Delete
46169	Braveheart	1995	8.3	Edit Delete

Read the following template carefully and design the corresponding Vue app to mount onto the <div id="app">. Pay attention to the names of the variables and functions and define them in your Vue app accordingly.

index.html



```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Vue-Spring Data Table</title>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"/>
    </head>
    <body>
        <div id="app" class="container">
            <h1>IMDB Movies</h1>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Movie Name</th>
                        <th>Year</th>
                        <th>Rank</th>
                        <th>Action</th>
                    </tr>
                </thead>
                <tbody>
                    <tr v-for="row in rows">
                        <td>{{row['id']}}</td>

                        <td v-if="row.id == editForm.id">
                            <input type="text" v-model="editForm.name" required class="form-control">
                        </td>
                        <td v-else>{{row['name']}}</td>

                        <td v-if="row.id == editForm.id">
                            <input type="number" v-model="editForm.year" required class="form-control">
                        <td v-else>{{row['year']}}</td>

                        <td v-if="row.id == editForm.id">
                            <input type="number" v-model="editForm.rank" required min="0" max="10.0" step="0.1" class="form-control">
                        </td>
                        <td v-else>{{row['rank']}}</td>

                        <td>
                            <button v-on:click.prevent="saveMovie" v-if="row.id == editForm.id" class="btn btn-success btn-sm" >Save</button>
                            <button v-on:click.prevent="editMovie(row)" class="btn btn-primary btn-sm">Edit</button>
                            <button v-on:click.prevent="deleteMovie(row)" class="btn btn-danger btn-sm">Delete</button>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
        <script src='https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.10/vue.min.js'></script>
        <script src="js/script.js"></script>
    </body>
</html>
```

Checkpoint 3:

- Demonstrate your full stack web app and show how you can edit and delete the movies.

- END -