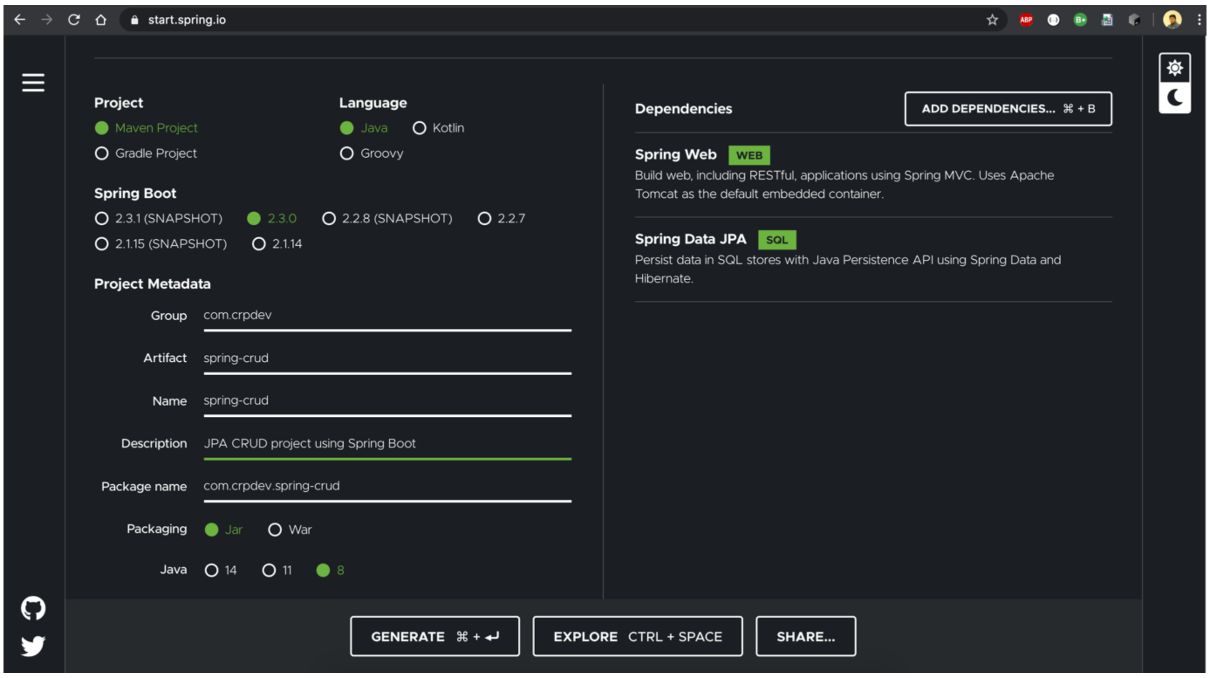
Spring REST CRUD Demo with JPA Configuration

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]



Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
2.3.1 (SNAPSHOT) [✓ 2.3.0] 2.2.8 (SNAPSHOT) 2.2.7 2.1.15 (SNAPSHOT) 2.1.14

Project Metadata

Group	com.crpdev
Artifact	spring-crud
Name	spring-crud
Description	JPA CRUD project using Spring Boot
Package name	com.crpdev.spring-crud
Packaging	[✓ Jar] War
Java	14 11 [✓ 8]

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.

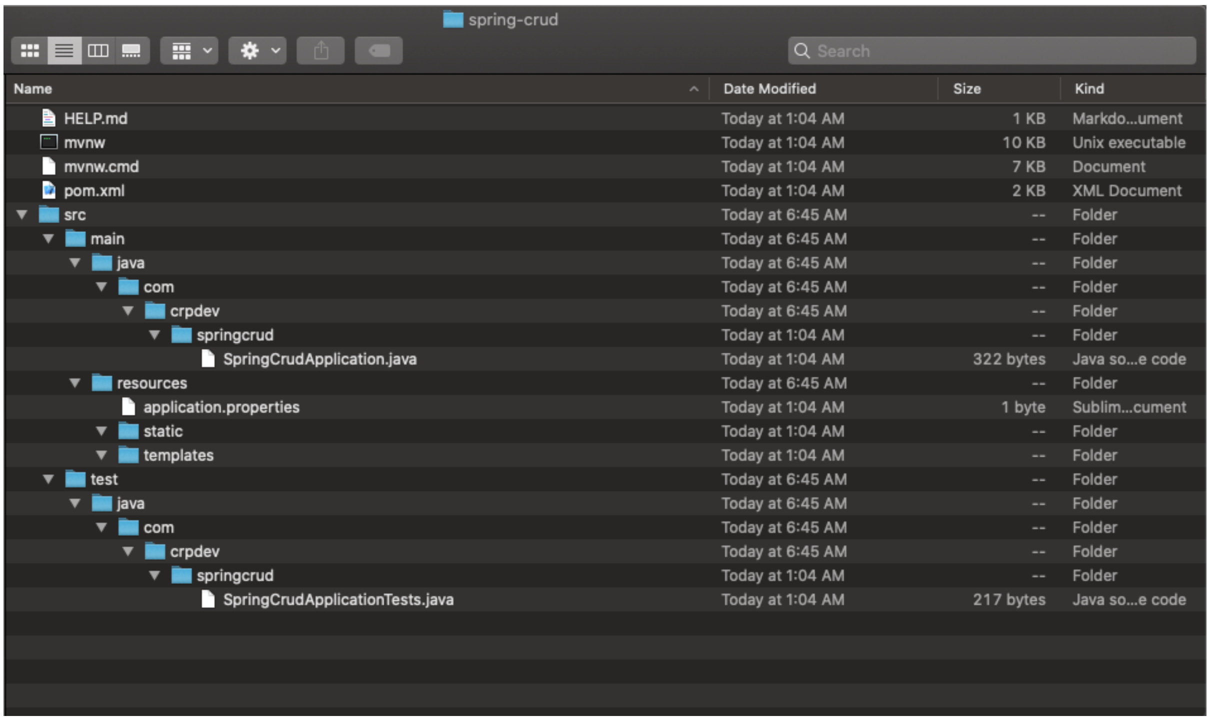
Our motive is to implement a JPA CRUD functionality using Spring Boot. So all that is needed now are:

- 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

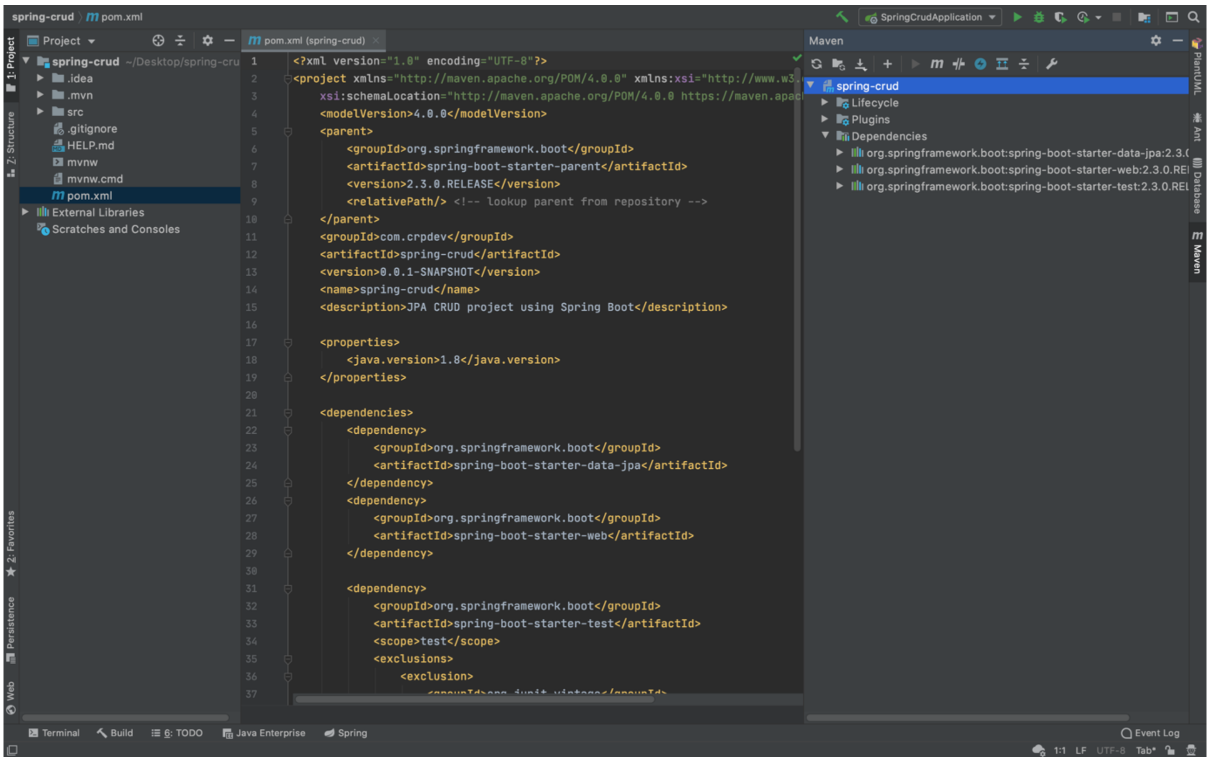
Extract the downloaded project zip



spring-crud

Name	Date Modified	Size	Kind
HELP.md	Today at 1:04 AM	1 KB	Markdo...ument
mvnw	Today at 1:04 AM	10 KB	Unix executable
mvnw.cmd	Today at 1:04 AM	7 KB	Document
pom.xml	Today at 1:04 AM	2 KB	XML Document
src	Today at 6:45 AM	--	Folder
main	Today at 6:45 AM	--	Folder
java	Today at 6:45 AM	--	Folder
com	Today at 6:45 AM	--	Folder
crpdev	Today at 6:45 AM	--	Folder
springcrud	Today at 1:04 AM	--	Folder
SpringCrudApplication.java	Today at 1:04 AM	322 bytes	Java so...e code
resources	Today at 6:45 AM	--	Folder
application.properties	Today at 1:04 AM	1 byte	Sublim...cument
static	Today at 1:04 AM	--	Folder
templates	Today at 1:04 AM	--	Folder
test	Today at 6:45 AM	--	Folder
java	Today at 6:45 AM	--	Folder
com	Today at 6:45 AM	--	Folder
crpdev	Today at 6:45 AM	--	Folder
springcrud	Today at 1:04 AM	--	Folder
SpringCrudApplicationTests.java	Today at 1:04 AM	217 bytes	Java so...e code

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.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apac
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.crpdev</groupId>
    <artifactId>spring-crud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-crud</name>
    <description>JPA CRUD project using Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
```

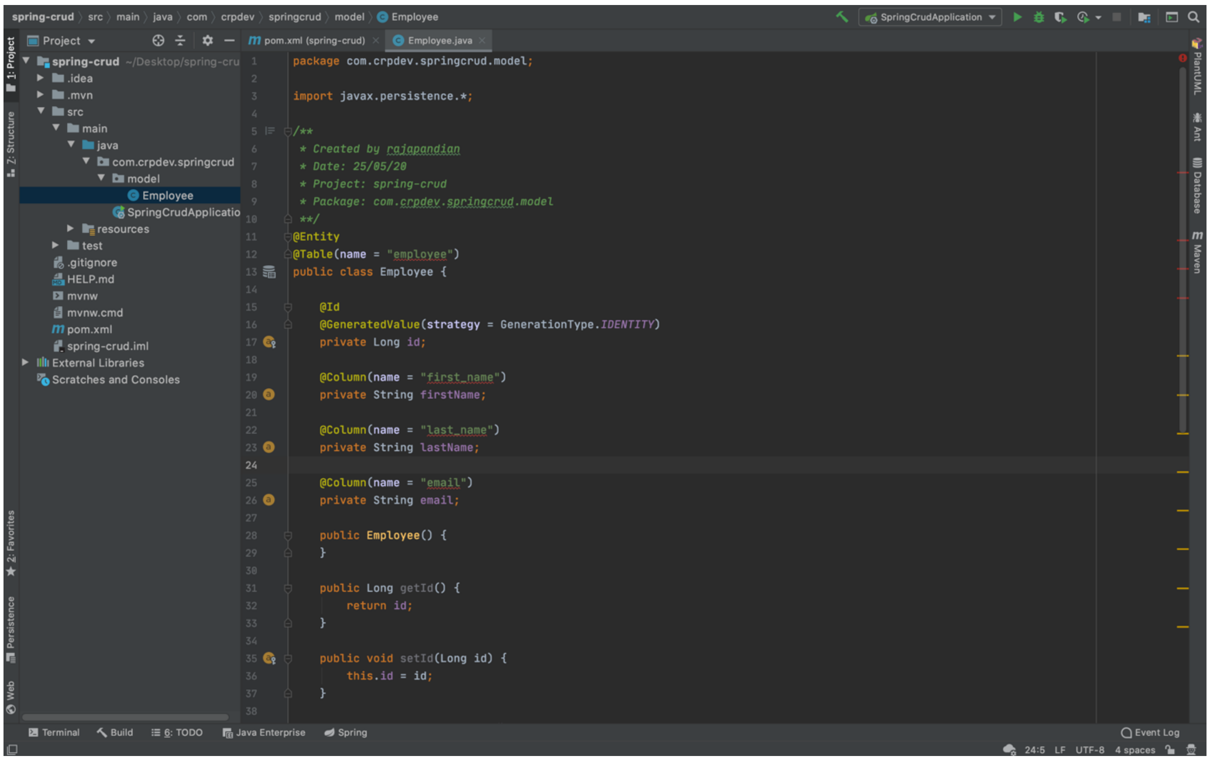
3. Entity class

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**”.

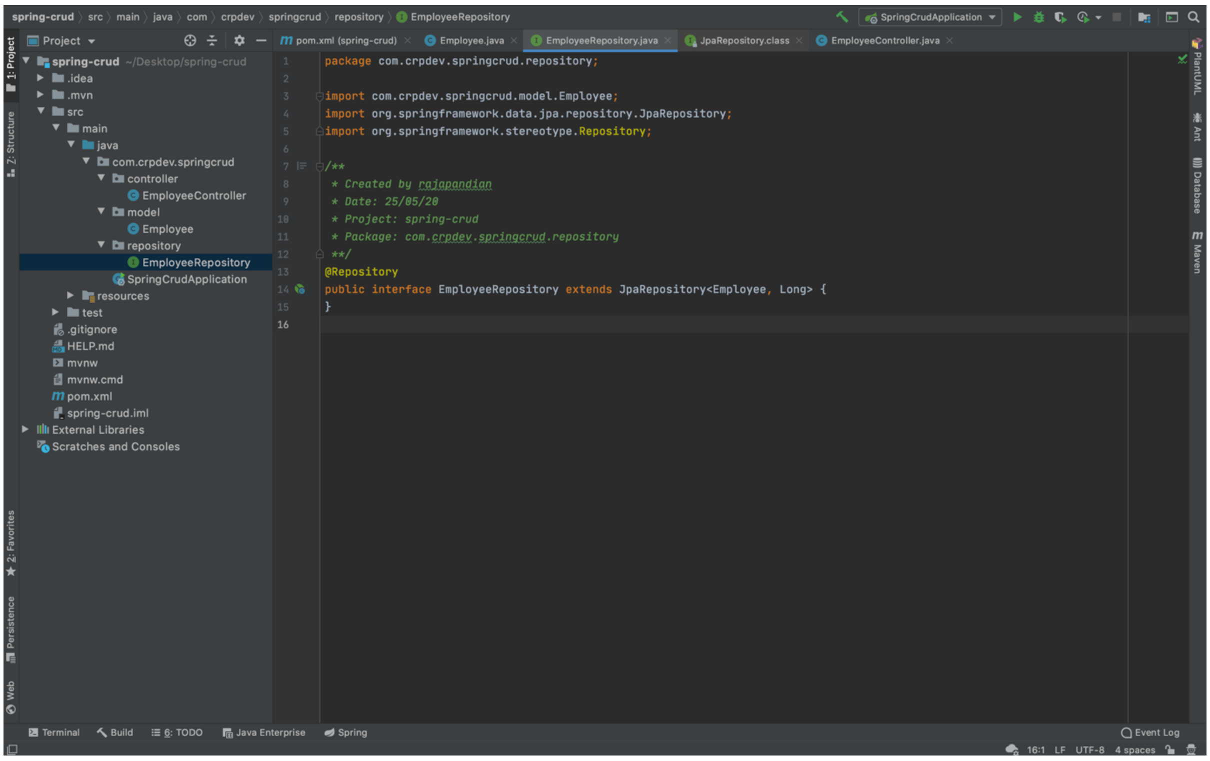


```java
package com.crpdev.springcrud.model;

import javax.persistence.*;

/**
 * Created by rajapandian
 * Date: 25/05/20
 * Project: spring-crud
 * Package: com.crpdev.springcrud.model
 **/
@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    public Employee() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

```

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.



```java
package com.crpdev.springcrud.repository;

import com.crpdev.springcrud.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

/**
 * Created by rajapandian
 * Date: 25/05/20
 * Project: spring-crud
 * Package: com.crpdev.springcrud.repository
 **/
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

We have created **JpaRepository** implementation as **EmployeeRepository** which provides all defaults 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.

<u>Note:</u>

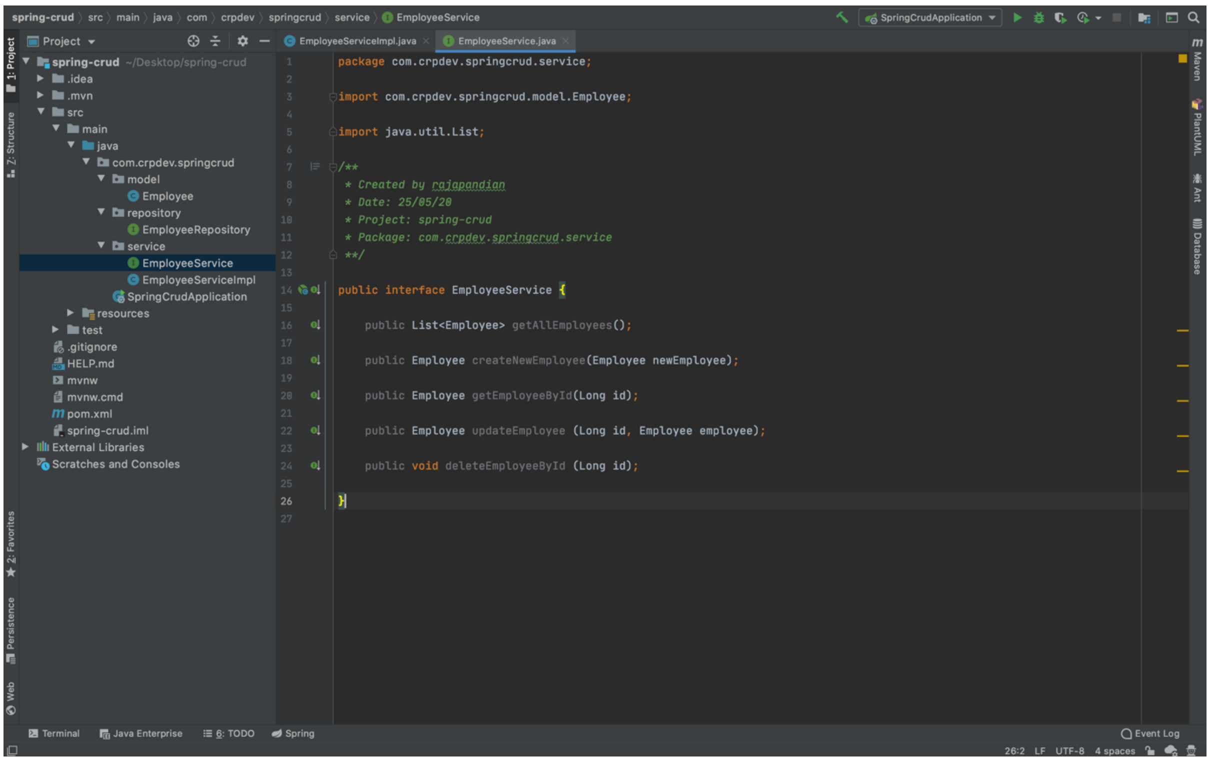
Spring Data JPA provides CRUDRepository Interface and a JpaRepository Interface.

- **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.



```java
package com.crpdev.springcrud.service;

import com.crpdev.springcrud.model.Employee;

import java.util.List;

/**
 * Created by rajapandian
 * Date: 25/05/20
 * Project: spring-crud
 * Package: com.crpdev.springcrud.service
 **/

public interface EmployeeService {

    public List<Employee> getAllEmployees();

    public Employee createNewEmployee(Employee newEmployee);

    public Employee getEmployeeById(Long id);

    public Employee updateEmployee (Long id, Employee employee);

    public void deleteEmployeeById (Long id);

}
```

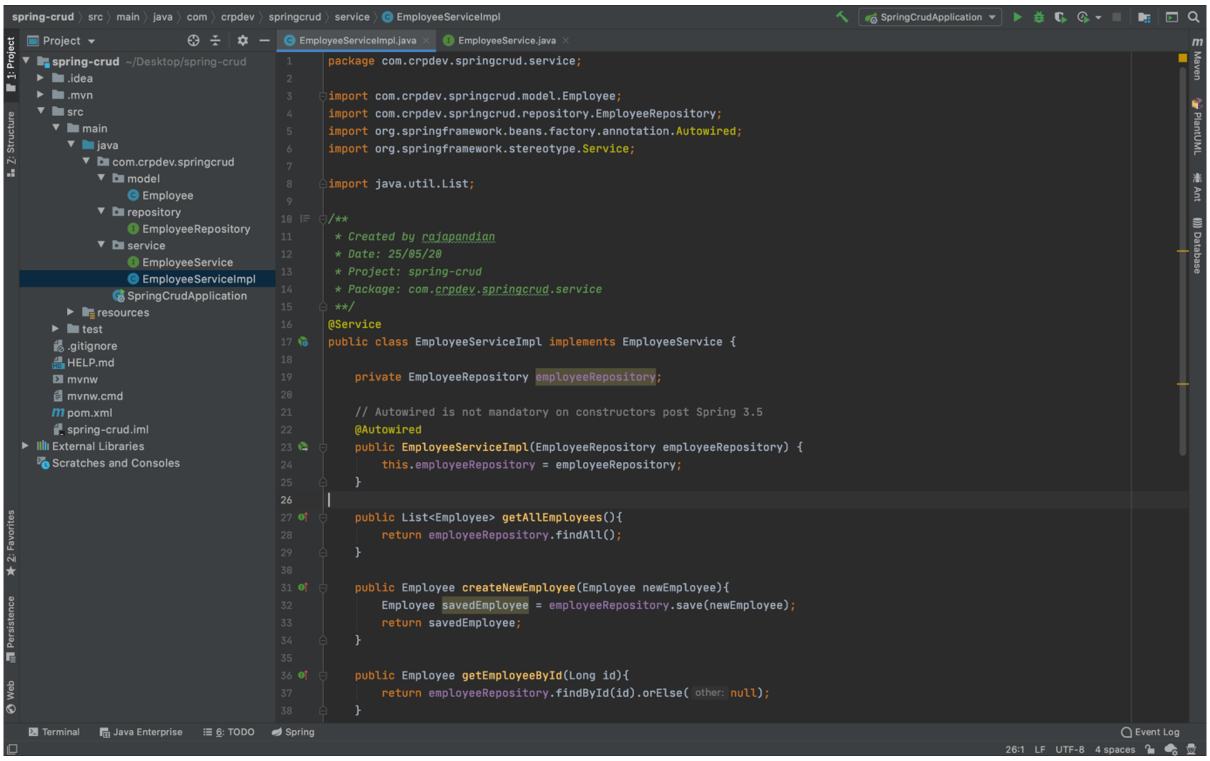
We will now create an implementation class “**EmployeeServiceImpl**” that will implement the “**EmployeeService**” Interface.

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

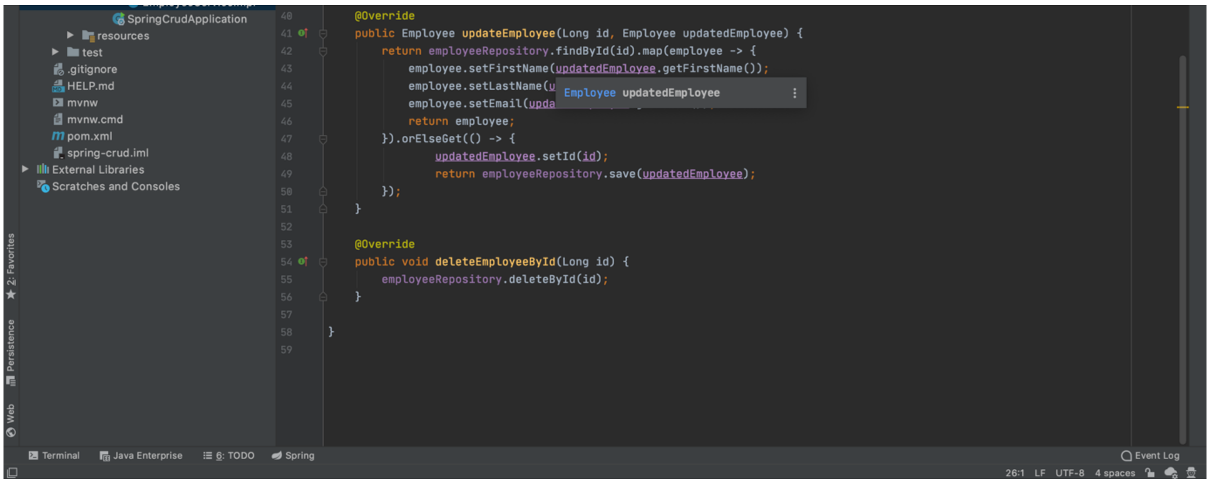
The reason for “**Programming To An Interface**” is that EmployeeService can take any form at runtime. In this case, we will be using our **EmployeeRepository** to perform CRUD operations. At runtime, we might have to perform CRUD using an external service. Hence, this approach will help us define an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



```java
package com.crpdev.springcrud.service;

import com.crpdev.springcrud.model.Employee;
import com.crpdev.springcrud.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * Created by rajapandian
 * Date: 25/05/20
 * Project: spring-crud
 * Package: com.crpdev.springcrud.service
 **/
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    // Autowired is not mandatory on constructors post Spring 3.5
    @Autowired
    public EmployeeServiceImpl(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    public List<Employee> getAllEmployees(){
        return employeeRepository.findAll();
    }

    public Employee createNewEmployee(Employee newEmployee){
        Employee savedEmployee = employeeRepository.save(newEmployee);
        return savedEmployee;
    }

    public Employee getEmployeeById(Long id){
        return employeeRepository.findById(id).orElse( other: null);
    }
```



```java
    @Override
    public Employee updateEmployee(Long id, Employee updatedEmployee) {
        return employeeRepository.findById(id).map(employee -> {
            employee.setFirstName(updatedEmployee.getFirstName());
            employee.setLastName(u
            employee.setEmail(upda
            return employee;
        }).orElseGet(() -> {
                updatedEmployee.setId(id);
                return employeeRepository.save(updatedEmployee);
        });
    }

    @Override
    public void deleteEmployeeById(Long id) {
        employeeRepository.deleteById(id);
    }

}
```

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

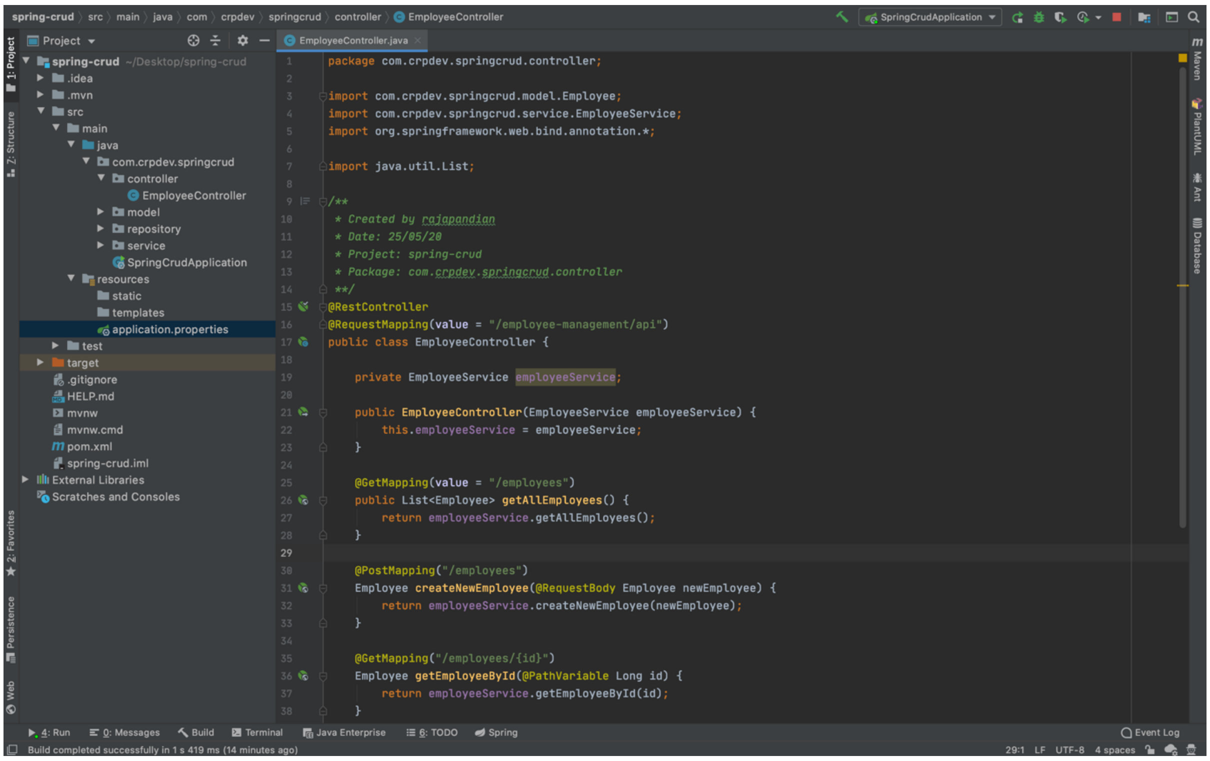
6. REST Controller

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

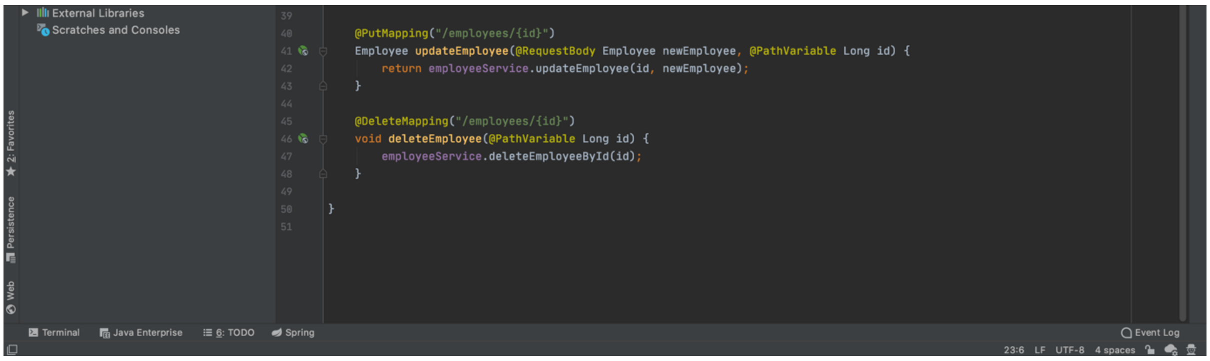
Annotate the Interface with **@RestController stereotype.** This will help the Spring Context inject this dependency at start-up.

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.



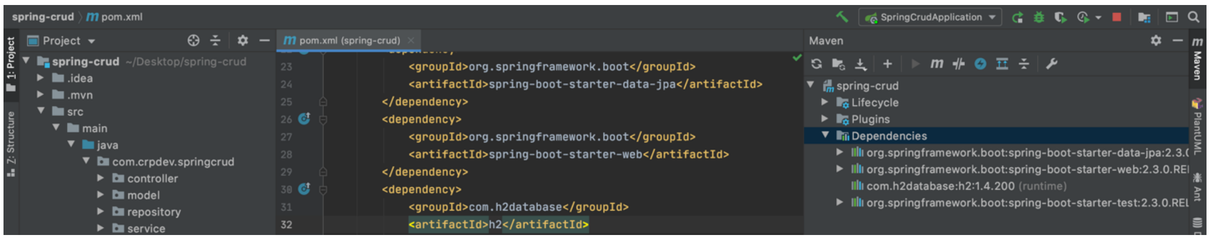
```java
package com.crpdev.springcrud.controller;

import com.crpdev.springcrud.model.Employee;
import com.crpdev.springcrud.service.EmployeeService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * Created by rajapandian
 * Date: 25/05/20
 * Project: spring-crud
 * Package: com.crpdev.springcrud.controller
 **/
@RestController
@RequestMapping(value = "/employee-management/api")
public class EmployeeController {

    private EmployeeService employeeService;

    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    @GetMapping(value = "/employees")
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @PostMapping("/employees")
    Employee createNewEmployee(@RequestBody Employee newEmployee) {
        return employeeService.createNewEmployee(newEmployee);
    }

    @GetMapping("/employees/{id}")
    Employee getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }
```



```java

    @PutMapping("/employees/{id}")
    Employee updateEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {
        return employeeService.updateEmployee(id, newEmployee);
    }

    @DeleteMapping("/employees/{id}")
    void deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployeeById(id);
    }

}
```

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.



```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
```

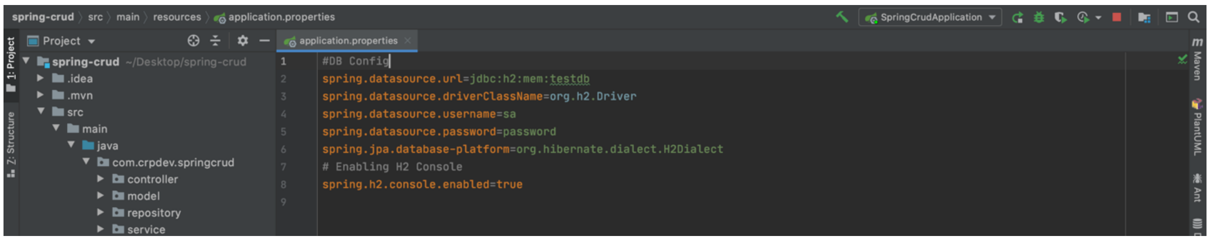
```
spring-crud
  Lifecycle
  Plugins
  Dependencies
    org.springframework.boot:spring-boot-starter-data-jpa:2.3.(
    org.springframework.boot:spring-boot-starter-web:2.3.0.REI
    com.h2database:h2:1.4.200 (runtime)
    org.springframework.boot:spring-boot-starter-test:2.3.0.REI
```

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.



```
#DB Config
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# Enabling H2 Console
spring.h2.console.enabled=true
```

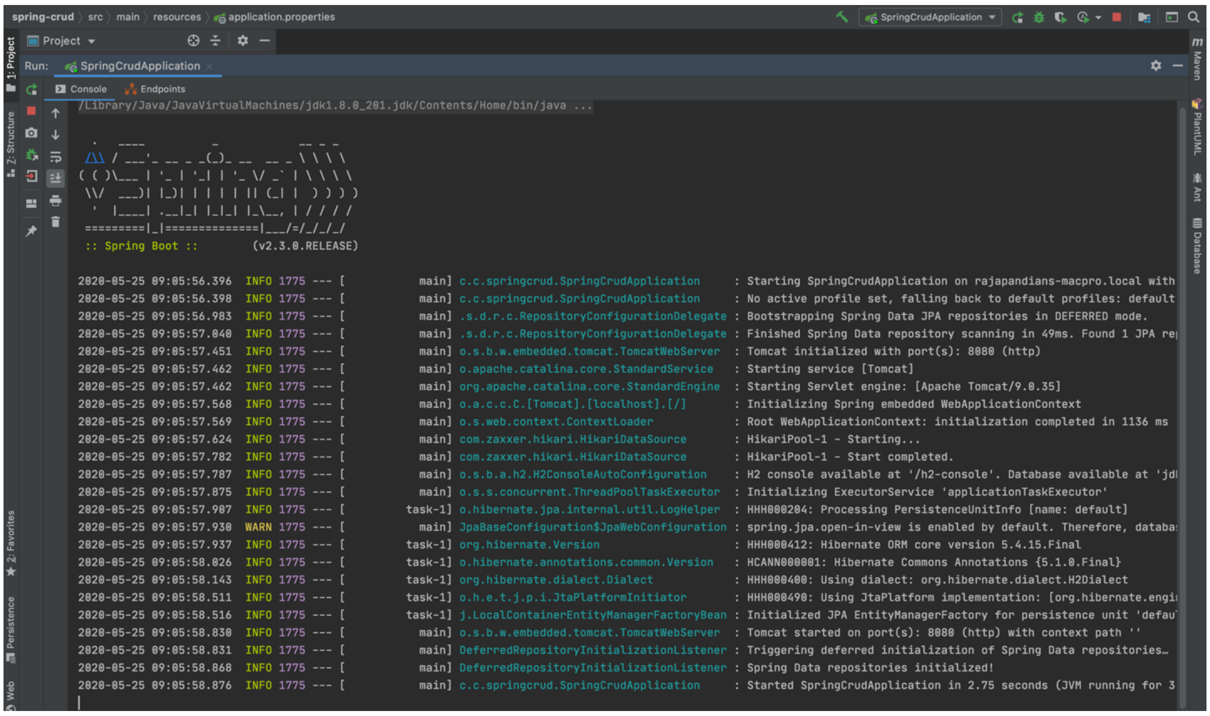
The first set of parameters help Spring Boot to configure the application to use H2 In-Memory DB.

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.

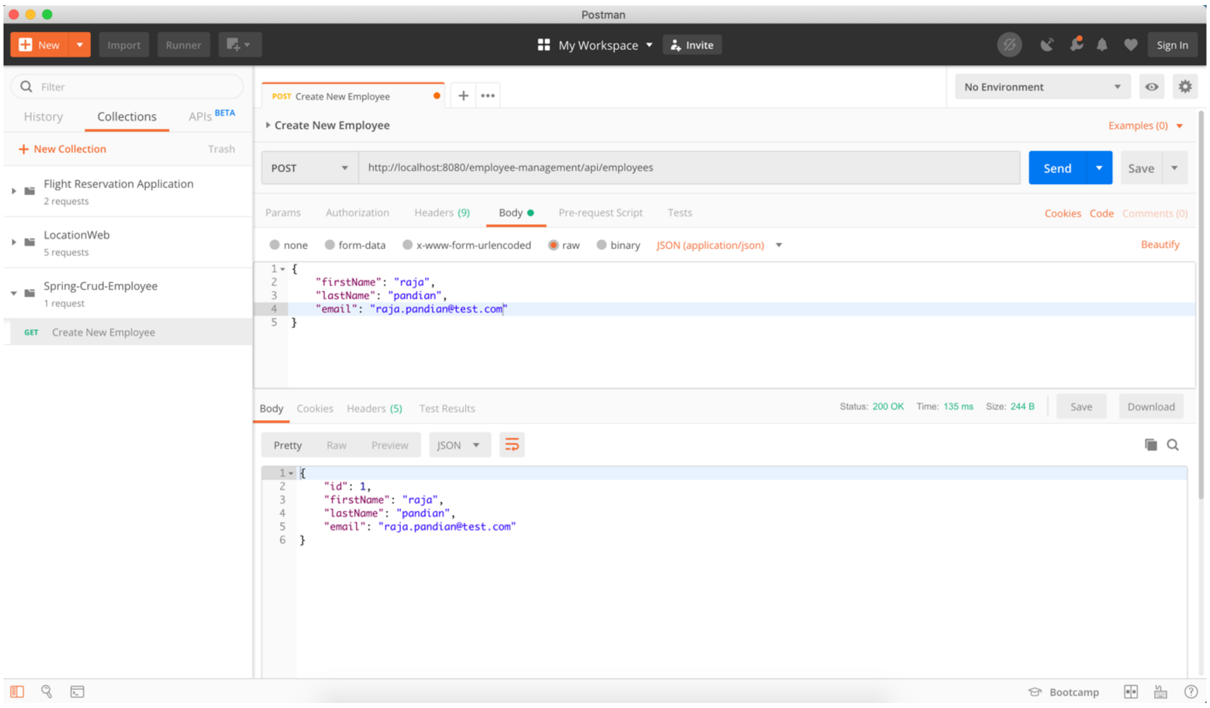
8. Spring REST CRUD Operations Demo

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



```
/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 'jd
2020-05-25 09:05:57.875  INFO 1775 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-05-25 09:05:57.907  INFO 1775 --- [         task-1] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-05-25 09:05:57.930  WARN 1775 --- [           main] 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.engi
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:



POST http://localhost:8080/employee-management/api/employees

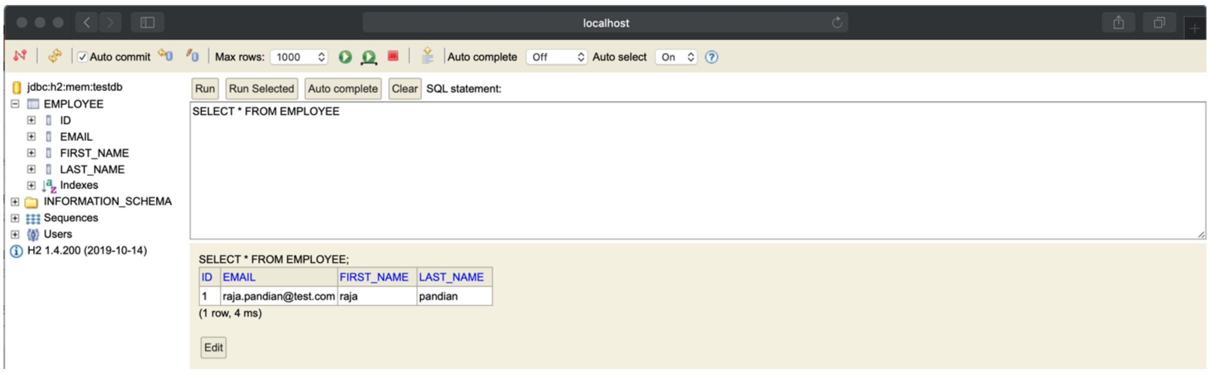
Body (raw, JSON (application/json)):

```
{
    "firstName": "raja",
    "lastName": "pandian",
    "email": "raja.pandian@test.com"
}
```

Status: 200 OK Time: 135 ms Size: 244 B

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

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



```
SELECT * FROM EMPLOYEE
```

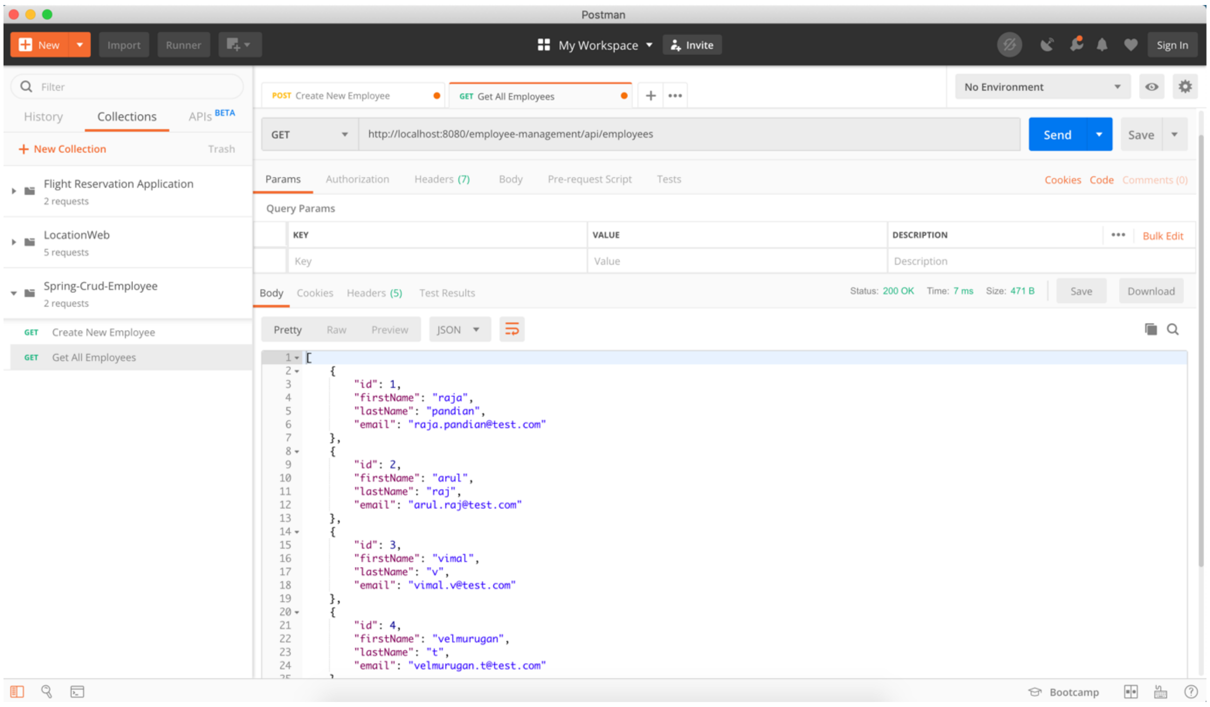
SELECT * FROM EMPLOYEE;

ID	EMAIL	FIRST_NAME	LAST_NAME
1	raja.pandian@test.com	raja	pandian

(1 row, 4 ms)

We can now test the other endpoints.

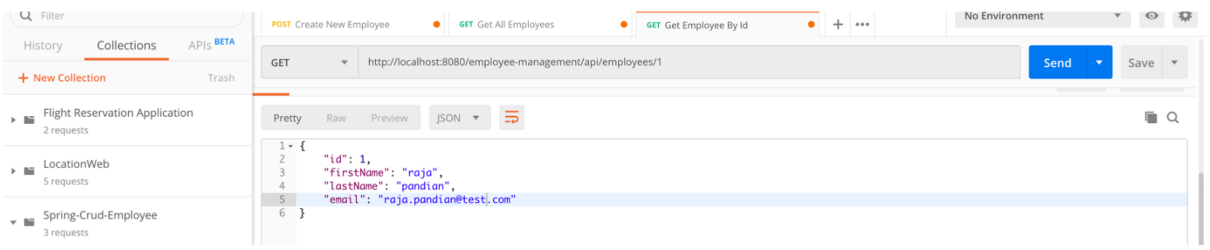
Get All Employees:



GET http://localhost:8080/employee-management/api/employees

```
[
    {
        "id": 1,
        "firstName": "raja",
        "lastName": "pandian",
        "email": "raja.pandian@test.com"
    },
    {
        "id": 2,
        "firstName": "arul",
        "lastName": "raj",
        "email": "arul.raj@test.com"
    },
    {
        "id": 3,
        "firstName": "vimal",
        "lastName": "v",
        "email": "vimal.v@test.com"
    },
    {
        "id": 4,
        "firstName": "velmurugan",
        "lastName": "t",
        "email": "velmurugan.t@test.com"
```

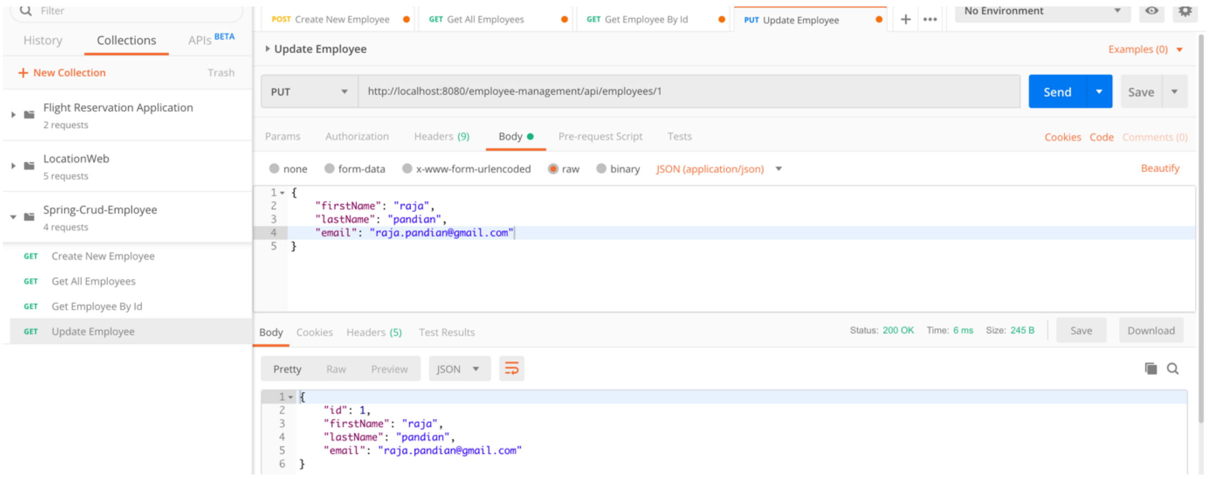
Get Employee By Id:



GET http://localhost:8080/employee-management/api/employees/1

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

Update Employee:



Update Employee

PUT http://localhost:8080/employee-management/api/employees/1

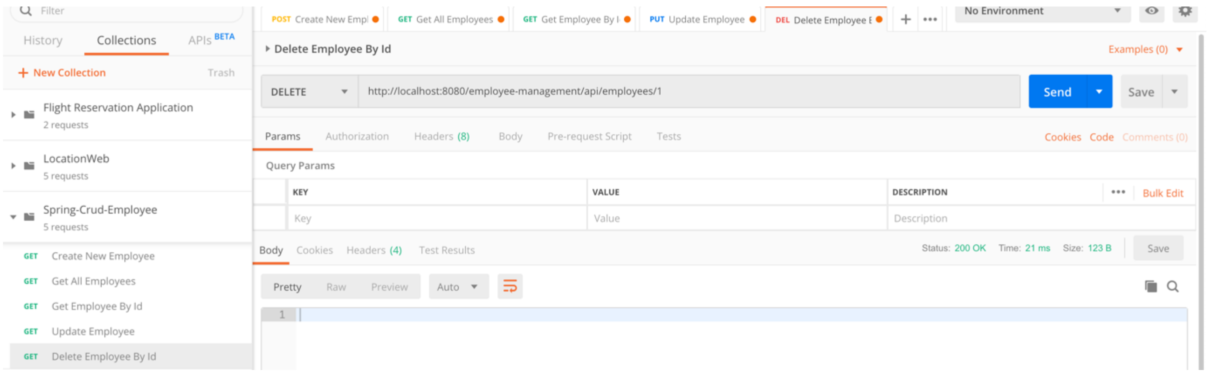
Body: raw JSON (application/json)

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

Status: 200 OK Time: 6 ms Size: 245 B

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

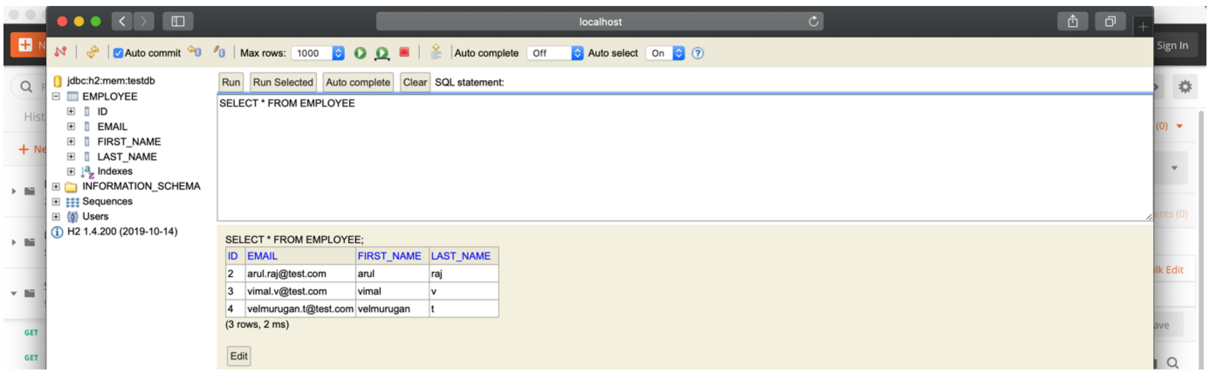
Delete Employee By Id:



Delete Employee By Id

DELETE http://localhost:8080/employee-management/api/employees/1

Status: 200 OK Time: 21 ms Size: 123 B



jdbc:h2:mem:testdb

SQL statement:

```
SELECT * FROM EMPLOYEE
```

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 ***