



Spring REST CRUD Demo with JPA Configuration

Learn to **create REST APIs for CRUD operations** using **Spring REST** and **JPA** configuration (H2 database as backend) **with Spring boot** auto configuration feature.

Agenda:

1. **Creating a Spring Boot Project from Spring Templates**
2. **Setting Up the Spring Starter Template in your preferred IDE**
3. **Entity class**
4. **Repository**
5. **Service**
6. **REST Controller**
7. **H2: In-Memory DB Setup**
8. **Spring REST CRUD Operations Demo**

1. Creating a Spring Boot Project from [Spring Templates](#)

The screenshot shows the start.spring.io web interface. The interface is divided into several sections:

- Project:** Radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Radio buttons for versions: 2.3.1 (SNAPSHOT), **2.3.0** (selected), 2.2.8 (SNAPSHOT), 2.2.7, 2.1.15 (SNAPSHOT), and 2.1.14.
- Project Metadata:** Text input fields for:
 - Group: `com.crpdev`
 - Artifact: `spring-crud`
 - Name: `spring-crud`
 - Description: `JPA CRUD project using Spring Boot`
 - Package name: `com.crpdev.spring-crud`
- Packaging:** Radio buttons for **Jar** (selected) and **War**.
- Java:** Radio buttons for versions: 14, 11, and **8** (selected).
- Dependencies:** A section with a button "ADD DEPENDENCIES... + B". It lists two dependencies:
 - 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.

At the bottom, there are three buttons: **GENERATE** (with a keyboard shortcut icon), **EXPLORE** (with a keyboard shortcut icon), and **SHARE...**

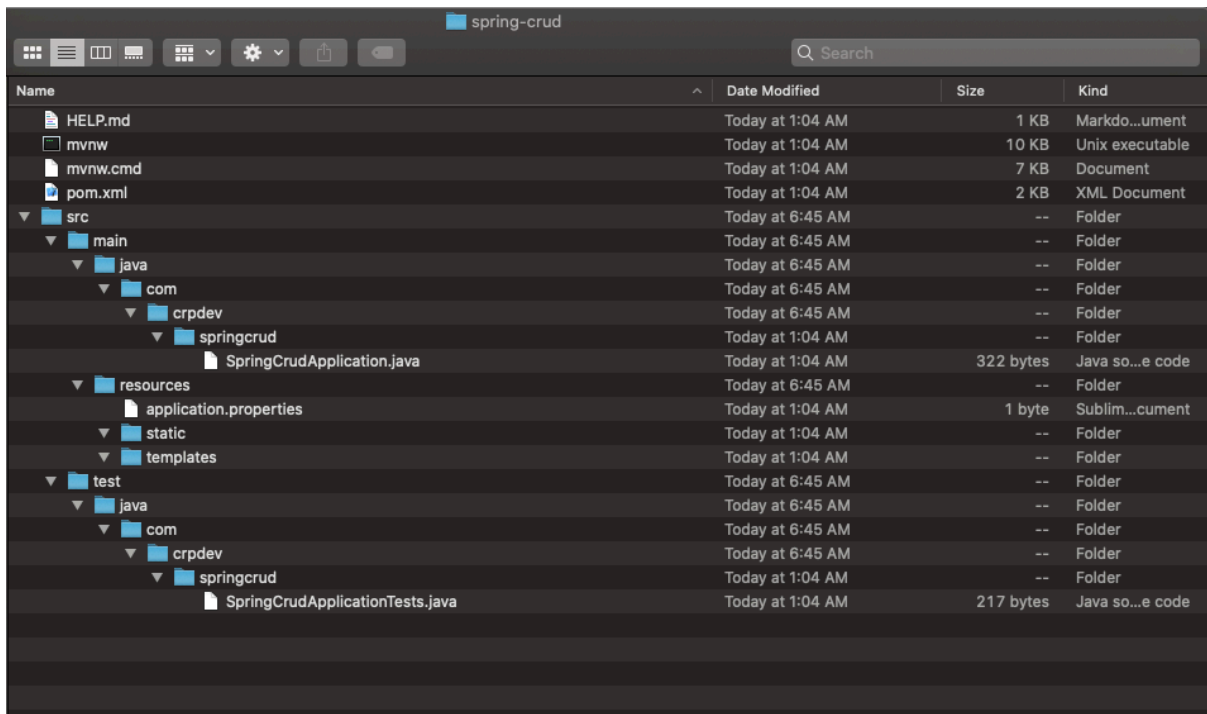
Our motive is to implement a JPA CRUD functionality using Spring Boot. So all that is needed now are:

- Spring Web - Dependency to expose the functionality as REST endpoints
- Spring Data JPA - Define entities and perform CRUD operations on embedded H2 database

Once the two dependencies are added, click **“Generate”** which will download a zip file of the ***Spring Starter Template*** to your system.

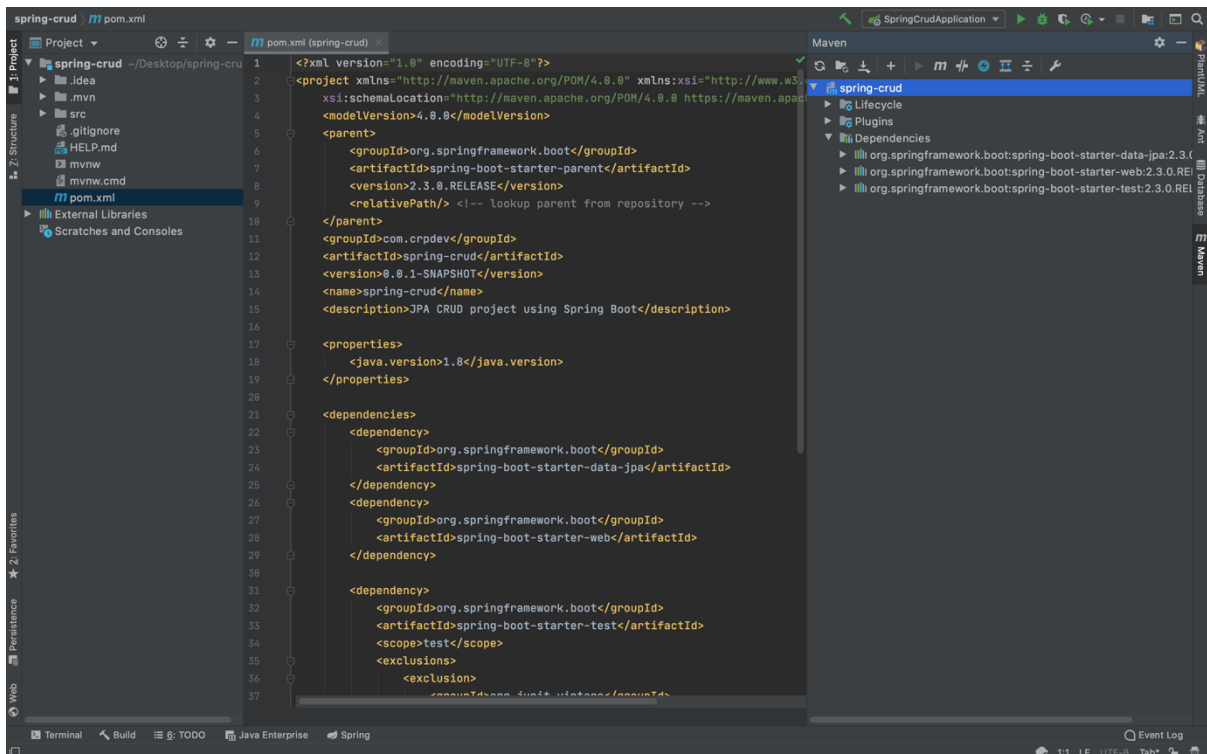
2. Setting Up the Spring Starter Template in your preferred IDE

Extract the downloaded project zip



On your preferred IDE, import the downloaded starter template.

When defining our template, we opted to have a **Maven project**, hence after Import the IDE will setup the project by downloading all Spring Starter dependencies defined in the pom file.



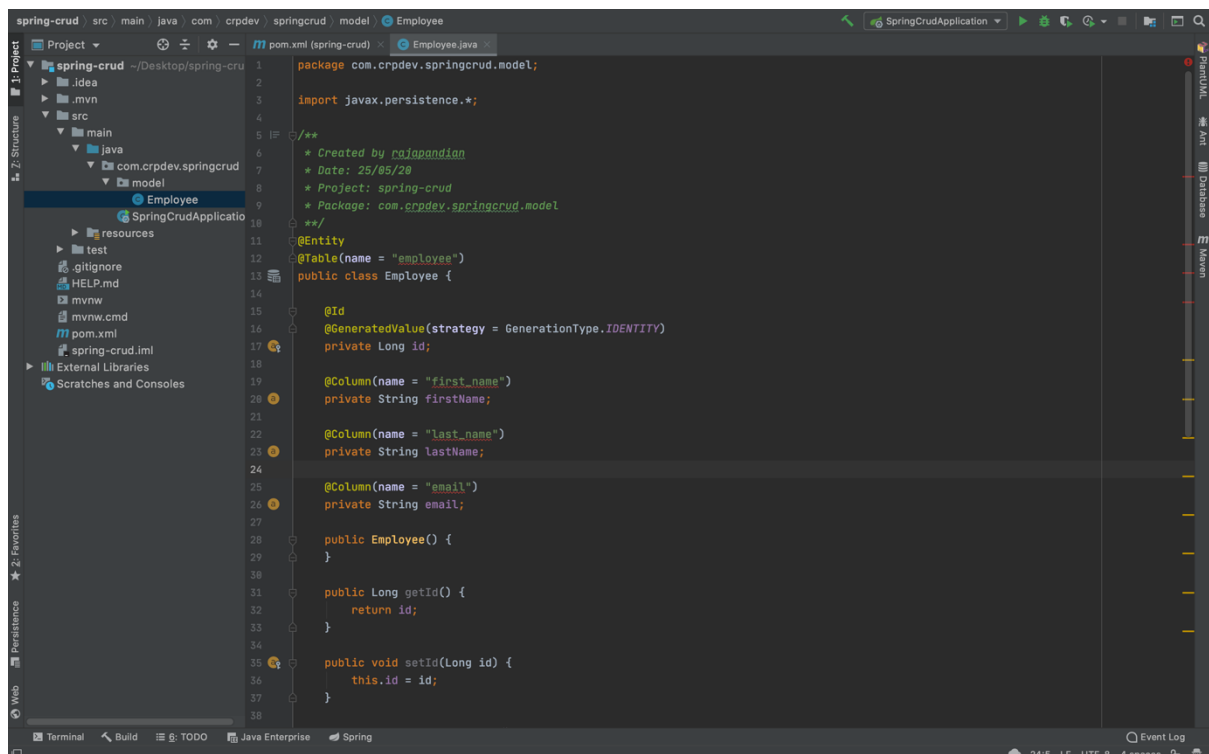
3. Entity class

Create a new package named “model” and an Employee class.

We have only one entity class which will be persisted in DB and will be returned as response from REST APIs.

We have only one entity class which will be persisted in DB and will be returned as response from REST APIs.

- **@Entity** – JPA annotation to make the object ready for storage in a JPA-based data store.
- **@Table** – Name of the table in data storage where this entity will be stored.
- Define 4 attributes to this class [*id*, *firstName*, *lastName* and *email*] and generate the default constructor and Getter-Setter
- By convention, though we follow camel case in Java, snake case is followed in DB and we can define them using annotations.
- The id attribute is marked with annotations “Id” and “**GeneratedValue**”, which will set the primary key and manage setting a unique value based on the “**GenerationType.IDENTITY**” property set. Most of the current relational databases support this functionality. If you have a custom Sequence defined, you can use it using “**GenerationType.SEQUENCE**”.



```

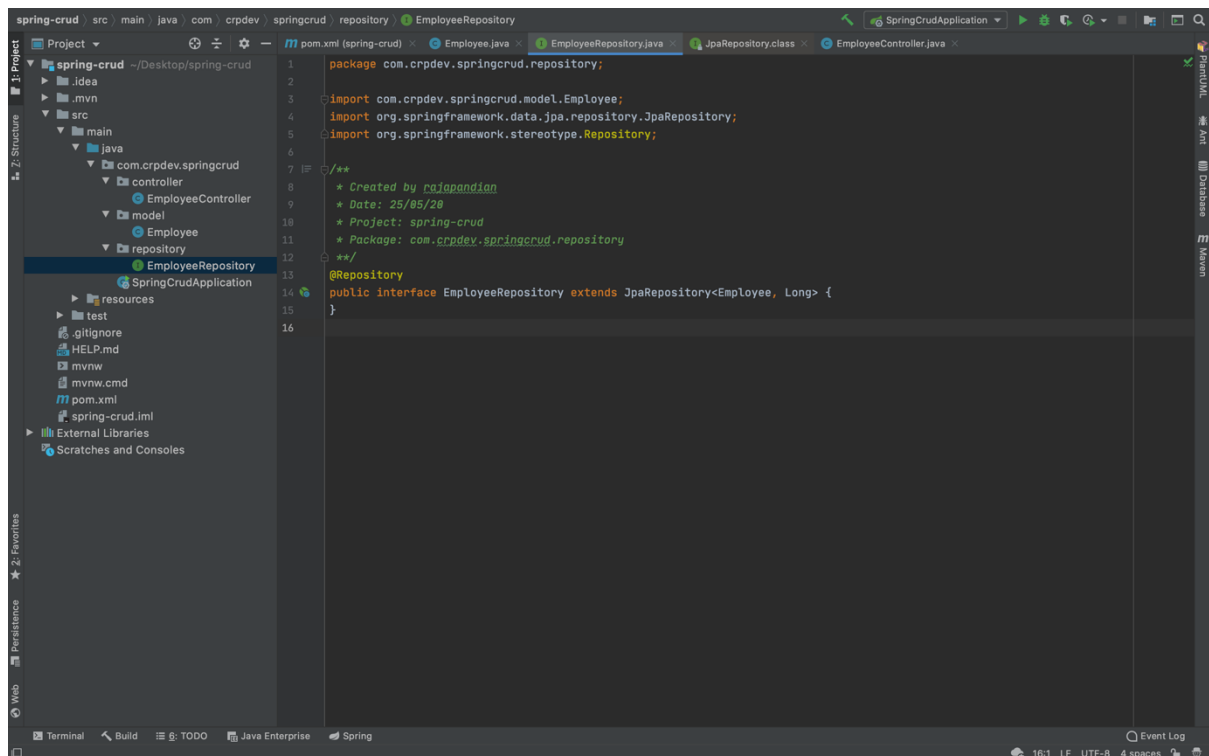
1 package com.crpdev.springcrud.model;
2
3 import javax.persistence.*;
4
5 /**
6  * Created by rajapandian
7  * Date: 25/05/20
8  * Project: spring-crud
9  * Package: com.crpdev.springcrud.model
10 */
11 @Entity
12 @Table(name = "employee")
13 public class Employee {
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18
19     @Column(name = "first_name")
20     private String firstName;
21
22     @Column(name = "last_name")
23     private String lastName;
24
25     @Column(name = "email")
26     private String email;
27
28     public Employee() {
29     }
30
31     public Long getId() {
32         return id;
33     }
34
35     public void setId(Long id) {
36         this.id = id;
37     }
38 }

```

4. Repository

Create a new package named “**repository**” and an **EmployeeRepository** interface. Annotate the Interface with **@Repository stereotype**. This will help the Spring Context inject this dependency at start-up.

The reason we are creating an Interface instead of a Class is that, we are going to use Spring Data JPA features instead of re-writing what is already available to us.



We have created **JpaRepository** implementation as **EmployeeRepository** which provides all default operations for performing CRUD – Create, Read, Update and Delete operations on **Employee** entity.

Spring Data JPA’s **JpaRepository** has made available a generic implementation that gives us the functionality needed.

Note:

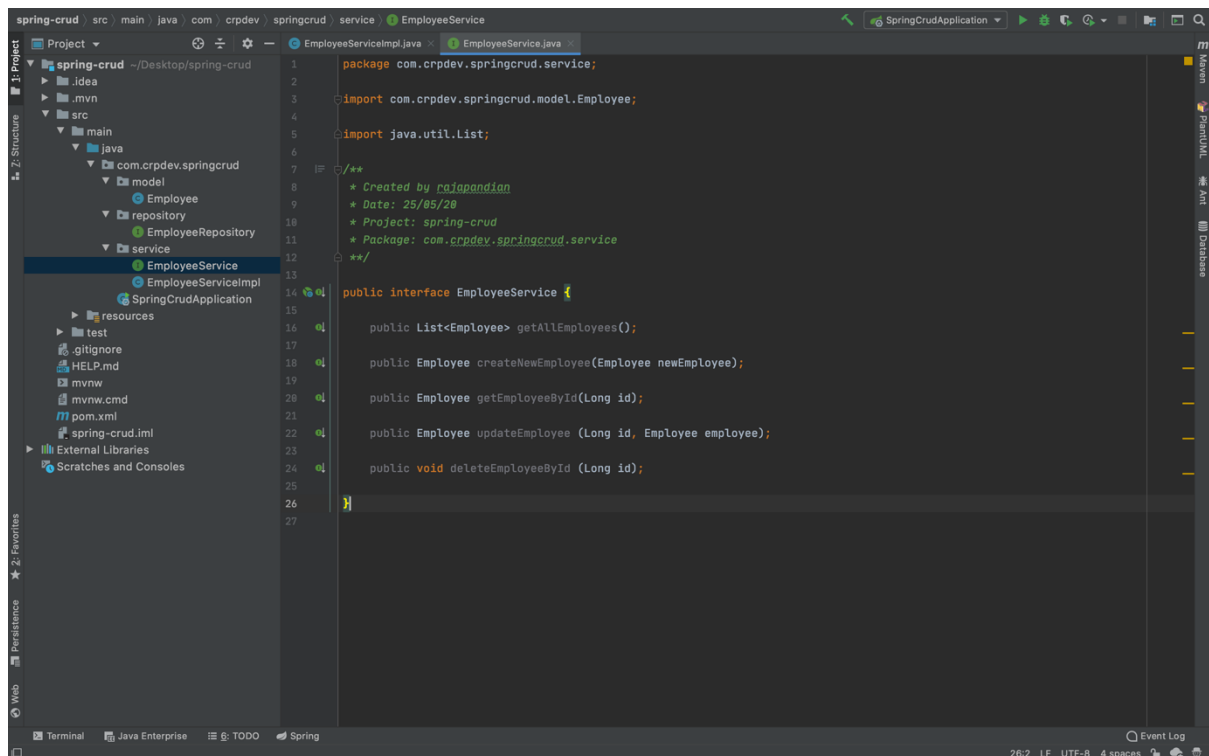
Spring Data JPA provides **CRUDRepository** Interface and a **JpaRepository** Interface.

- **CRUDRepository** is a basic interface only to perform CRUD operations.
- **JpaRepository** provides additional functionalities by extending **CRUDRepository**

5. Service

Create a new package named “**service**” and an **EmployeeService** interface. This Interface will define the methods that will be exposed via the controller as REST endpoints.

It's a good design principle to define a service layer, that acts as an interaction handler between the exposed endpoints and the data layer. Direct interaction between the presentation and business logic is not recommended in ideal scenarios.



We will now create an implementation class “**EmployeeServiceImpl**” that will implement the “**EmployeeService**” Interface.

Annotate the Interface with **@Service stereotype**. This will help the Spring Context inject this dependency at start-up.

The reason for “**Programming To An Interface**” is that `EmployeeService` can take any form at runtime. In this case, we will be using our `EmployeeRepository` to perform CRUD operations. At runtime, we might have to perform CRUD using an external service. Hence, this approach will help us define an opinionated programming model.

The `EmployeeServiceImpl` class will make use of the `EmployeeRepository` [Dependency Injection] and perform CRUD operations. Since `EmployeeServiceImpl` class does not have logic on its own and will always rely on the repository, we add the repository to the constructor.

At runtime, any implementation of the `EmployeeService` can be passed and the program will execute as desired.

```

1 package com.crpdev.springcrud.service;
2
3 import com.crpdev.springcrud.model.Employee;
4 import com.crpdev.springcrud.repository.EmployeeRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9
10 /**
11  * Created by rajapandian
12  * Date: 25/05/20
13  * Project: spring-crud
14  * Package: com.crpdev.springcrud.service
15  */
16 @Service
17 public class EmployeeServiceImpl implements EmployeeService {
18
19     private EmployeeRepository employeeRepository;
20
21     // Autowired is not mandatory on constructors post Spring 3.5
22     @Autowired
23     public EmployeeServiceImpl(EmployeeRepository employeeRepository) {
24         this.employeeRepository = employeeRepository;
25     }
26
27     public List<Employee> getAllEmployees(){
28         return employeeRepository.findAll();
29     }
30
31     public Employee createNewEmployee(Employee newEmployee){
32         Employee savedEmployee = employeeRepository.save(newEmployee);
33         return savedEmployee;
34     }
35
36     public Employee getEmployeeById(Long id){
37         return employeeRepository.findById(id).orElse( other: null);
38     }
39 }

```

```

40
41 @Override
42 public Employee updateEmployee(Long id, Employee updatedEmployee) {
43     return employeeRepository.findById(id).map(employee -> {
44         employee.setFirstName(updatedEmployee.getFirstName());
45         employee.setLastName(updatedEmployee.getLastName());
46         employee.setEmail(updatedEmployee.getEmail());
47         return employee;
48     }).orElseGet(() -> {
49         updatedEmployee.setId(id);
50         return employeeRepository.save(updatedEmployee);
51     });
52
53 @Override
54 public void deleteEmployeeById(Long id) {
55     employeeRepository.deleteById(id);
56 }
57
58 }
59

```

We've implemented the methods defined in **EmployeeService** Interface by injecting **EmployeeRepository**.

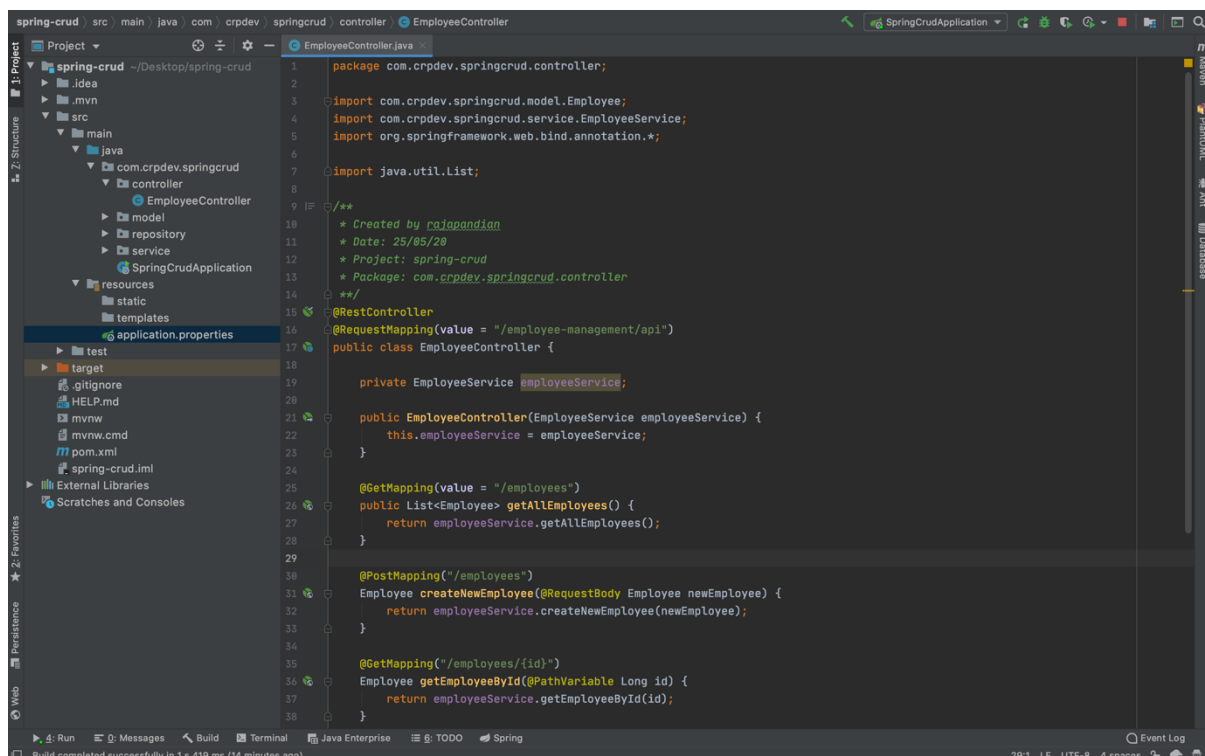
6. REST Controller

Create a new package named “**controller**” and an **EmployeeController** class. This Class will define the REST endpoints that will be exposed.

Annotate the Interface with **@RestController** stereotype. This will help the Spring Context inject this dependency at start-up.

The **EmployeeController** class will make use of the **EmployeeService** [Dependency Injection] to perform CRUD operations through the exposed REST endpoints.

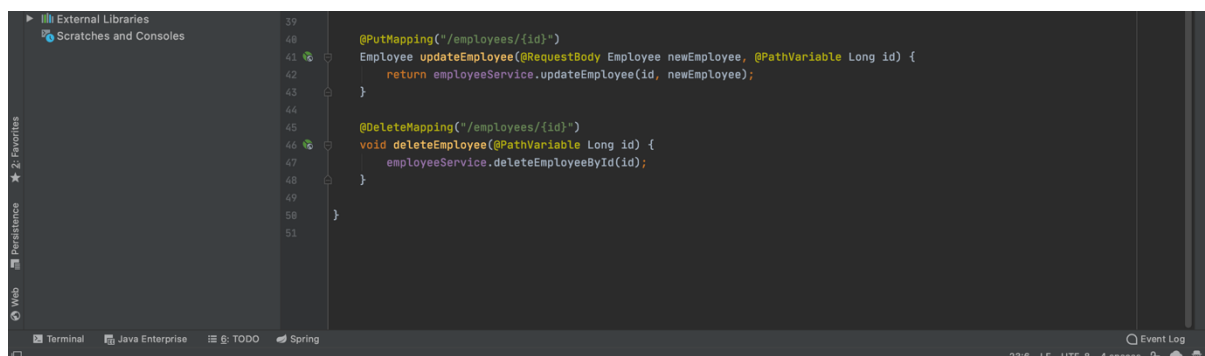
In given controller, **@RestController** annotation indicates that the data returned by the methods will be written straight into the response body instead of rendering a template. Other annotations (**@GetMapping**, **@PostMapping**, **@PutMapping** and **@DeleteMapping**) map the HTTP requests to corresponding methods.



```

1 package com.crpdev.springcrud.controller;
2
3 import com.crpdev.springcrud.model.Employee;
4 import com.crpdev.springcrud.service.EmployeeService;
5 import org.springframework.web.bind.annotation.*;
6
7 import java.util.List;
8
9 /**
10  * Created by rajapandian
11  * Date: 25/05/20
12  * Project: spring-crud
13  * Package: com.crpdev.springcrud.controller
14  */
15 @RestController
16 @RequestMapping(value = "/employee-management/api")
17 public class EmployeeController {
18
19     private EmployeeService employeeService;
20
21     public EmployeeController(EmployeeService employeeService) {
22         this.employeeService = employeeService;
23     }
24
25     @GetMapping(value = "/employees")
26     public List<Employee> getAllEmployees() {
27         return employeeService.getAllEmployees();
28     }
29
30     @PostMapping("/employees")
31     Employee createNewEmployee(@RequestBody Employee newEmployee) {
32         return employeeService.createNewEmployee(newEmployee);
33     }
34
35     @GetMapping("/employees/{id}")
36     Employee getEmployeeById(@PathVariable Long id) {
37         return employeeService.getEmployeeById(id);
38     }
39
40     @PutMapping("/employees/{id}")
41     Employee updateEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {
42         return employeeService.updateEmployee(id, newEmployee);
43     }
44
45     @DeleteMapping("/employees/{id}")
46     void deleteEmployee(@PathVariable Long id) {
47         employeeService.deleteEmployeeById(id);
48     }
49 }

```



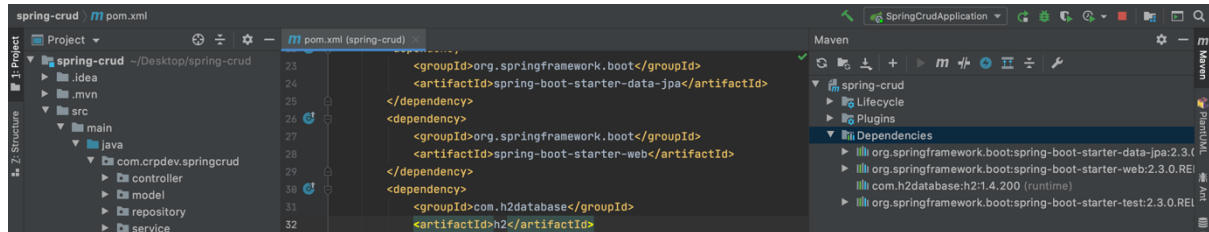
```

39
40     @PutMapping("/employees/{id}")
41     Employee updateEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {
42         return employeeService.updateEmployee(id, newEmployee);
43     }
44
45     @DeleteMapping("/employees/{id}")
46     void deleteEmployee(@PathVariable Long id) {
47         employeeService.deleteEmployeeById(id);
48     }
49 }
50
51

```


7. H2: In-Memory DB Setup

H2 is an In-Memory DB that can be used for testing Spring Data JPA applications. To use H2, we will have to add a dependency and the configuration.

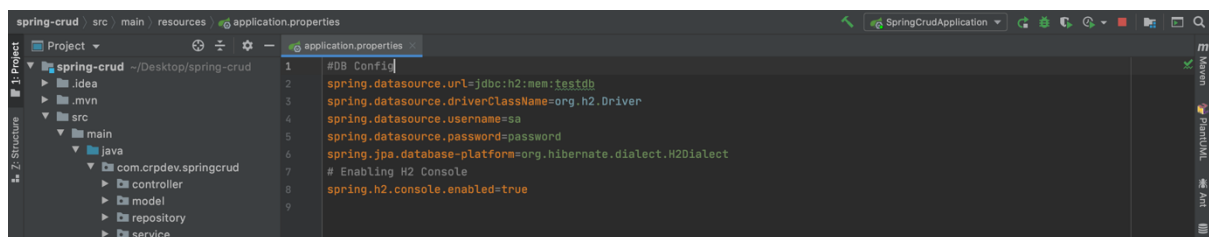


Add the **H2** dependency and refresh the pom, so the dependency is downloaded.

To make use of this H2 DB for our tests, we will have to define configurations, so that Spring Boot uses the DB at runtime.

Configuration parameters specific to the application are defined in **application.properties**.

These can also be defined in YAML files.



The first set of parameters help Spring Boot to configure the application to use H2 In-Memory DB.

The second set is to enable H2 console, to view and execute queries via a UI console.

This completes the application implementation and setup. We can now start our **Bootiful Spring Data JPA Application** and run some tests.

8. Spring REST CRUD Operations Demo

Let's now start the Spring Boot application and test the endpoints.

```

spring-crud  src  main  resources  application.properties
Run: SpringCrudApplication
Console Endpoints
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...

:: Spring Boot :: (v2.3.0.RELEASE)

2020-05-25 09:05:56.396 INFO 1775 --- [main] c.c.springcrud.SpringCrudApplication : Starting SpringCrudApplication on rajapandians-macpro.local with
2020-05-25 09:05:56.398 INFO 1775 --- [main] c.c.springcrud.SpringCrudApplication : No active profile set, falling back to default profiles: default
2020-05-25 09:05:56.983 INFO 1775 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-05-25 09:05:57.040 INFO 1775 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 49ms. Found 1 JPA re
2020-05-25 09:05:57.451 INFO 1775 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-05-25 09:05:57.462 INFO 1775 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-05-25 09:05:57.462 INFO 1775 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.35]
2020-05-25 09:05:57.568 INFO 1775 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-05-25 09:05:57.569 INFO 1775 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1136 ms
2020-05-25 09:05:57.624 INFO 1775 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-05-25 09:05:57.782 INFO 1775 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-05-25 09:05:57.787 INFO 1775 --- [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:
2020-05-25 09:05:57.875 INFO 1775 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-05-25 09:05:57.987 INFO 1775 --- [task-1] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-05-25 09:05:57.930 WARN 1775 --- [task-1] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, databa
2020-05-25 09:05:57.937 INFO 1775 --- [task-1] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.15.Final
2020-05-25 09:05:58.026 INFO 1775 --- [task-1] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2020-05-25 09:05:58.143 INFO 1775 --- [task-1] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2020-05-25 09:05:58.511 INFO 1775 --- [task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engin
2020-05-25 09:05:58.516 INFO 1775 --- [task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'defau
2020-05-25 09:05:58.830 INFO 1775 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-05-25 09:05:58.831 INFO 1775 --- [main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories..
2020-05-25 09:05:58.868 INFO 1775 --- [main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-05-25 09:05:58.876 INFO 1775 --- [main] c.c.springcrud.SpringCrudApplication : Started SpringCrudApplication in 2.75 seconds (JVM running for 3
  
```

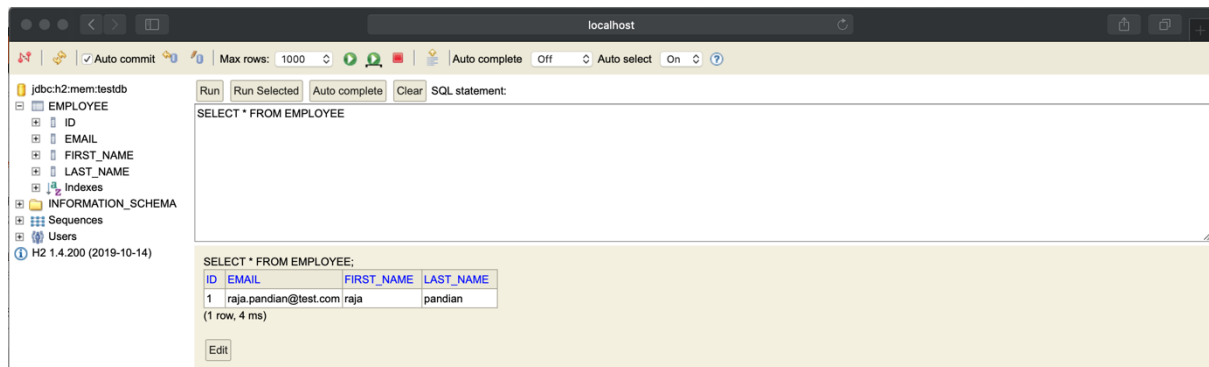
Create New Employee:

```

Postman
My Workspace Invite
No Environment

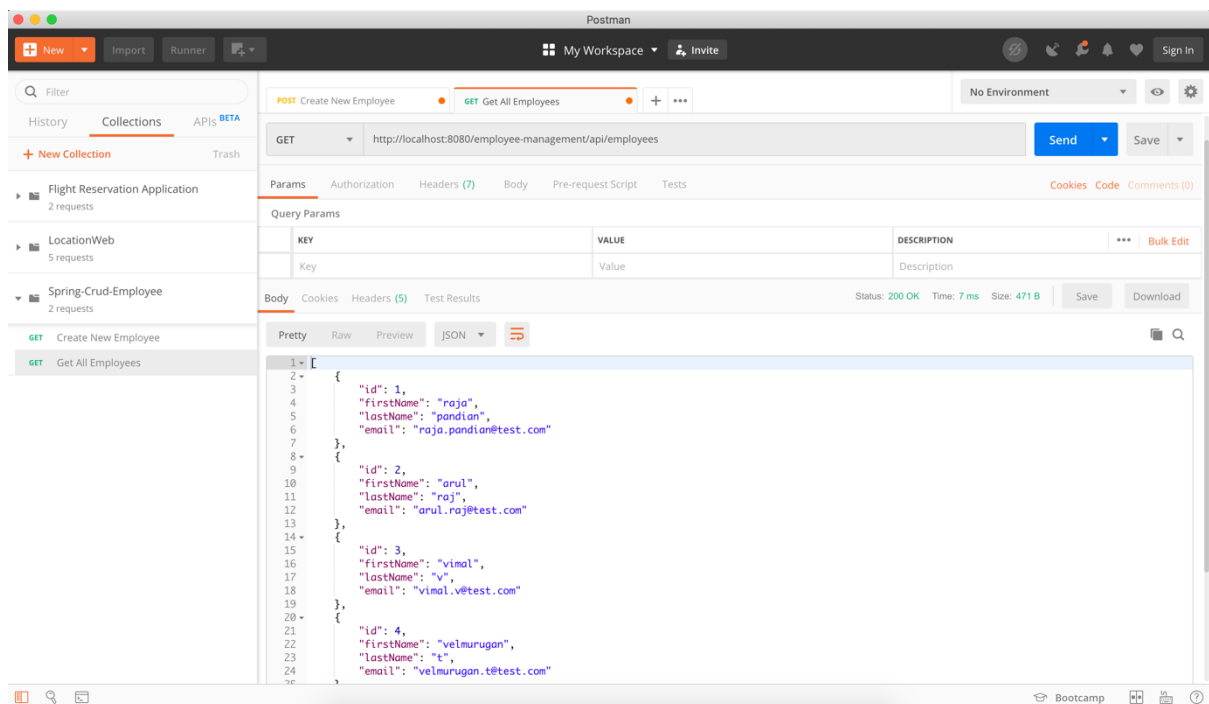
POST Create New Employee
Create New Employee
POST http://localhost:8080/employee-management/api/employees
Body
1 - {
2   "firstName": "raja",
3   "lastName": "pandian",
4   "email": "raja.pandian@test.com"
5 }
Body
1 - {
2   "id": 1,
3   "firstName": "raja",
4   "lastName": "pandian",
5   "email": "raja.pandian@test.com"
6 }
Status: 200 OK Time: 135 ms Size: 244 B
  
```

We can now check from H2 console if the new employee is saved.

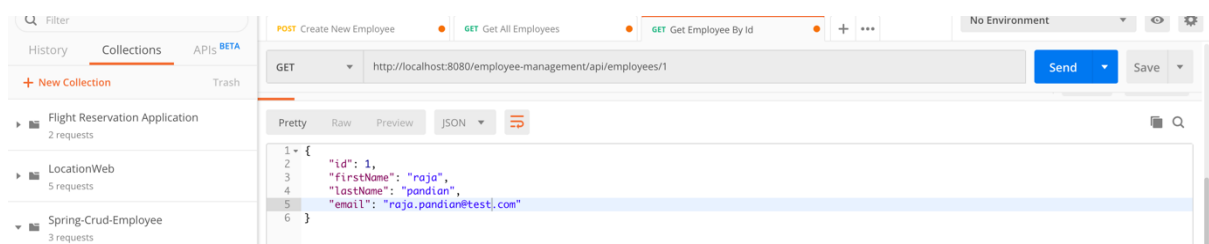


We can now test the other endpoints.

Get All Employees:



Get Employee By Id:



Update Employee:

The screenshot shows the Postman interface for a PUT request to `http://localhost:8080/employee-management/api/employees/1`. The request body is a JSON object:

```
{
  "id": 1,
  "firstName": "raja",
  "lastName": "pandian",
  "email": "raja.pandian@gmail.com"
}
```

The response status is 200 OK, with a time of 6 ms and a size of 245 B. The response body is a JSON object:

```
{
  "id": 1,
  "firstName": "raja",
  "lastName": "pandian",
  "email": "raja.pandian@gmail.com"
}
```

Delete Employee By Id:

The screenshot shows the Postman interface for a DELETE request to `http://localhost:8080/employee-management/api/employees/1`. The response status is 200 OK, with a time of 21 ms and a size of 123 B. The response body is empty.

The screenshot shows a database client interface with the following SQL query and result:

```
SELECT * FROM EMPLOYEE;
```

ID	EMAIL	FIRST_NAME	LAST_NAME
2	arul.raj@test.com	arul	raj
3	vimal.v@test.com	vimal	v
4	velmurugan.t@test.com	velmurugan	t

(3 rows, 2 ms)

This validates that our ***“Bootiful Spring Data JPA Application”*** is working as desired.

*** END ***