**Lab 03 - Building RESTful Web Services with Spring Boot**

# 1. Introduction

Imagine you're a new fresher Spring Boot developer of IT department in a University. Your manager has asked you to develop a RESTful API for Employee Management to manage just a simple information of employee.

Employee (ID, Name, Designation, Salary)

The RESTful API has to support adding, viewing, modifying, and removing information for an employee - a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating a RESTful API using Spring Boot 3 with Data Source in Memory. A Data Source **(**Collection**)** will be created to persist the employee data.

# 2. Lab Objectives

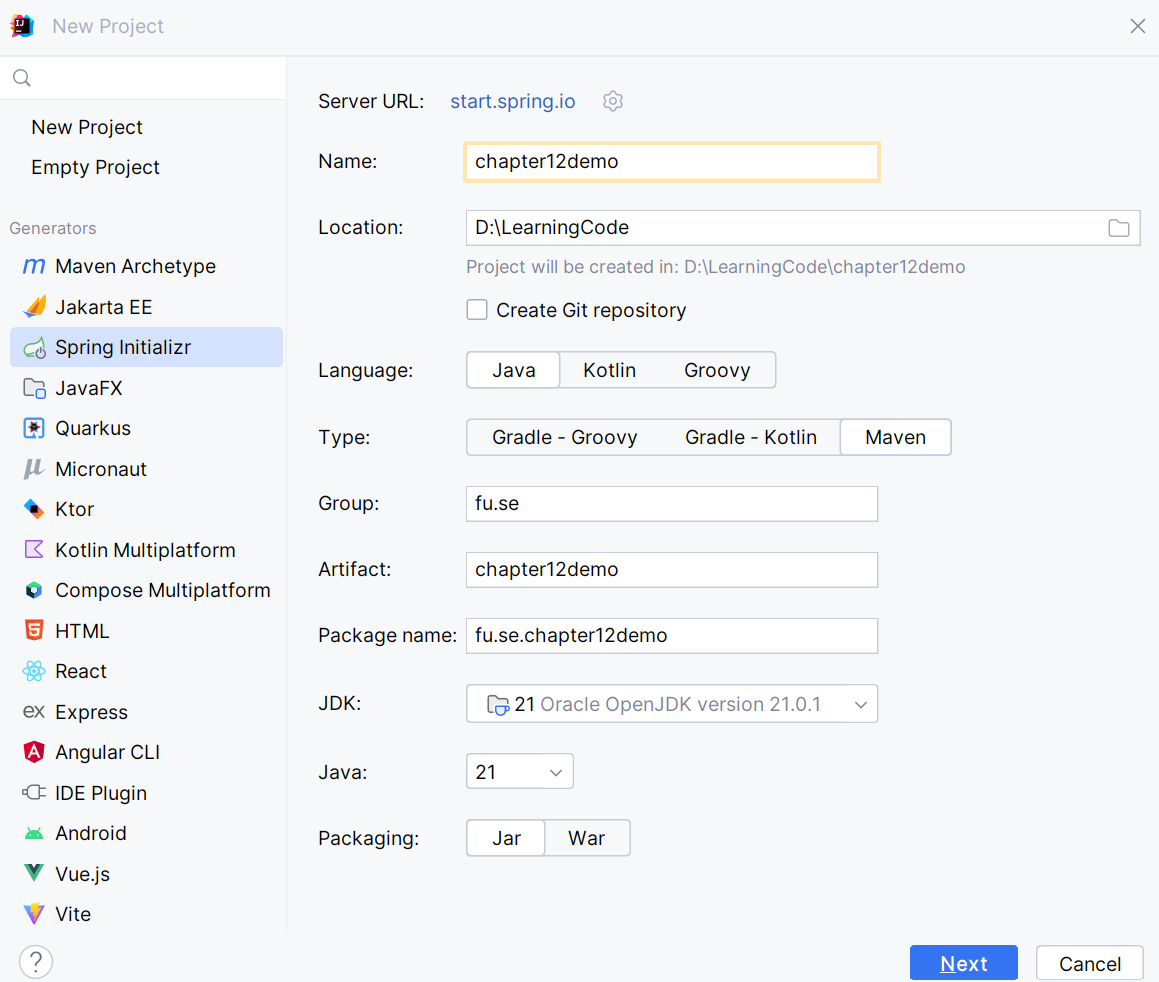
In this lab, you will:

* Use the **Intellij** **IDEA** to create Spring Initializr Project (Maybe you can create the Spring Initializr project in <https://start.spring.io/> then open this project in IntelliJ IDEA.
* Choose **Spring Web** for Building web, including *RESTful*, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
* Use in Memory Database as a Collection of employees
* Develop 3-Layer with Repository Architecture to perform CRUD actions using RESTful API with Spring Boot 3.
* Run the project and test the application actions.

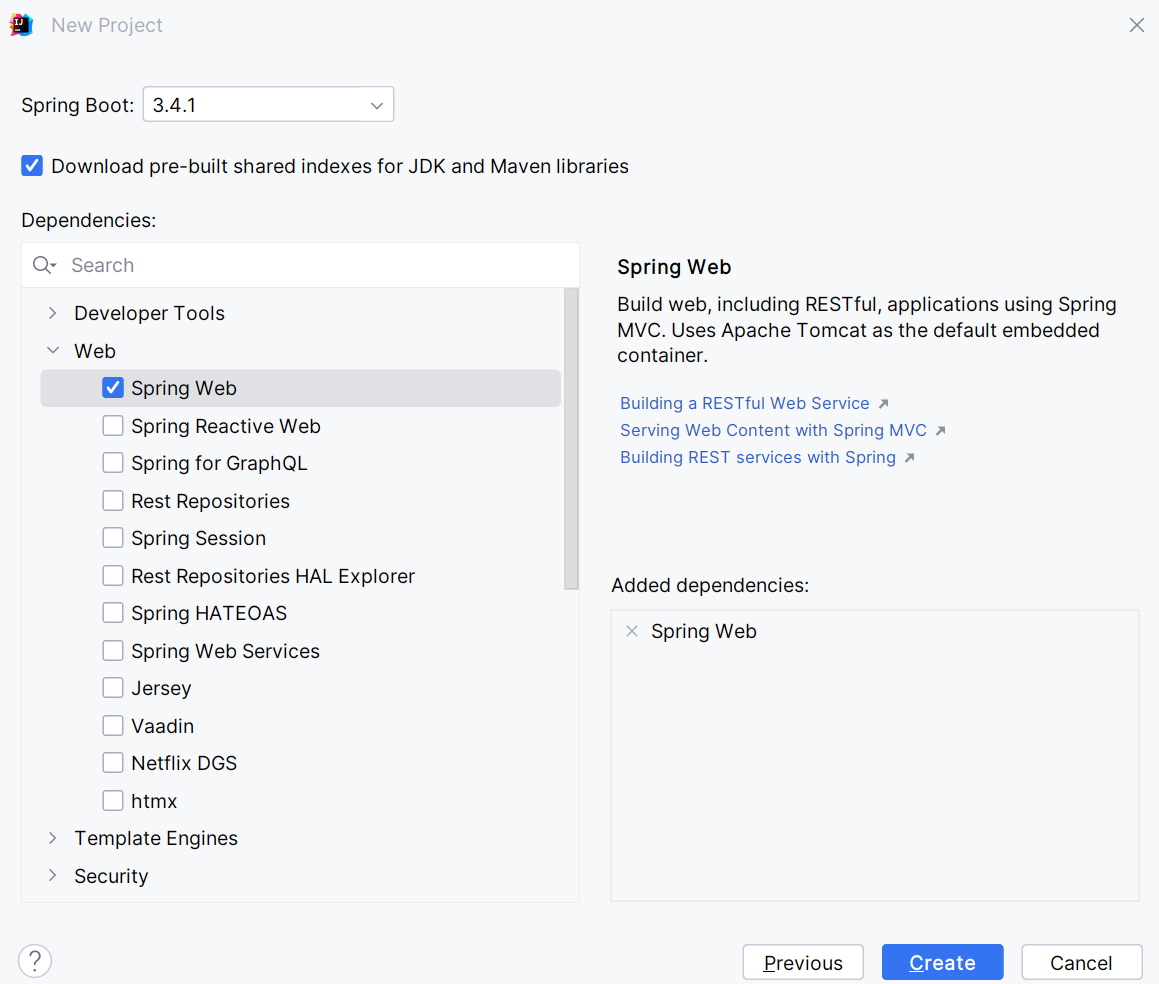
# Activity 01: Design the Employee Management

**Step 01**. Open IntelliJ IDEA, File | New | Initializr Project

In the left pane of the New Project wizard, select Spring Initializr Project.

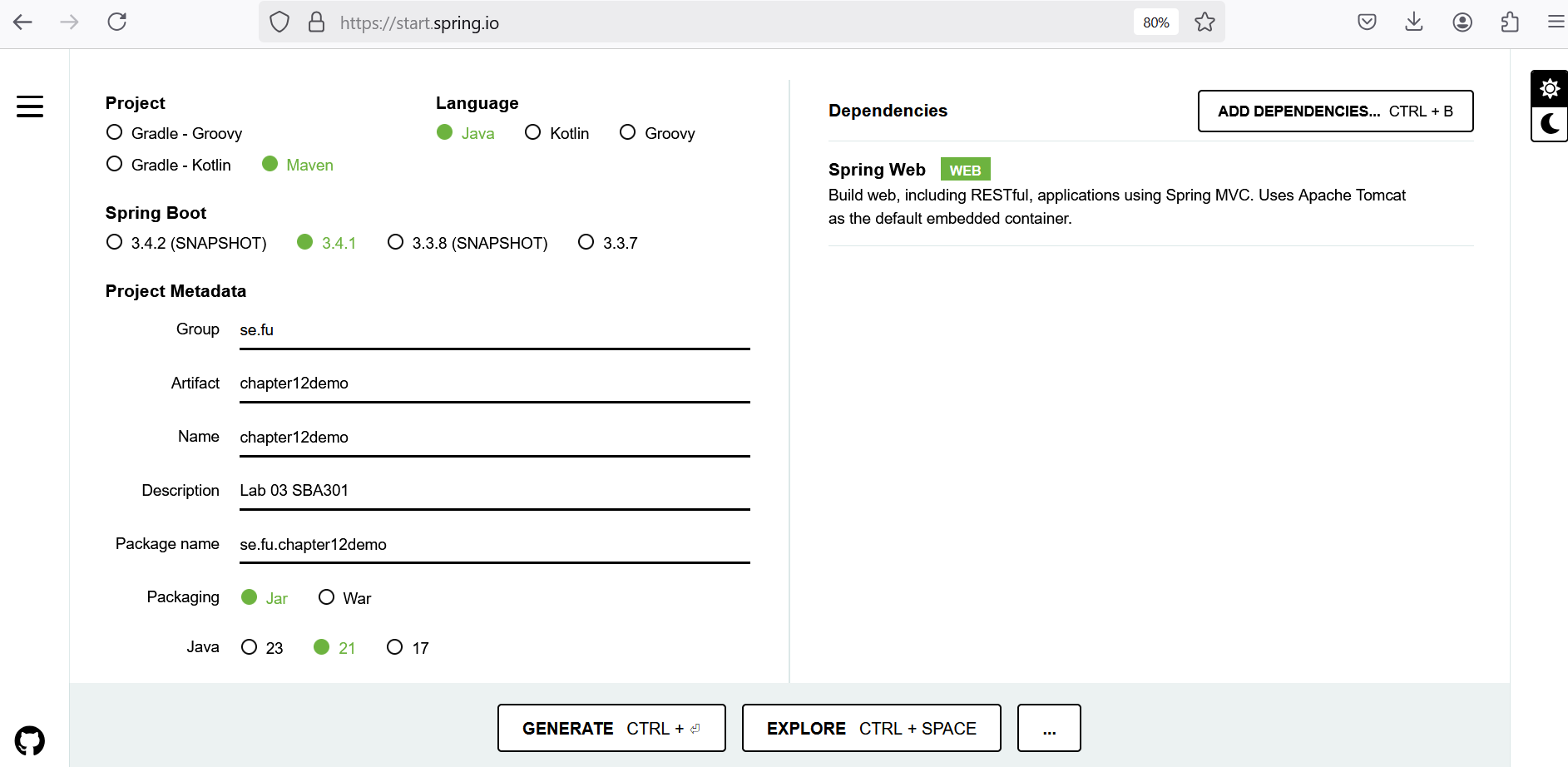


**Step 02.** Fill the information → Next



**Step 03**. Choose the dependency/dependencies → Click Create

Or you can using Spring Initializr project in <https://start.spring.io/> then open this project in IntelliJ IDEA

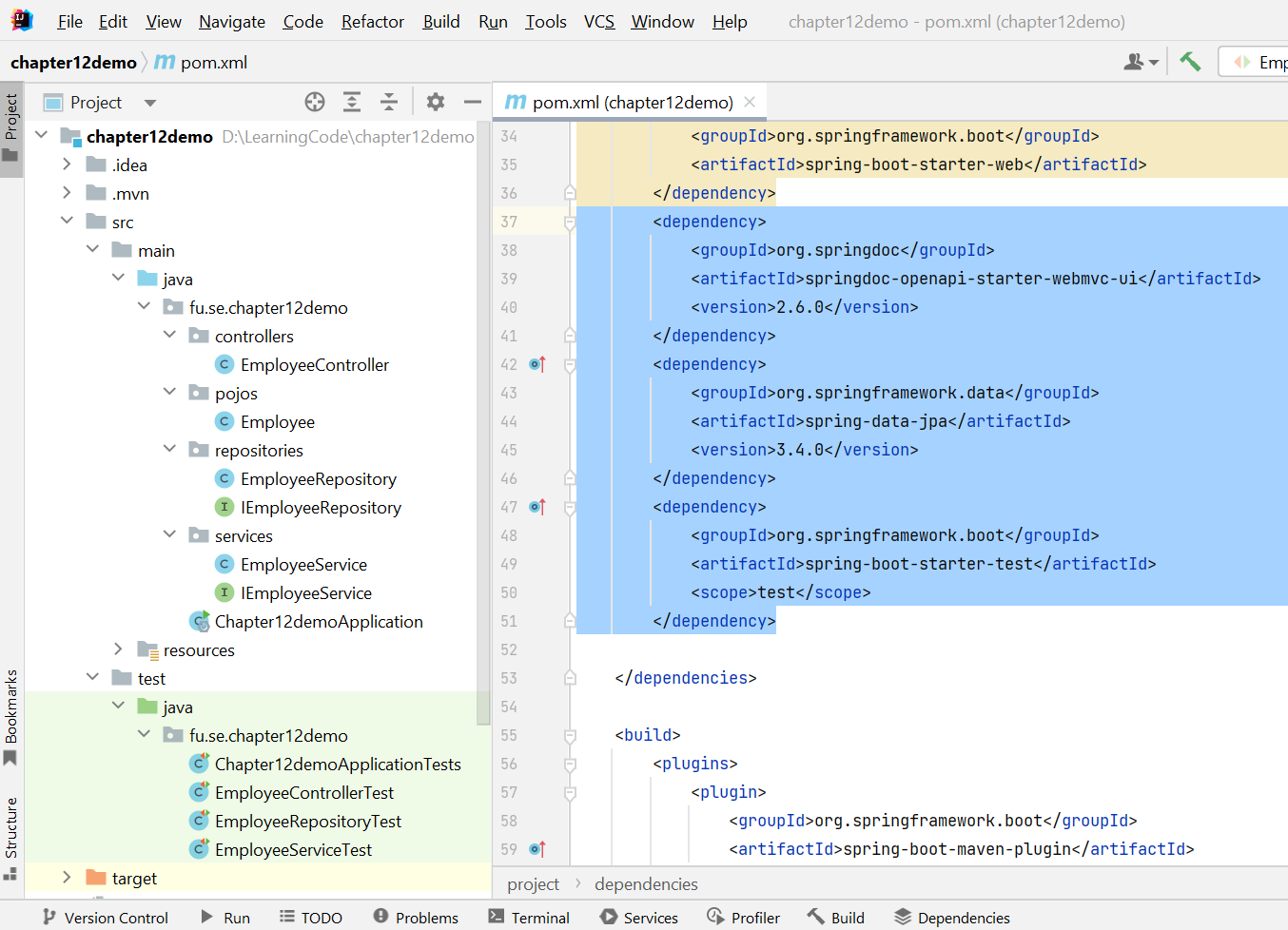


**Step 04**. Edit pom.xml, add the dependencies for

* Swagger Open API **springdoc-openapi-starter-webmvc-ui**
* Spring Data JPA for Paging and Sort data: **spring-data-jpa**
* Testing **spring-boot-starter-test**

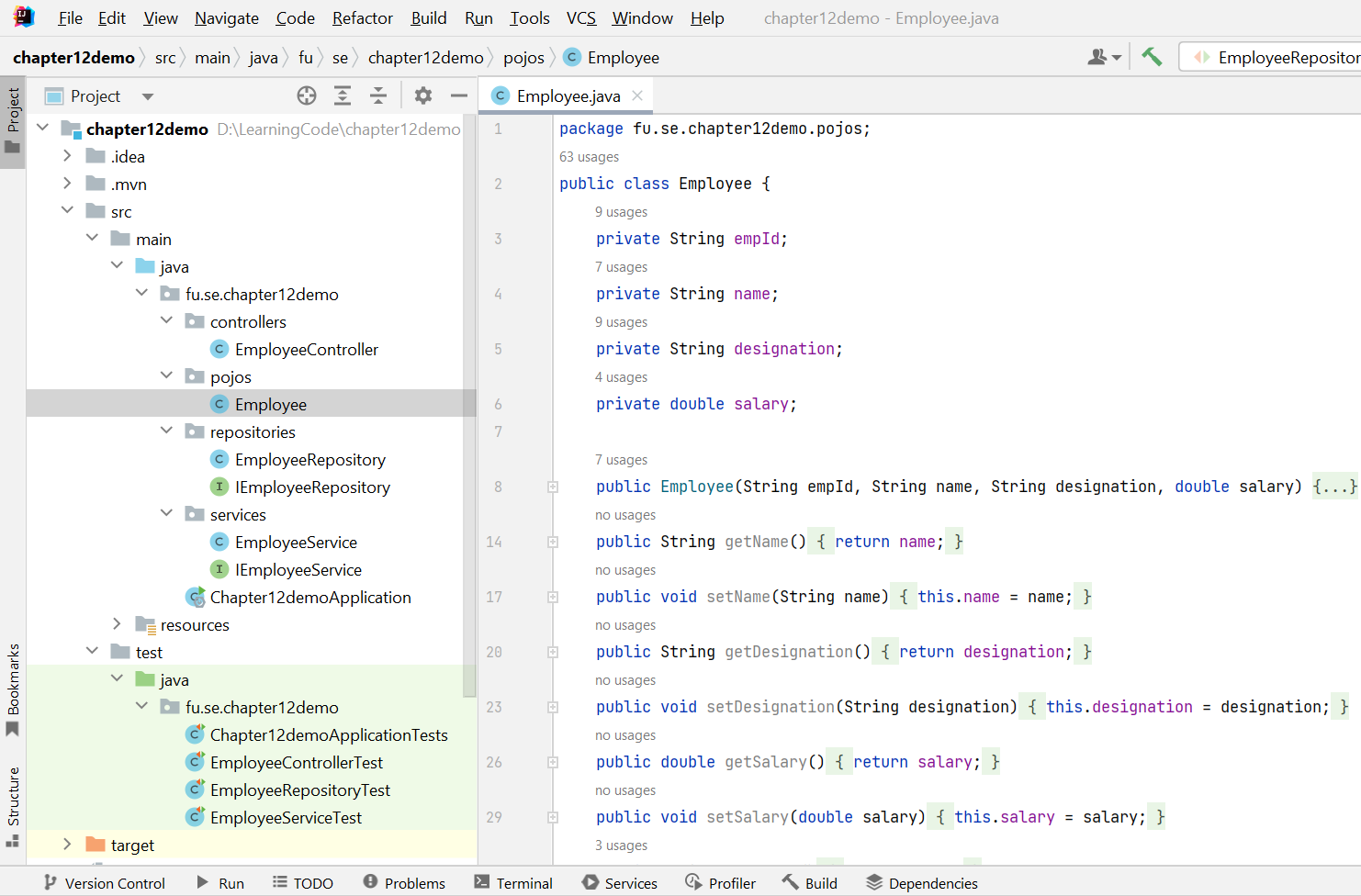
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>**spring-boot-starter-web**</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>**springdoc-openapi-starter-webmvc-ui**</artifactId>  
 <version>2.6.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>**spring-data-jpa**</artifactId>  
 <version>3.4.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>**spring-boot-starter-test**</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
</dependencies>

**Step 05**. The structure of the Lab project



# Activity 02: Implement RESTful API Employee Management with CRUD functions

**Step 01**. Create **pojos** package, then create Employee Entity

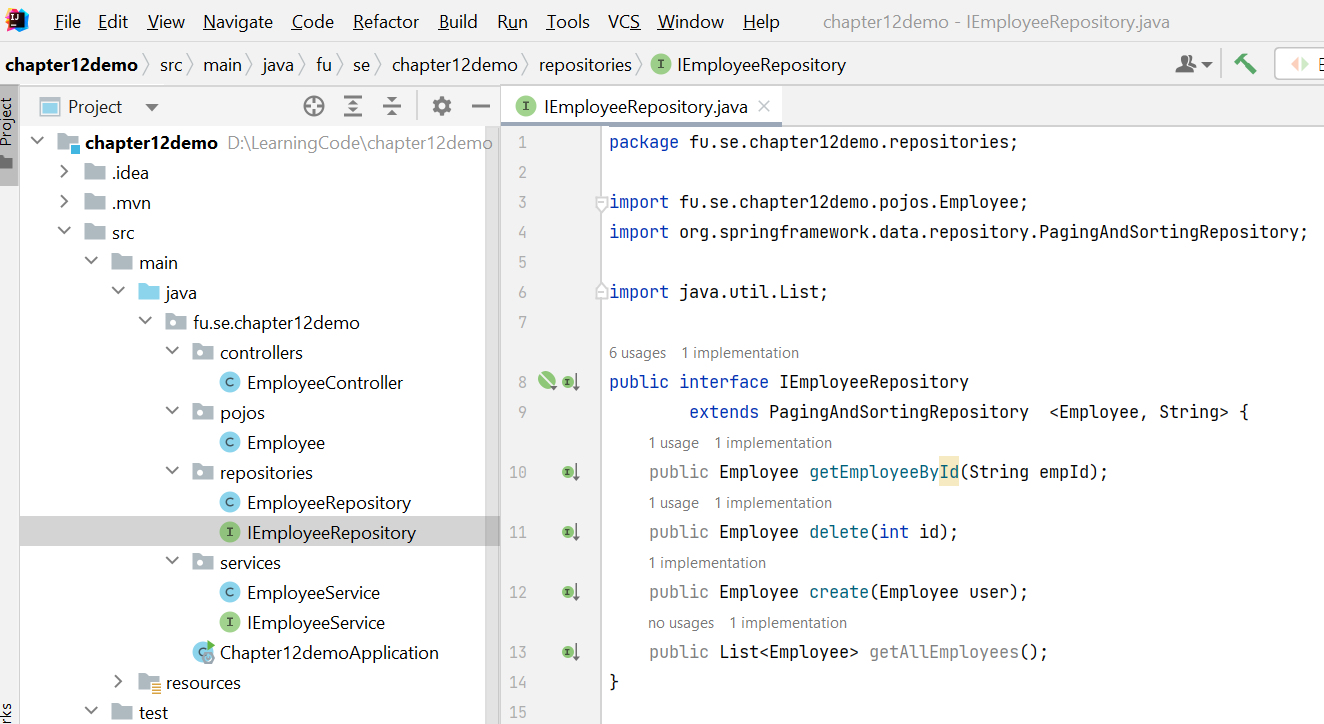


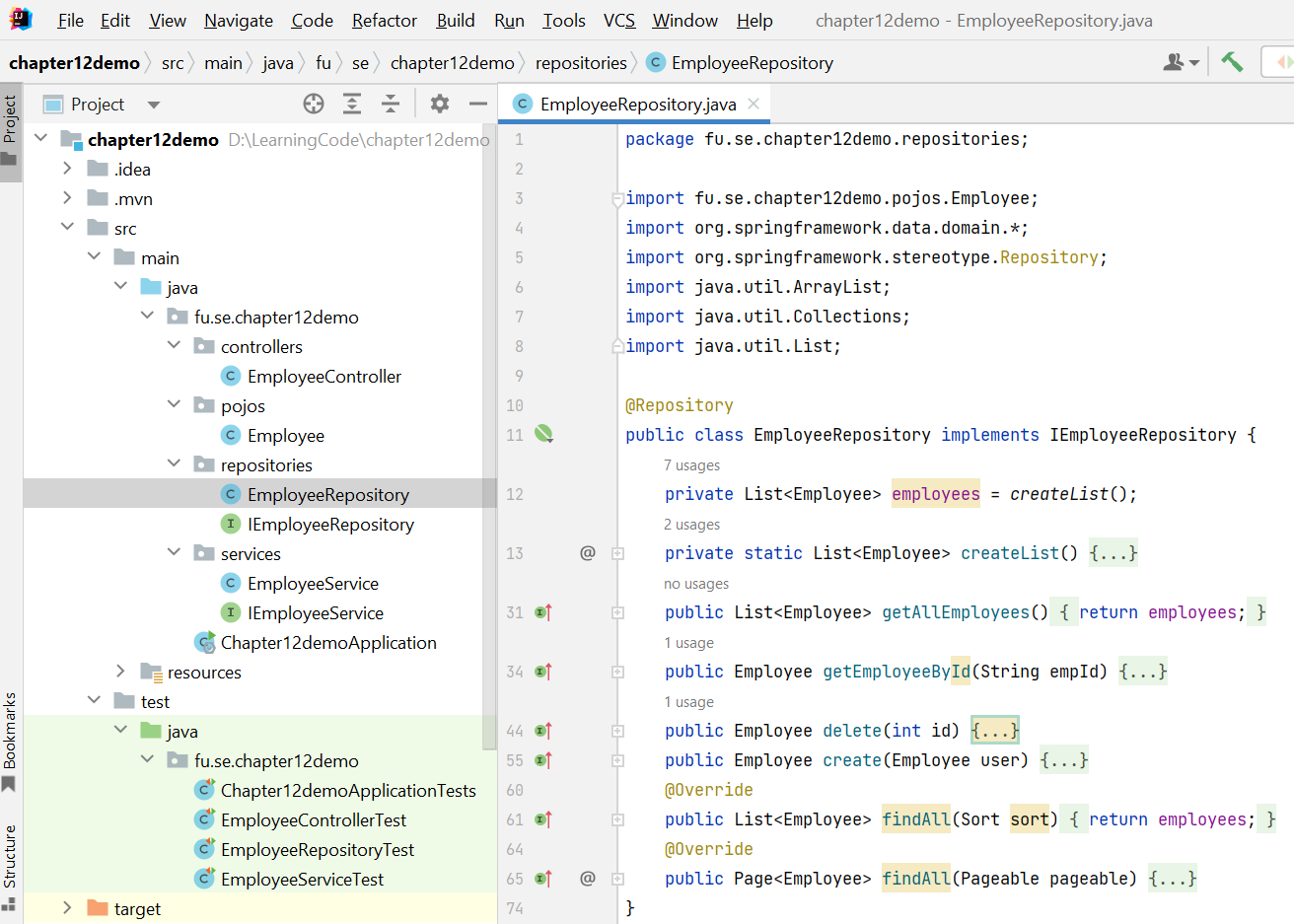
**Step 02**. Implement Repository by creating **repositories** package, IEmployeeRepository interface and EmployeeRepository class

* Implement methods for basic CRUD operations
* For in-memory paging and sorting without a database, override the findAll(Pageable pageable) method.

IEmployeeRepository interface should implement **PagingAndSortingRepository** <T, ID>

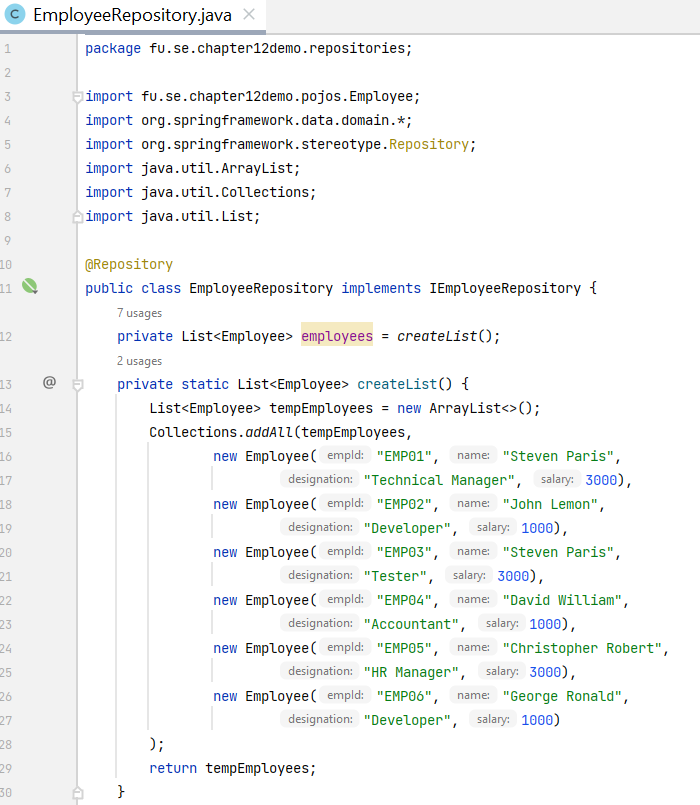
**PagingAndSortingRepository** repository fragment to provide methods to retrieve entities using the pagination and sorting abstraction. In many cases this will be combined with [CrudRepository](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html) or similar or with manually added methods to provide CRUD functionality.



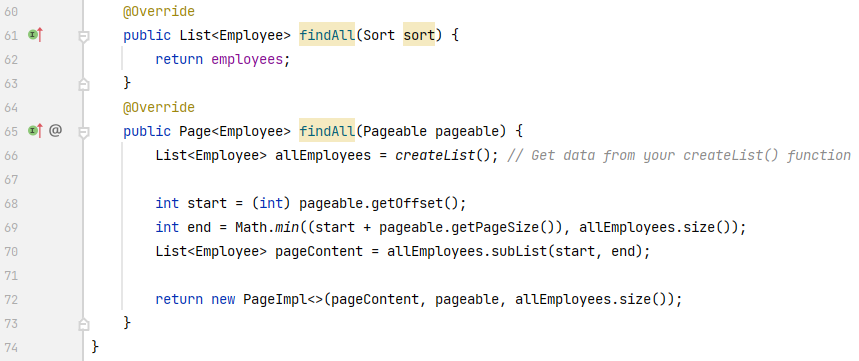


The functions in details

* List<Employee> createList()
* public List<Employee> getAllEmployees()
* public Employee getEmployeeById(String empId)
* public Employee delete(int id)
* public Employee create(Employee user)
* @Override  
  public List<Employee> findAll(Sort sort)
* @Override  
  public Page<Employee> findAll(Pageable pageable)

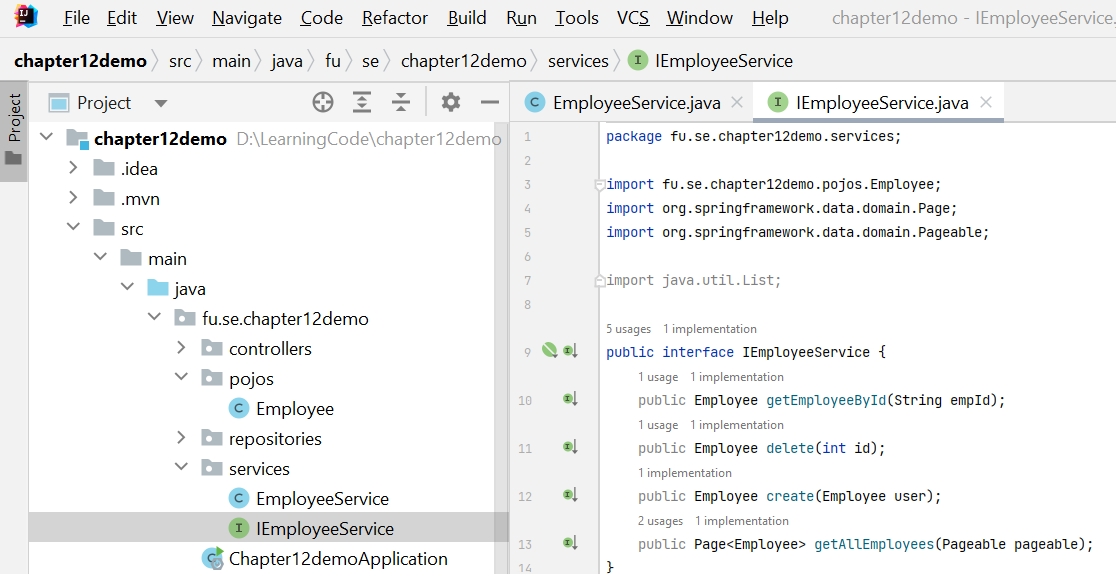


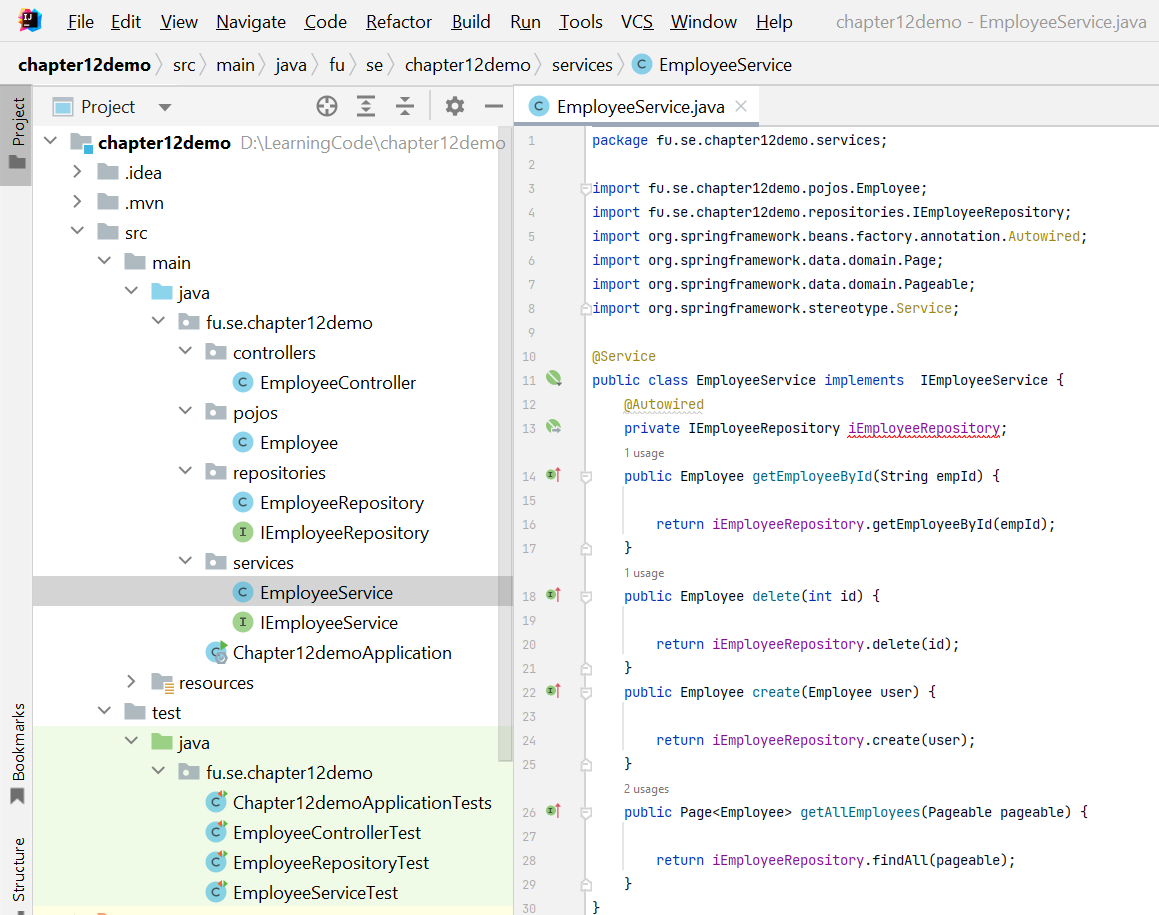


****

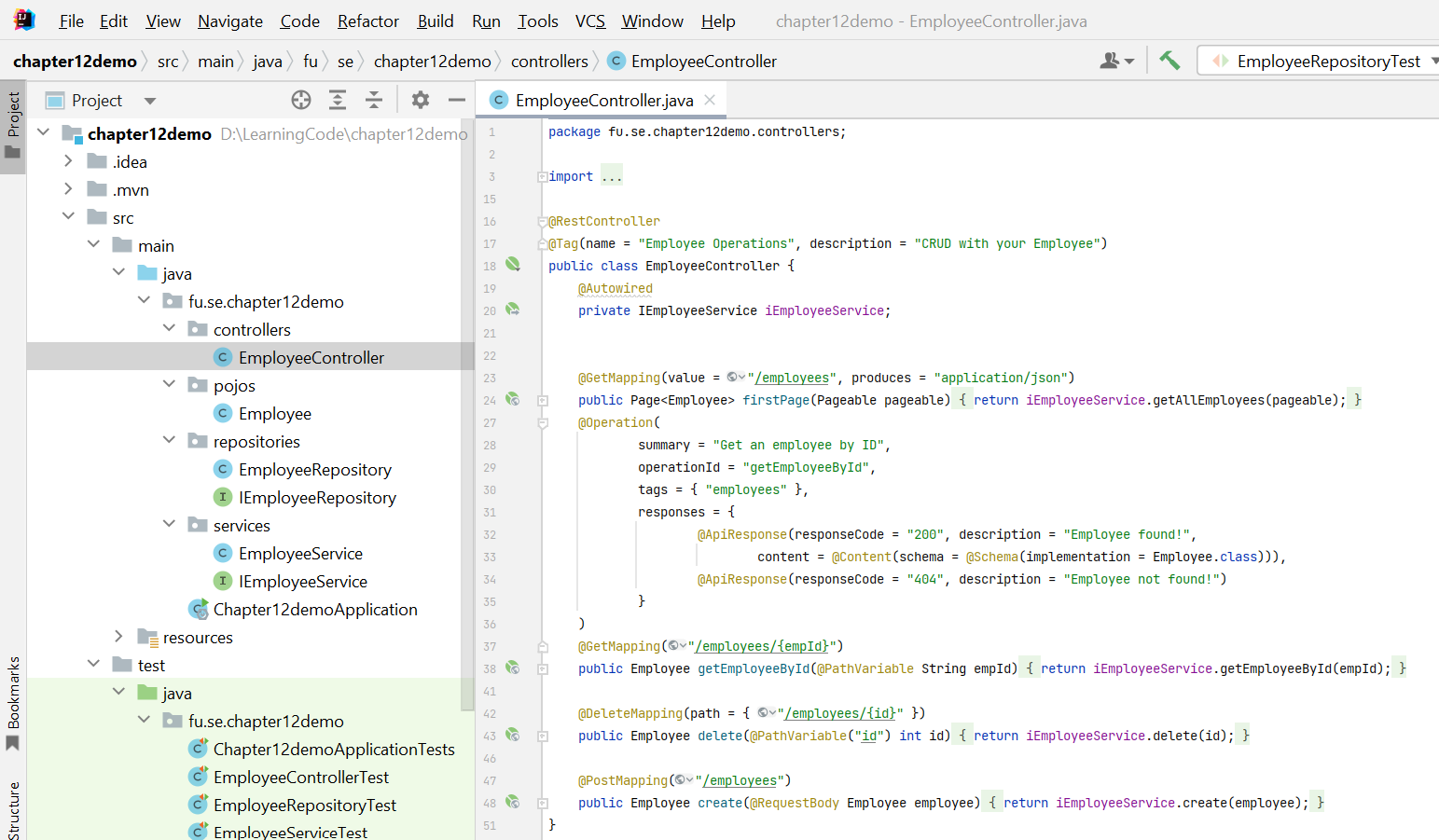
**Step 03**. Implement Service Layer includes: IEmployeeService interface, EmployeeService class in the **services** package.

* Implement business logic in your service methods.
* Use the EmployeeRepository to interact with the Repository (Data Layer)





**Step 04**. Implement REST Controller

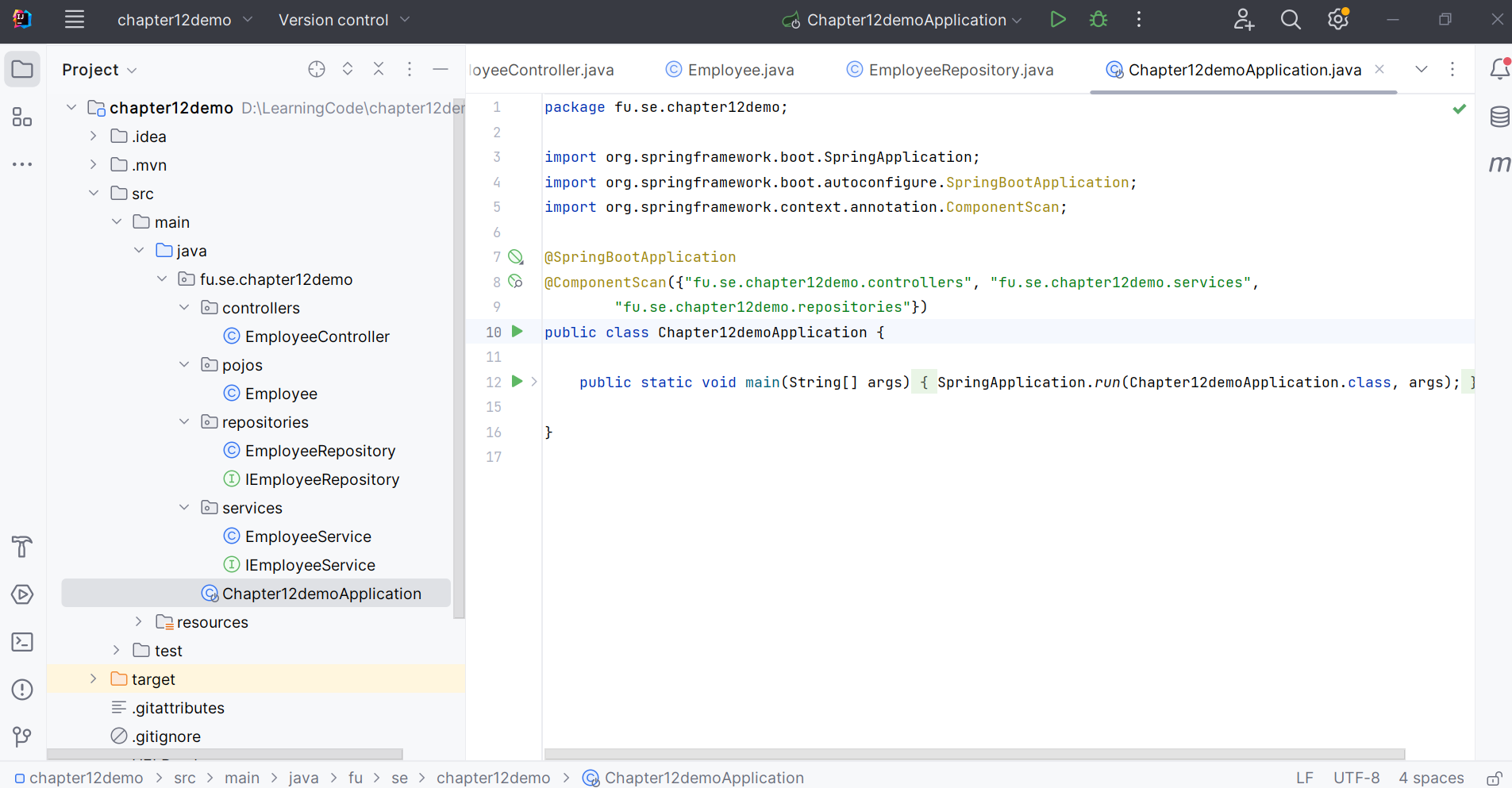
****

**Step 05**. The main entry point for Spring Boot application

The @SpringBootApplication annotation is a convenience annotation in Spring Boot that combines several annotations to simplify the configuration and initialization of a Spring Boot application.

When you use @SpringBootApplication, it automatically includes the following annotations:

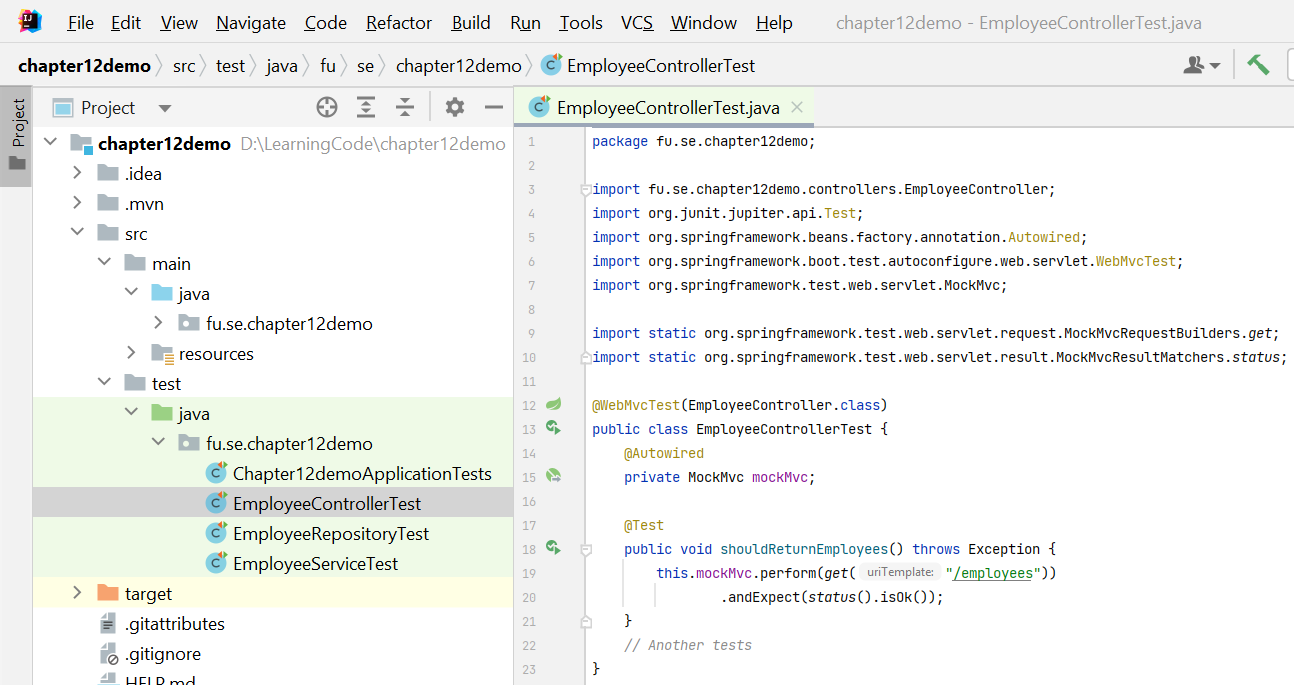
* **@EnableAutoConfiguration:** This annotation tells Spring Boot to automatically configure your application based on the libraries on the classpath. For example, if Spring Boot detects that you have a database driver in your project, it will automatically configure a datasource.
* **@ComponentScan**: This tells Spring to scan the current package and its sub-packages for annotated components (such as @Component, @Service, @Repository, and @Controller) to register them as Spring beans.
* **@Configuration**: This marks the class as a source of bean definitions for the application context, meaning that you can define @Bean methods in this class.

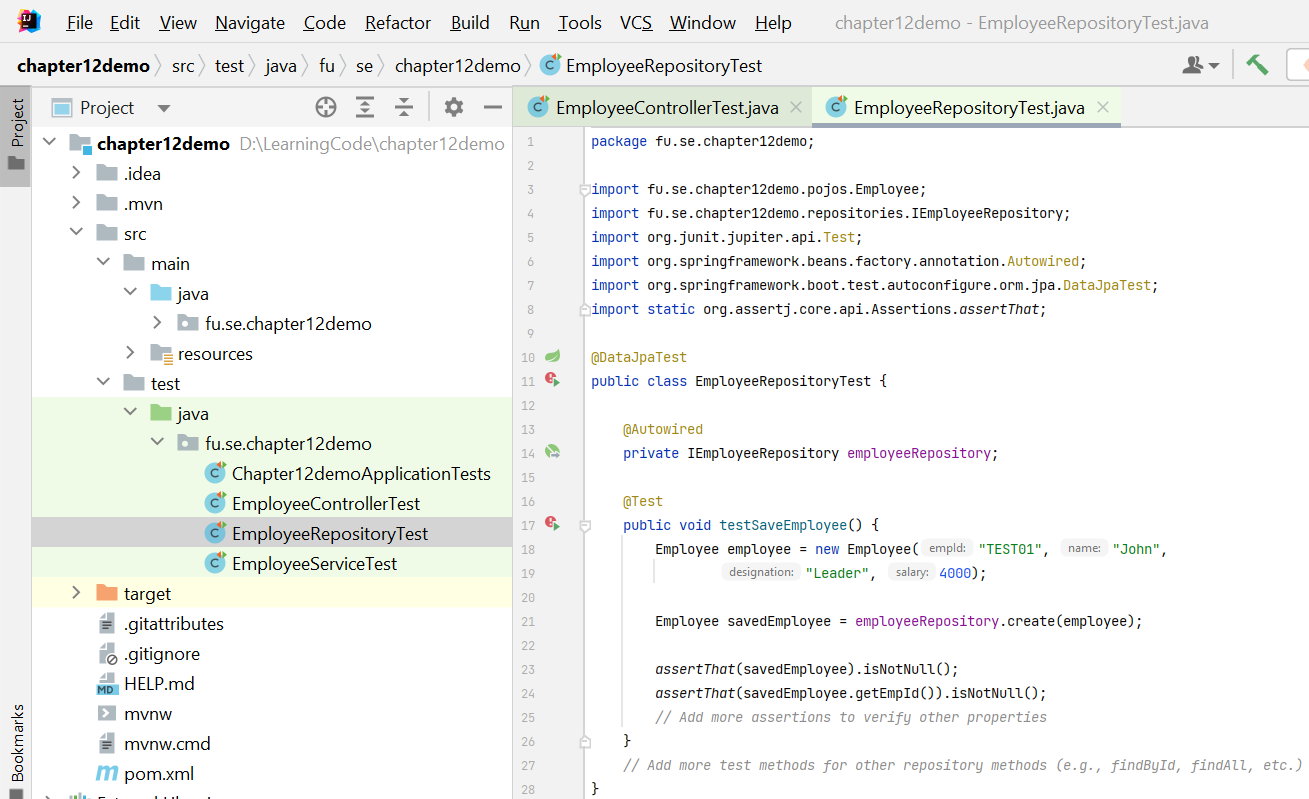
****

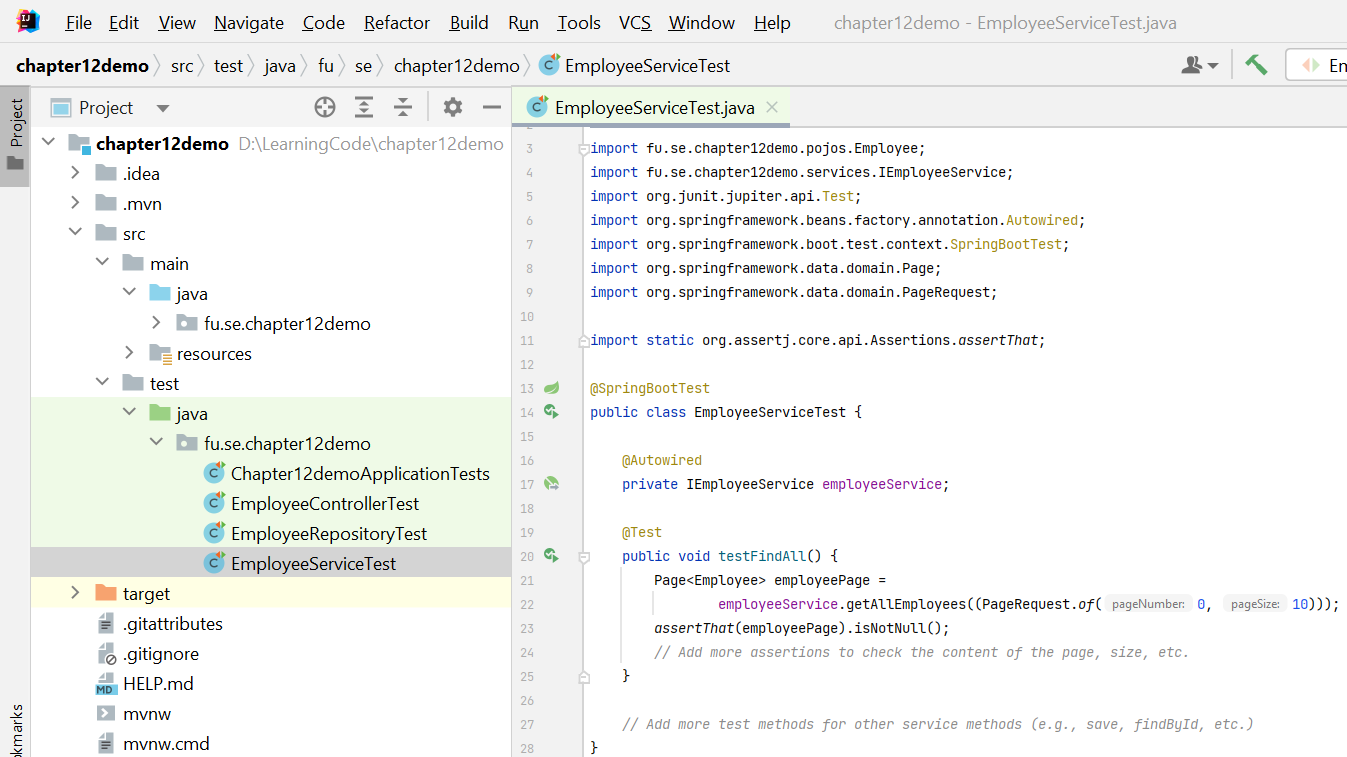
**Step 06**. Create Testing classes.Make sure that, the project has the following dependencies: spring-boot-starter-test, JUnit 5, Mockito, and AssertJ in **pom.xml** file.

Write tests

* Create test classes for your controller, service, and repository.
* Use @WebMvcTest for controller tests with MockMvc.
* Use @SpringBootTest for integration tests involving the full application context.
* Use @DataJpaTest for repository tests with an embedded database.

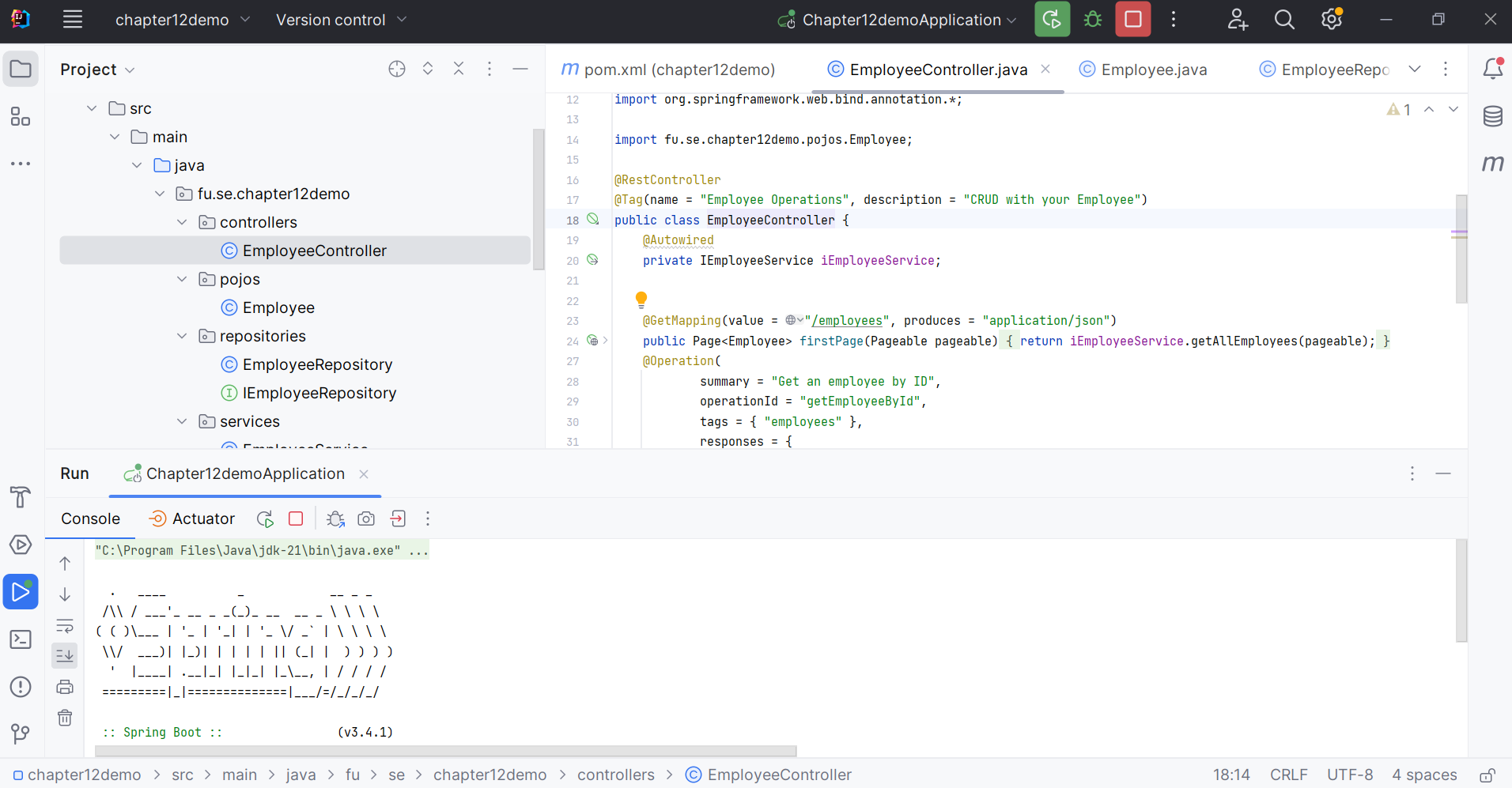
****

****

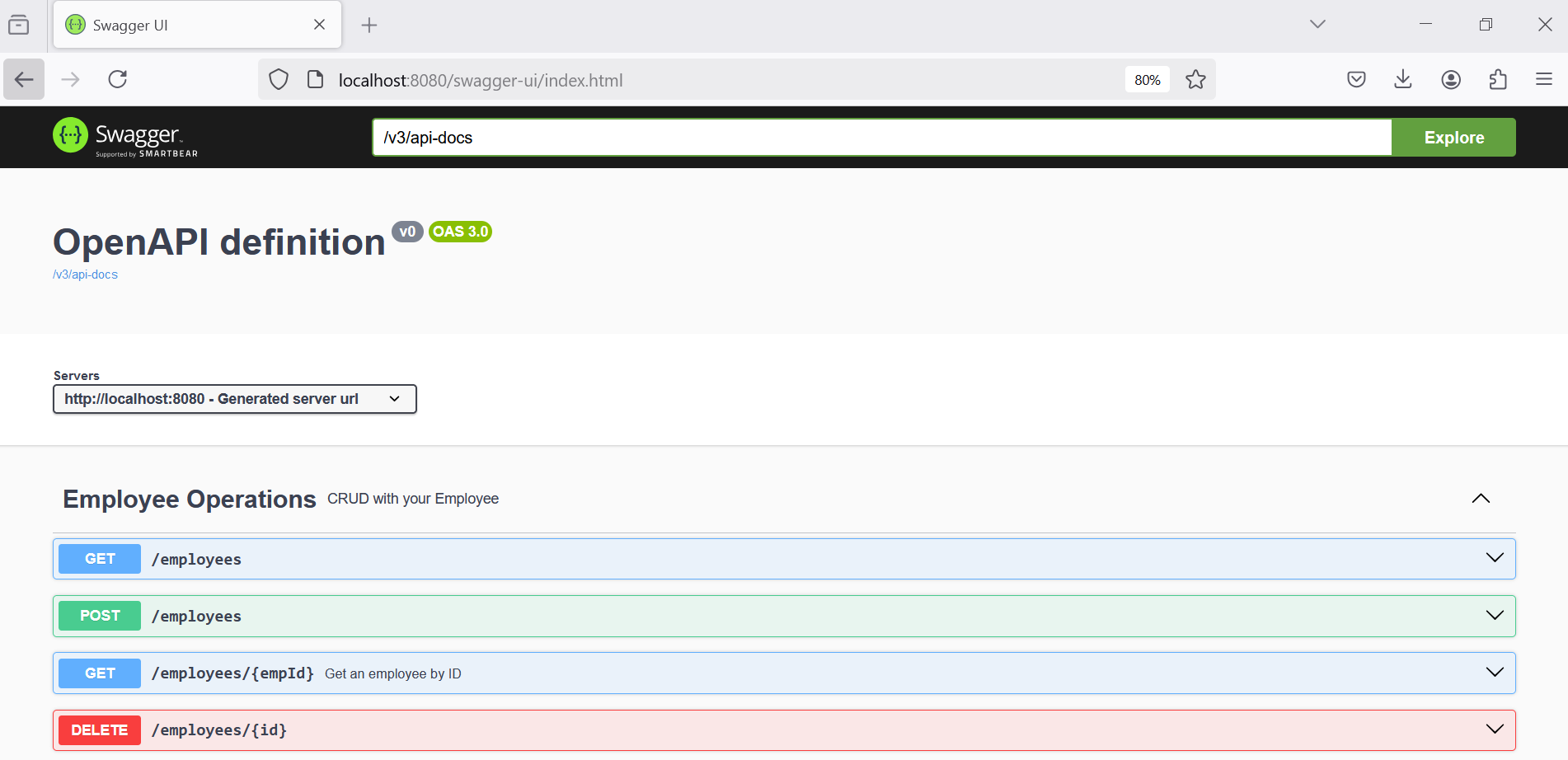


# Activity 03: Run and test the RESTful API

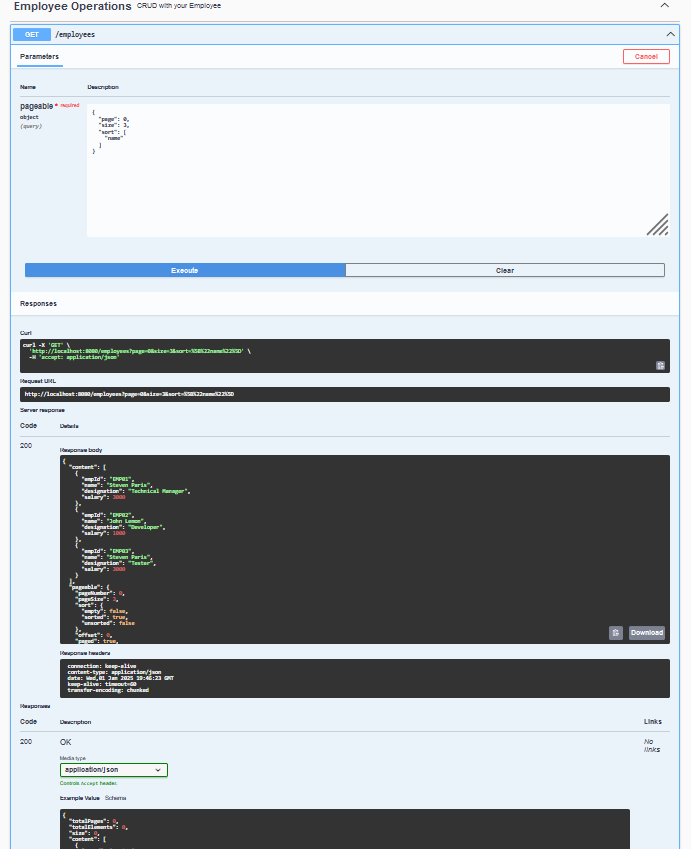
**Step 01.** Run the RESTful API Project



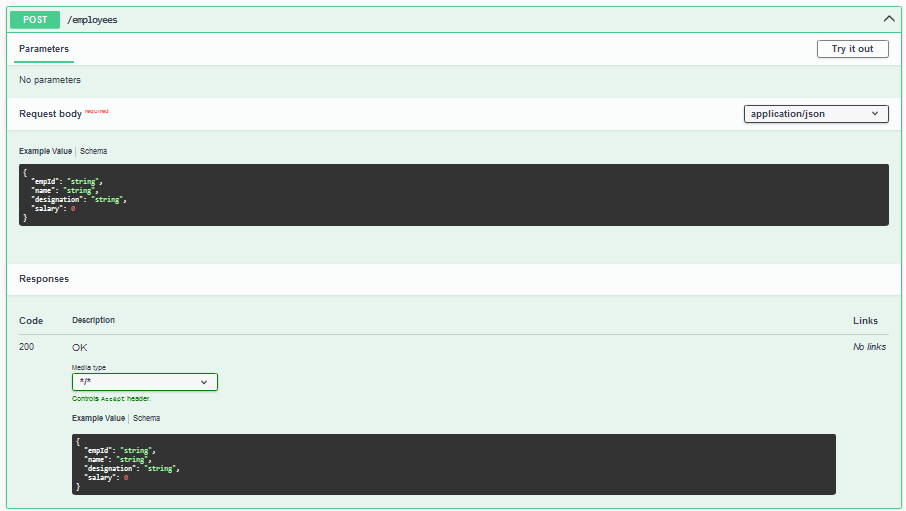
**Step 02.** Open <http://localhost:8080/swagger-ui/index.html> in your Browser



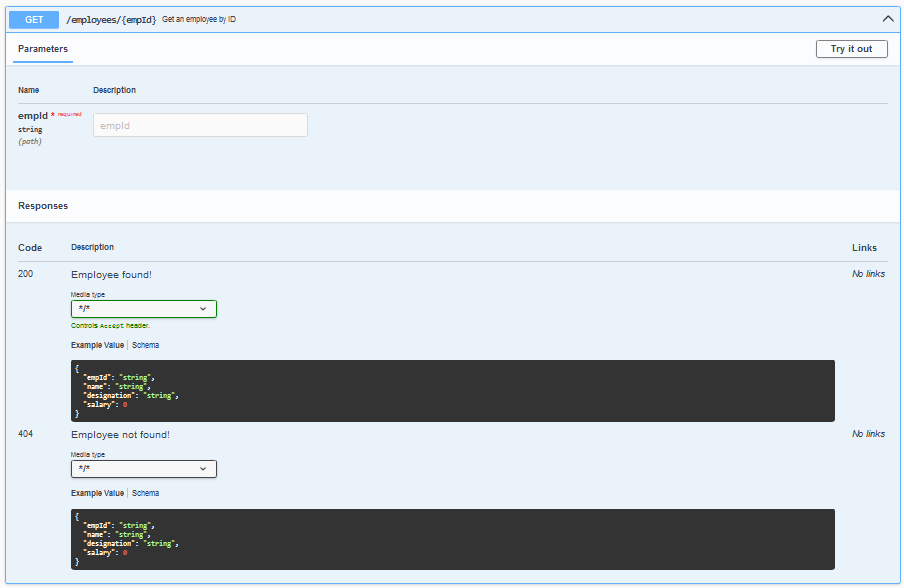
**Step 02.** Test the GET method with paging, provide page, size and sort parameters



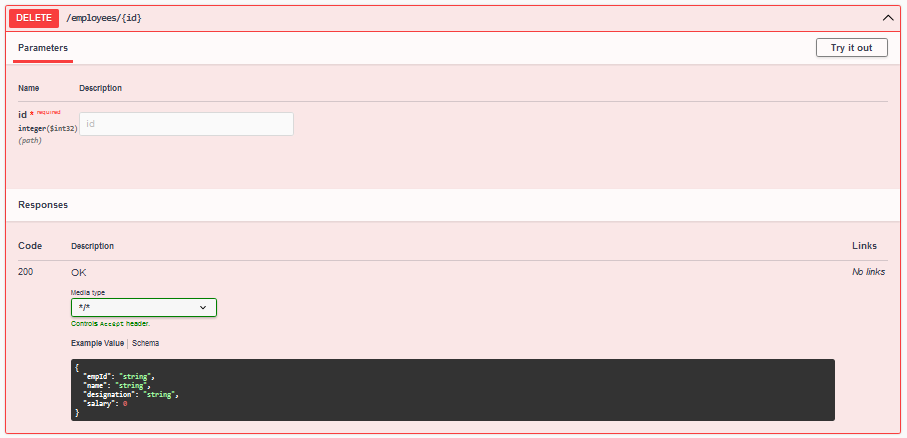
**Step 03.** Test the POST method for creating a new employee, provide empId, name, designation and salary parameters



**Step 04.** Test the GET method for existing employee, provide empId parameter.



**Step 05.** Test the DELETE method for existing employee, provide empId parameter.



Note: The Schema of this application in Swagger UI

