

# Practice Develop RESTful Services by Using Spring Boot by Using JPA

# Practice

- Develop an Application for Movie Service



An illustration of two people, a woman with dark hair and glasses wearing a red top, and a man with brown hair and glasses wearing a yellow top, sitting at a desk. They are looking at a large blue computer monitor. On the desk, there is a yellow clipboard, a white coffee cup with a red lid, a yellow pencil, and a notepad with a red pencil. The background is light green with some abstract shapes and a large green plant on the right.

## PRACTICE

### **Practice: Develop an Application for Movie Service**

In the previous sprint, we developed a simple Spring Boot application using the Spring Initializr. This challenge will help you create a boot application to work with application entities using an in-memory database.

Create a Spring Boot application with one domain class called Movie and provide the Service, Controller, and Repository layers.



# Implementation Environment

- Create a Spring Boot application from the Spring [Initializr](#).
- Add the necessary dependencies to pom.xml
- Download the project to your local machine.
- Extract the zip file.
- Export the project to your local IDE.

## Dependencies

ADD DEPENDENCIES... CTRL + B

### Spring Web

WEB

Build web, including RESTful, applications using Spring MVC.  
Uses Apache Tomcat as the default embedded container.

### Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

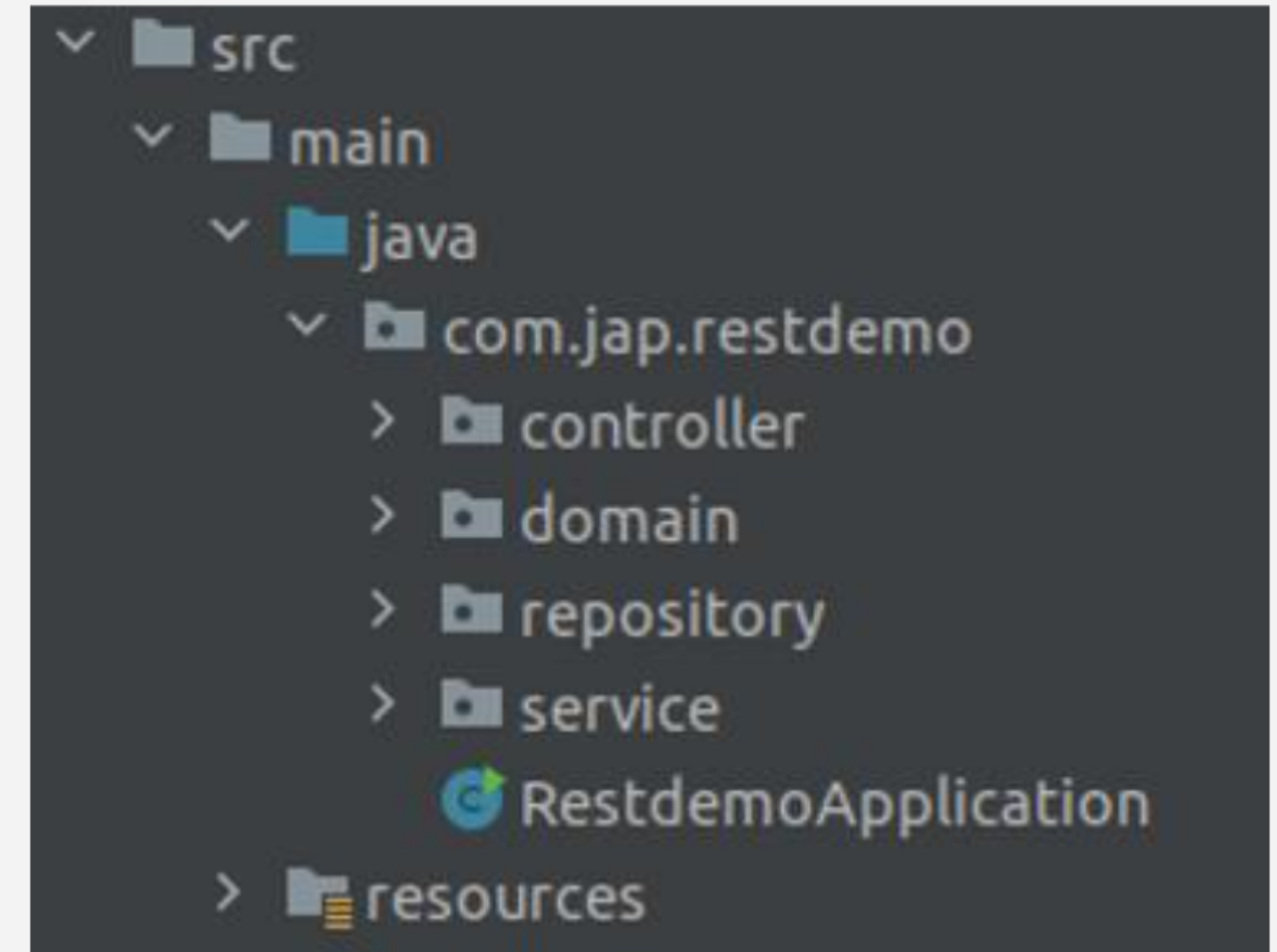
### H2 Database

SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

# Practice: Task 1

- The structure of the project is given for your reference.
- Create the domain class `Movie` with the following attributes: `movieName`, `actorName`, `directorName`, `movieId`.
- Annotate the `Movie` domain class with `@Entity` and `movieId` with `@Id`.
- Create getters and setters for all the attributes.
- Create the Repository Interface that extends `CrudRepository` past two parameters: first, the entity class name, and second, the data type of the `@Id` attribute.



# Practice: Task 2

- Inside the Service package, create two Java files: `MovieService` Interface and `MovieServiceImpl` class.
- Annotate the `MovieServiceImpl` class with `@Service`.
- Inside the `MovieService` Interface, create the method to save the movie object.
- Implement this interface inside the `MovieServiceImpl` class and override the method.
- Autowire `MovieRepository` inside the service layer.
- Call the `MovieRepository save()` method inside the service class to save the movie object in the H2 database.



# Practice: Task 3

- Inside the Controller package, create the `MovieController` class
- Annotate this class with `@RestController` and `@RequestMapping`.
- Autowire `MovieService` inside the controller layer.
- Create the handler method to save the movie data by calling the service save method.
- Annotate this handler method with `@PostMapping`.
- Set up the H2 database configuration details in the `application.properties` file.
- Run the boot application by using the Spring way of execution.
- Open Postman and call the REST API.
- Open the H2 console and check that the tables are created.

# Postman Output

The image shows the Postman application interface. At the top, a POST request is configured to `http://localhost:8080/api/v1/movie`. The 'Body' tab is selected, and the data type is set to 'JSON'. The request body contains the following JSON:

```
{
  "movieName": "The Shawshank Redemption ",
  "actorName": "Tim Robbins",
  "directorName": "Frank Darabont",
  "movieId": 101
}
```

Below the request editor, the 'Test Results' tab is active, displaying the response details:

- Status: 201 Created
- Time: 10 ms
- Size: 282 B

The response body is shown in 'Pretty' format, displaying the same JSON structure as the request body.



# Submission Instructions

- There is no boilerplate for the practice.
- Create a Git repository named **BEJ\_C1\_S5\_REST\_API\_PC\_1**.
- After completing the practice, push the code back to git using the below commands.

```
git init
git remote add origin <url>
git add .
git commit -m "comments on the push"
git push -u origin master
```

- Submit it for review.