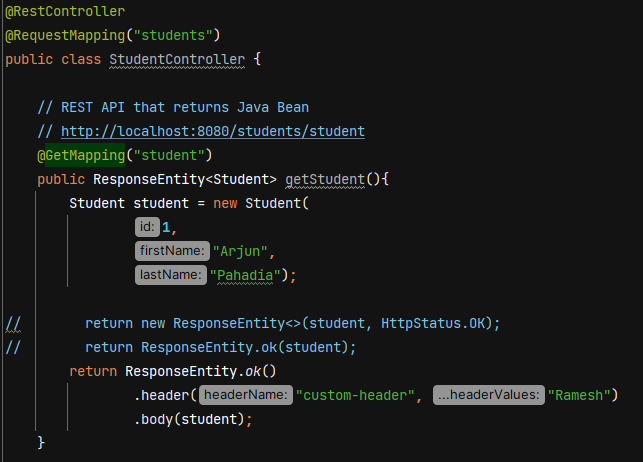
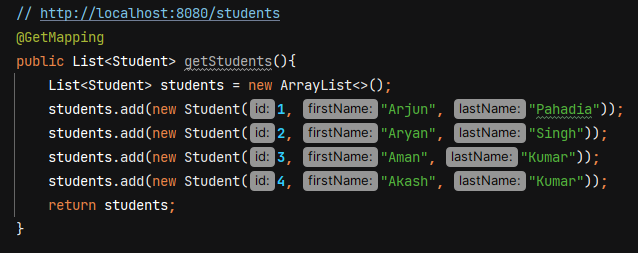
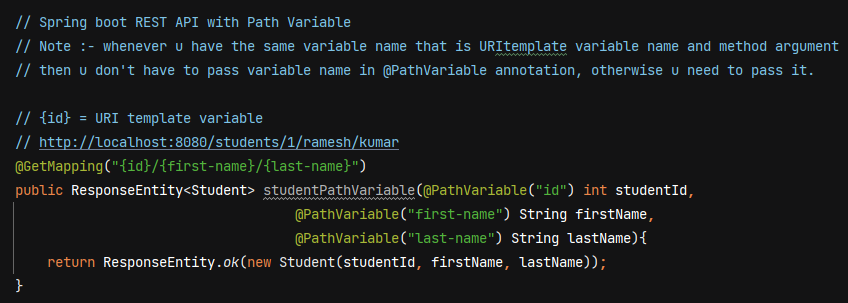
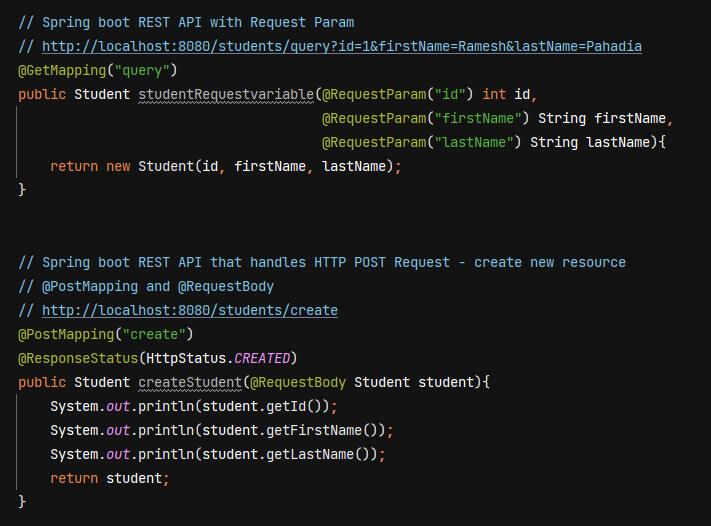
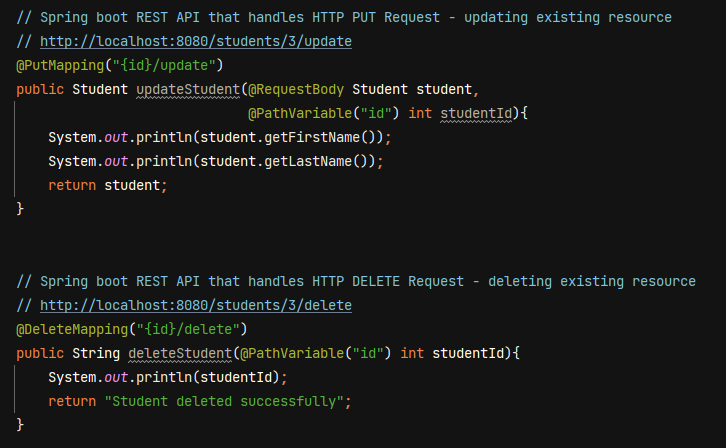
**REST APIs in SpringBoot**

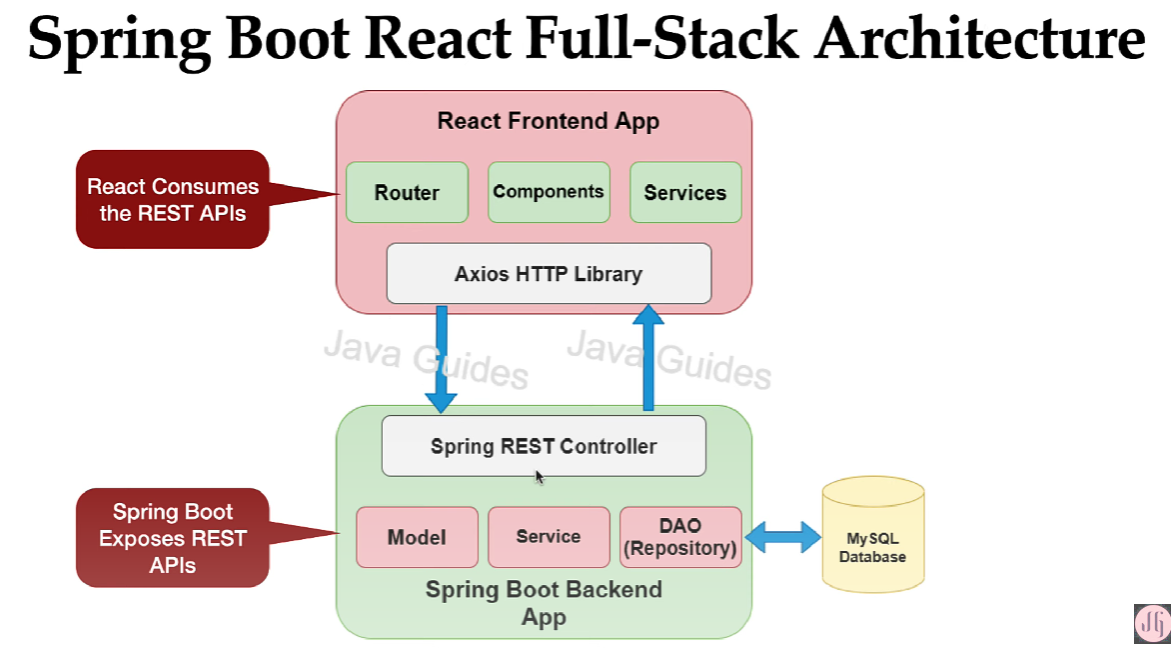












Both react frontend app and springboot backend app are loosely coupled.

For backend, we make 3 layer architecture – controler, service and DAO layer. The dao layer is responsible to talk with the DB and the service layer basically contains the business logic of app, and the controller basically contains the spring MVC controllers which exposes the REST APIs.

In react frontend app, we create components, services, router. We have used Axios http library to make a rest api call. We use a JSON format to exchange the data between react frontend and springboot backend.

**DTO (Data Transfer Object)**

It is widely used design pattern to transfer the data between client and server.

Server

Client

Client can create a DTO Object and it will send that DTO object in the HTTP request and server will extract the dto object from the request and it will use that DTO object. Similarlly server will create a DTO object and it will send that DTO object in the reponse of the rest API.

**Main advantage is to reduce the number of remote calls to the server**. For ex- in our employee managent system we have Organization inside which we have list of depts and within depts we have list of employees. Now if we want all the data, we have to make 3 individual rest api calls to get organization, list of depts and employees. But we can create a APIResponseDTO class having company, List<dept> and List<employeed> and send it back to the client.

Server can use DTO to transfer the only required amount of data to the client.

**How to use DTO pattern in Spring Boot**

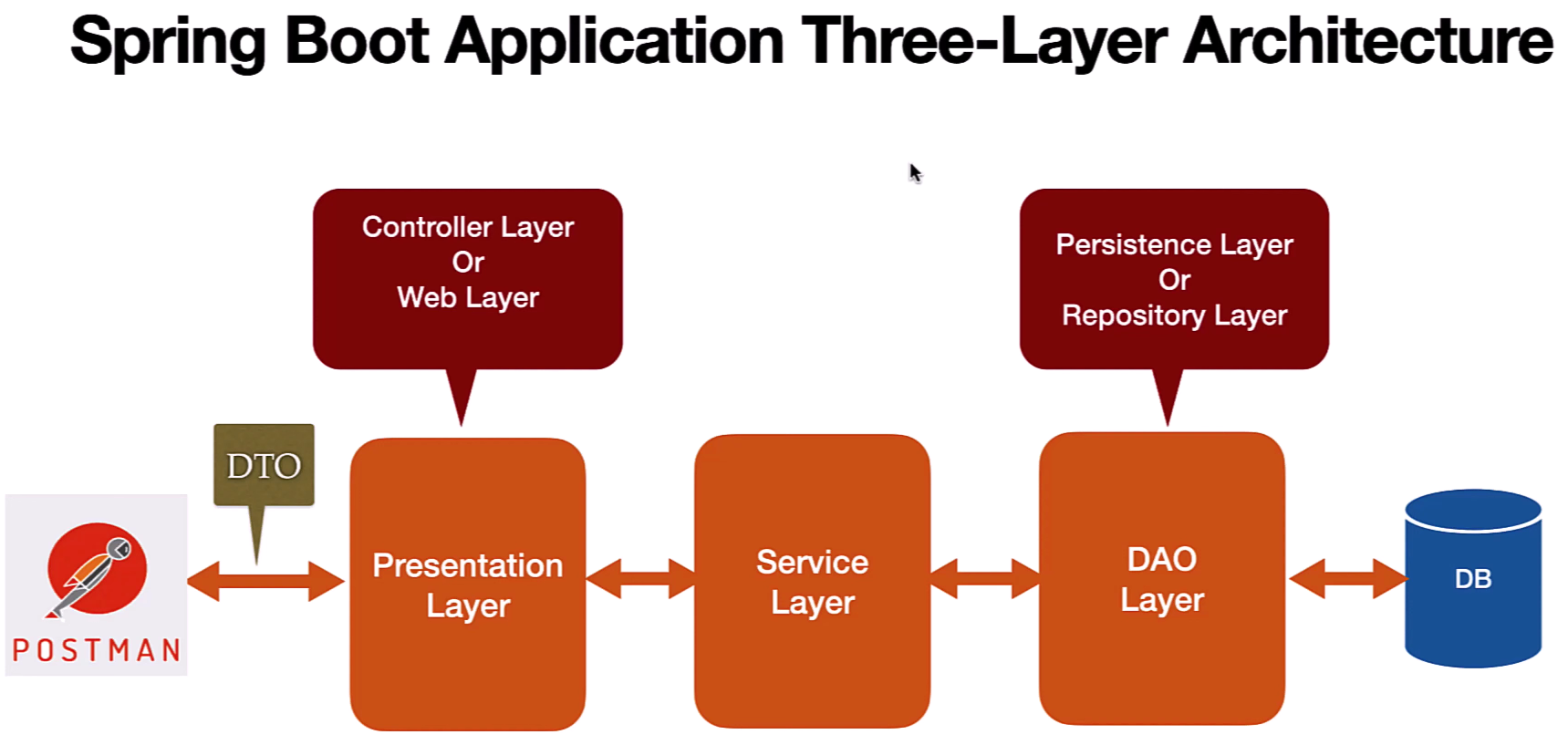
In springboot we have JPA entities and we use JPA entitity to map object to the relational database table. DAO / Repository layer uses JPA entities to store the data into a database and retrieve it.

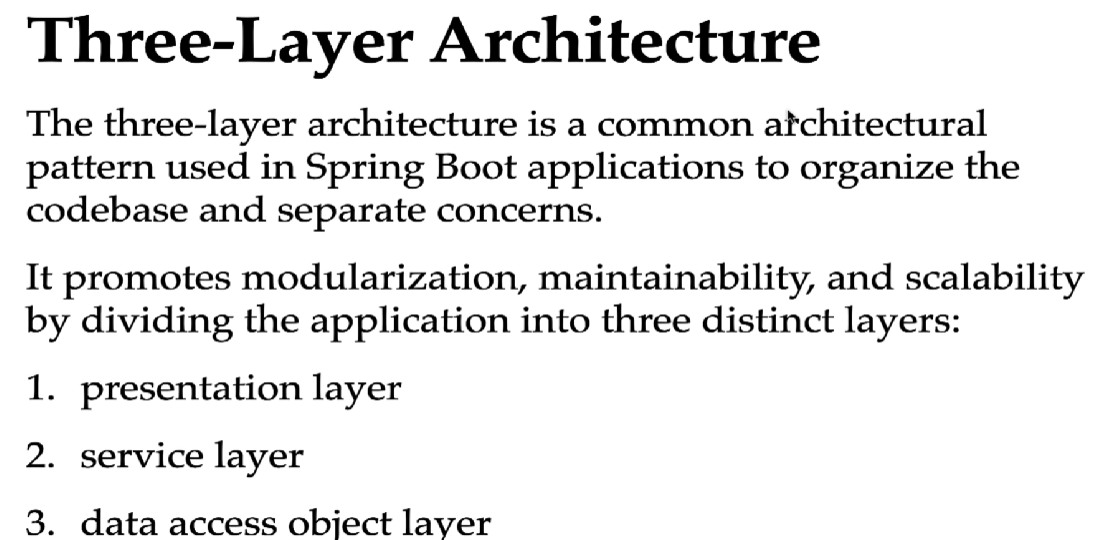
Client and Server uses DTO to transfer data. Some developers uses JPA entitites to transfer to data between client and server. But transfering JPA entitity has some disadvantages –

* Transporting the sensitive information. Consider our JPA entity has some fields like username, password, some codes. If we don’t handle this sensitive info and our REST API directly send the JPA entity to the client, the client will get the password and all the sensitive information.

To overcome this problem we can use DTO to transfer the data between client and server.

**In DTO we’ll keep only the required data that client expect in a response of the rest API.**

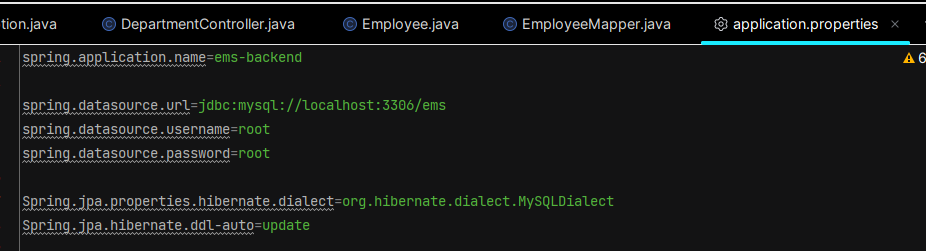




In application.properties file we configure database details

Hibernate uses **MySQLDialect** to create the SQL statement based on the database that we are using.

**Spring.jpa.hibernate.ddl-auto=update ->**  this property tell hibernate to automatically create the database tables if the tables don’t exist in the database and is there are any changes in a JPA entities, then it will also tell Hibernate to update those changes in the DB tables as well.



Springboot by default uses Hikari data source and Hikari connection pool.

**Creating Employee JPA Entity**  - create a java class Employee.java in entity folder

@Entity - to specify a class as a JPA entity

@Table(name = ”employees”) – to specify the table name, if we don’t give name=” ” then jpa will take table name same as the class name

@Id – to configure the primary key

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

* to configure PK generation strategy
* IDENTITY generation strategy uses database autoincrement feature to automatically increment the PK

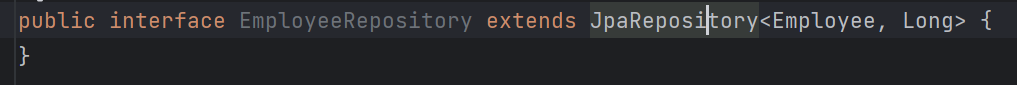
@Column(name = ”first\_name”, nullable = false, unique = true)

* to map a database table column with a class field. If u don’t mention it then JPA willl by default give column name as field name
* nullable = false makes the column value not null
* unique = true to make column value unique

After creating JPA entity, if we run the app then hibernate will automatically create a table in our database.



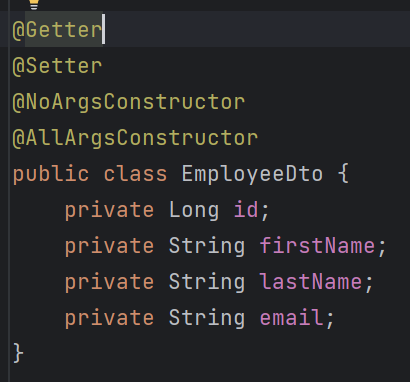
**Creating EmployeeRepository** – create interface EmployeeRepository in repository folder

 JpaRepositry is a generic interface, so pass 2 parameters – type of Entity and type of Primary Key

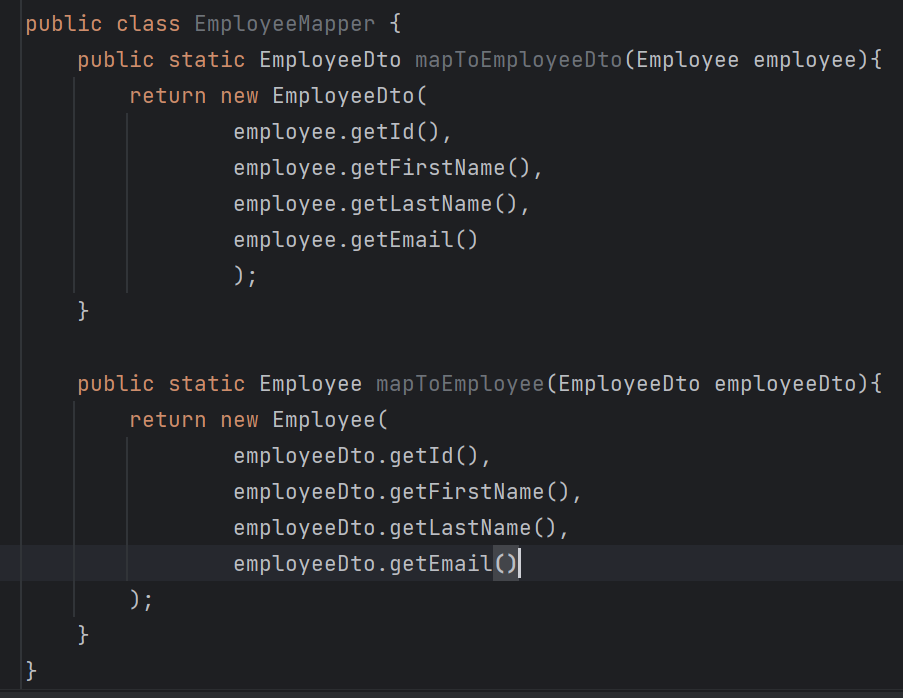
* EmployeeRepository will get methods to perform CRUD database operations on Employee Entity
* JpaRepository will inherit all the methods from all the entended interfaces. SimpleJpaRepository class of Spring Data JPA provide the impl for JpaRepository interface.
* We don’t have to annotate EmployeeRepository with @Repository annotation because the impl class SimpleJpaRepository is already annotated with @Repository annotation.
* SimpleJpaRepository class is also annotated with @Transactional. All the public methods in a SimpleJpaRepository are transactional so we don’t have to again use @Transactional to make these methods transactional.

**Create EmployeeDto and EmployeeMapper**

We’ll use EmployeeDto class to transfer data between client and server.

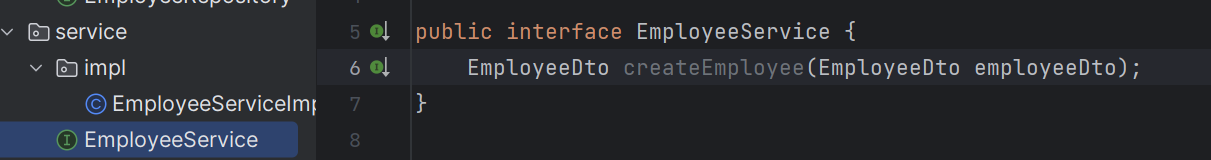


We create EmployeeMapper class to map Employee entity to EmployeeDto and EmployeeDto to EmployeeEntity.



**Build Add Employee REST API**

As Controller layer depends on Service layer, so we’ll first create service layer.

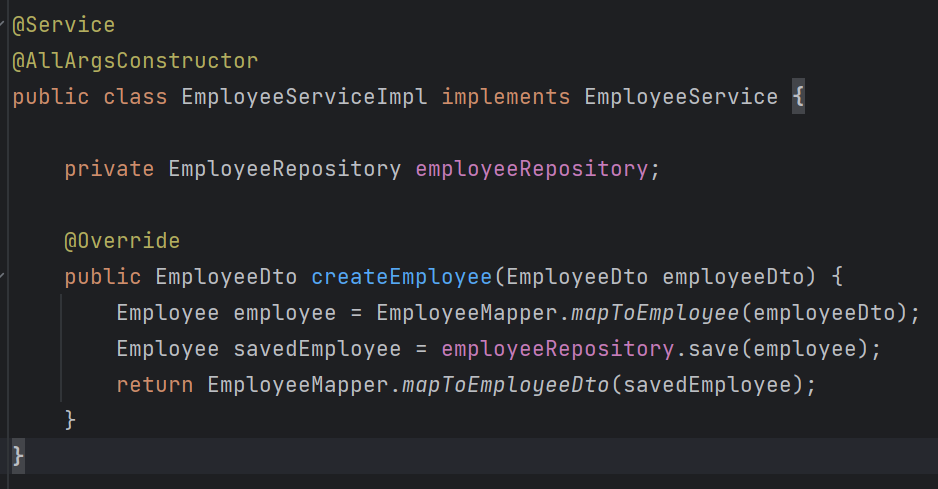


Create an impl of this – EmployeeServiceImpl

Use **@Service** on EmployeeServiceImpl, it will tell spring container to create for this class.

We’ll use constructor based DI (when dependencies are provided through the class constructor) to inject the dependencies so annotate EmployeeServiceImpl with @AllArgsConstructor

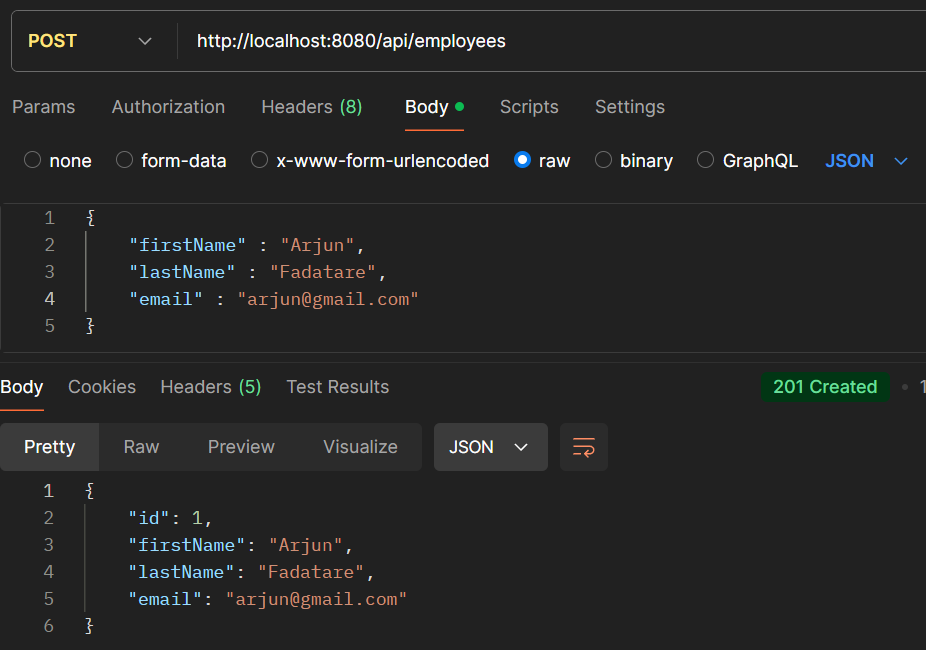
Inside createEmployee method, we need to first convert EmployeeDto into Employee entity becox we need to store the Employee entity in DB.



Now let’s create Controller –



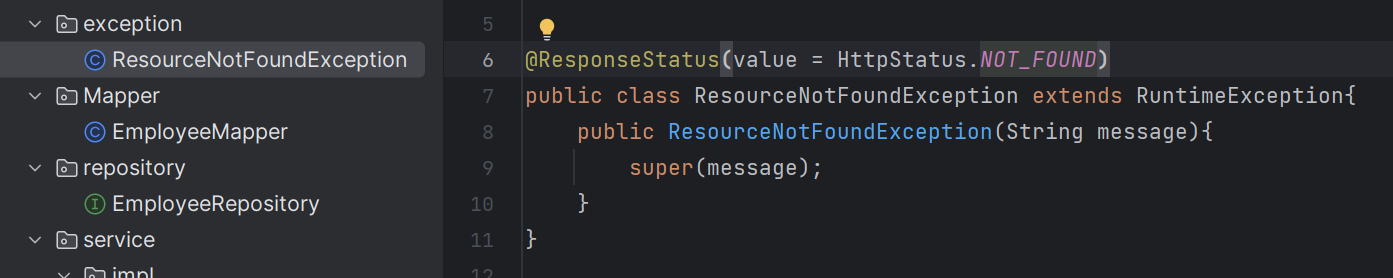
When testing throught Postman and passing JSON in body then json object properties name should match with the Dto class variables.

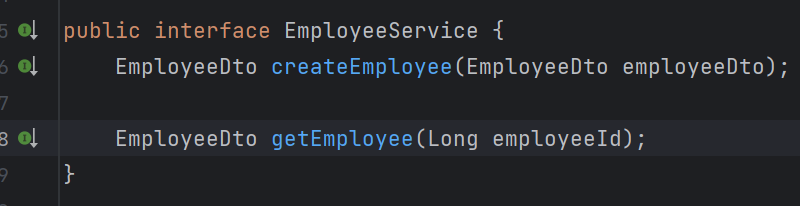


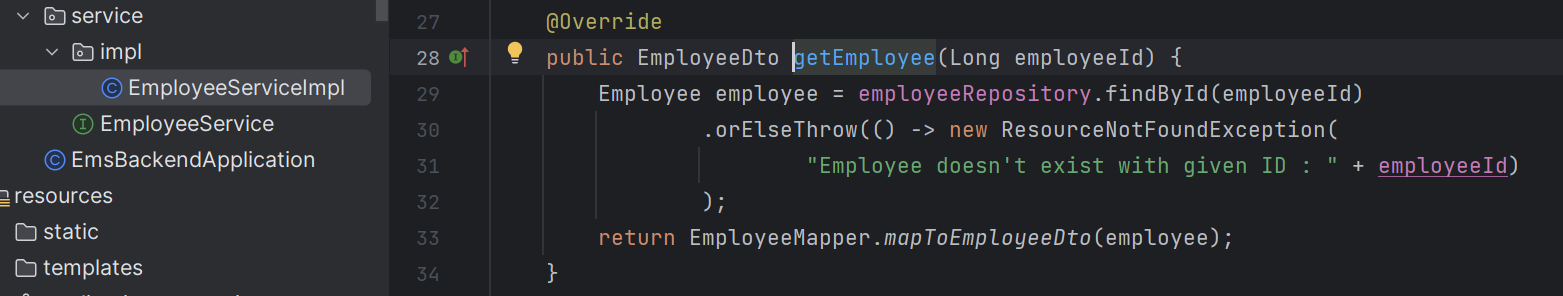
**Build Get Employee REST API**

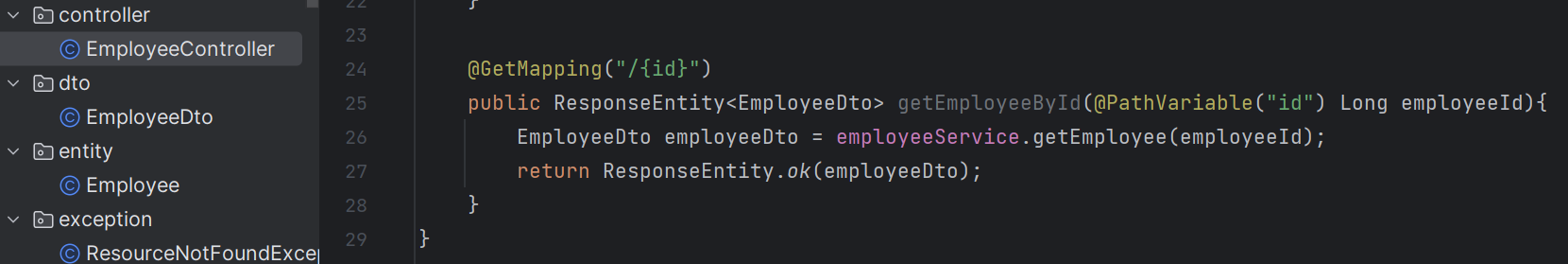
**@ResponseStatus** - used to mark a method or an exception class with the HTTP status code that should be returned in the response.

If a Employee with a given ID does not exist in DB, then throw custom ResourceNotFoundException and then springboot will catch this exception and will get the error message from exception, and it will send the error message along with the http status to client.

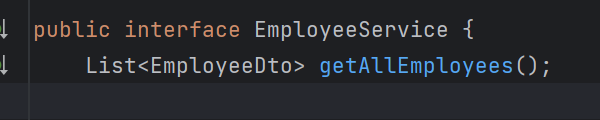




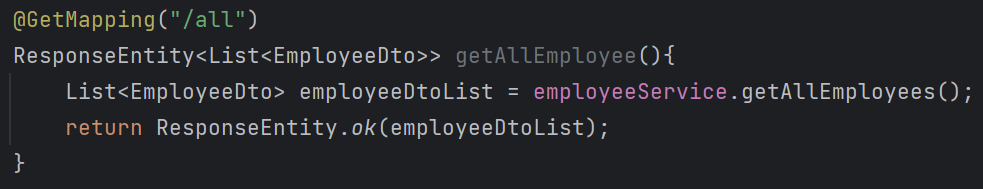




**Build Get All employee REST API**



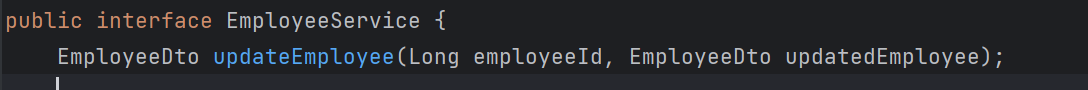




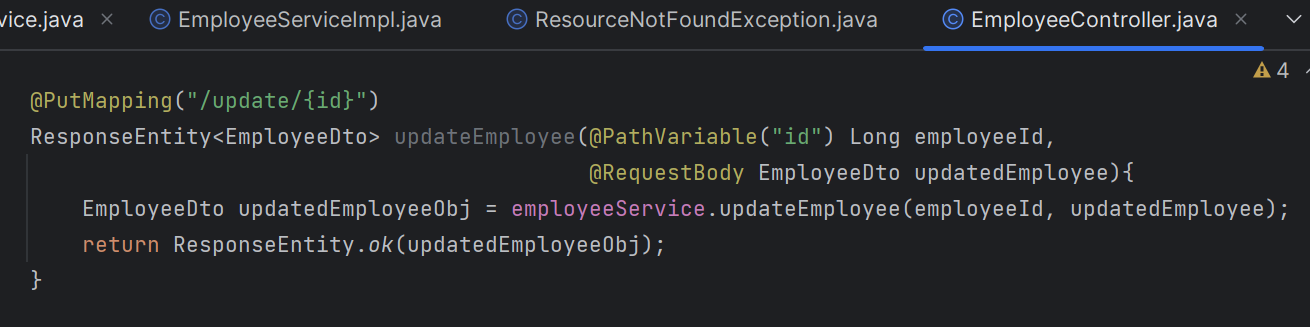
**Build Update Employee REST API**

Save() method of JpaRespository perform both save and update operation.

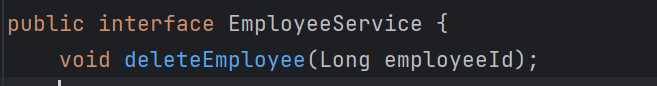
If employee object contains ID then the save method internally perform the update operation. And if employee doesn’t contain the Primary Key ID, then it will perform the insert operation.

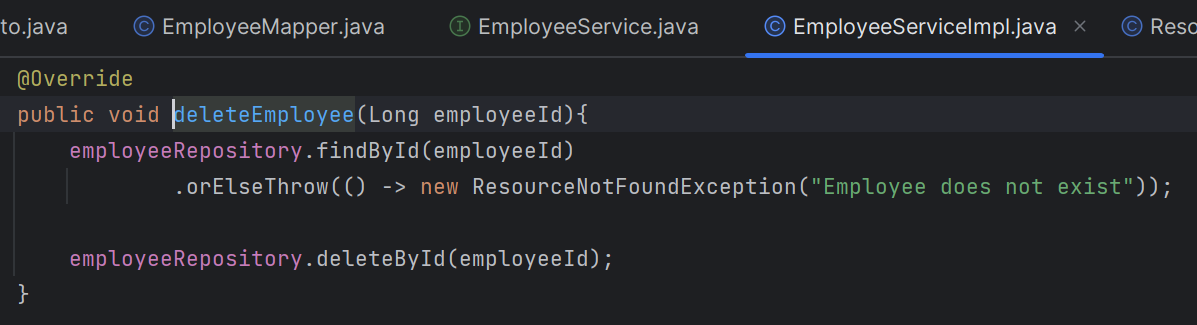


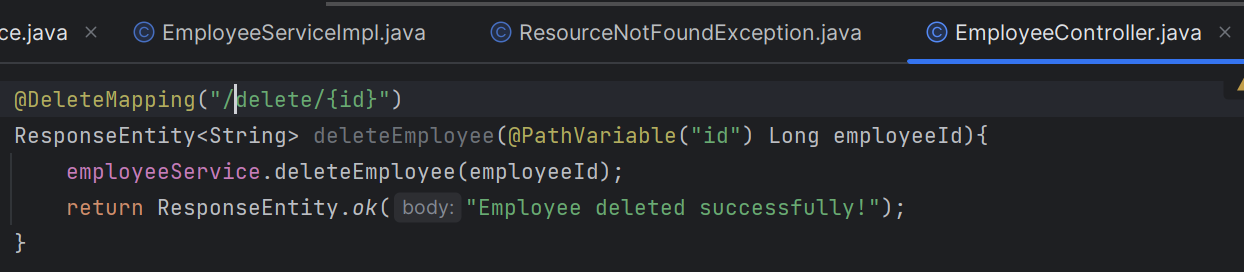




**Build Delete Employee REST API**



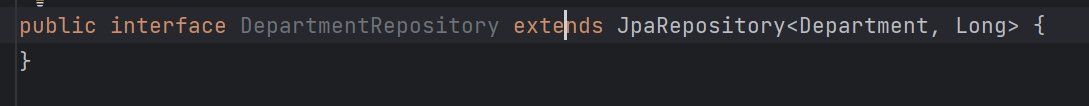




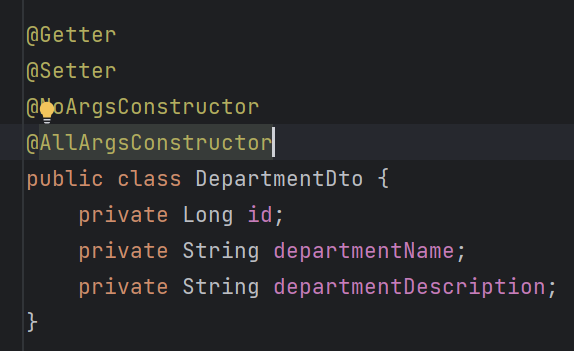
**Requirement 3 :- REST APIs For Department Management Module :**

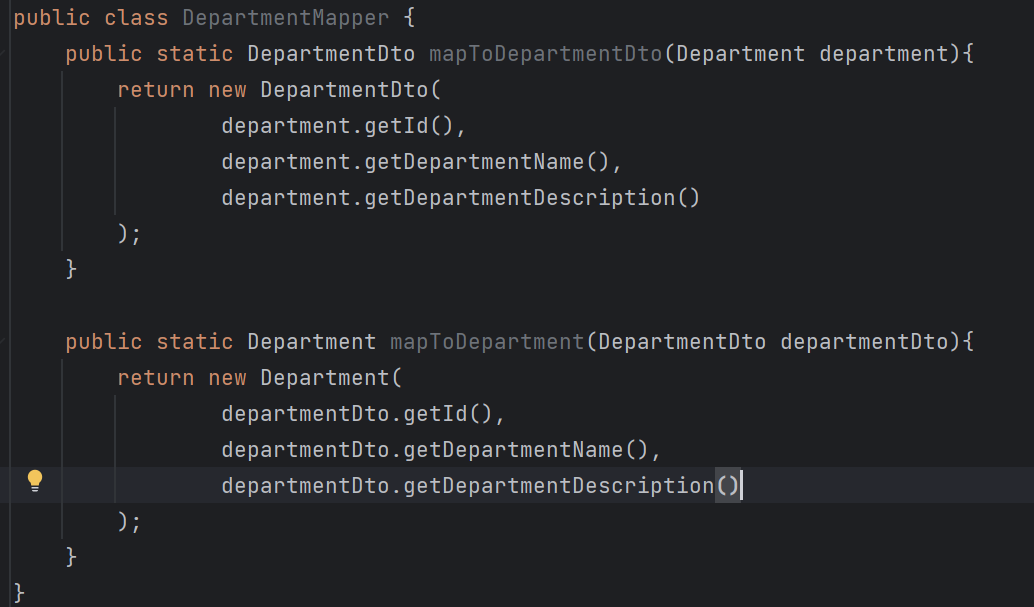
**Create Department Entity & DepartmentRepository**



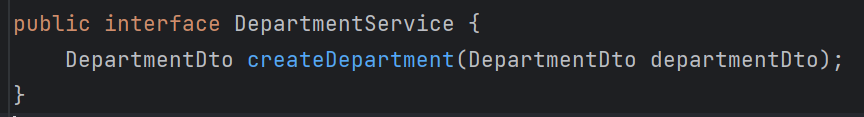


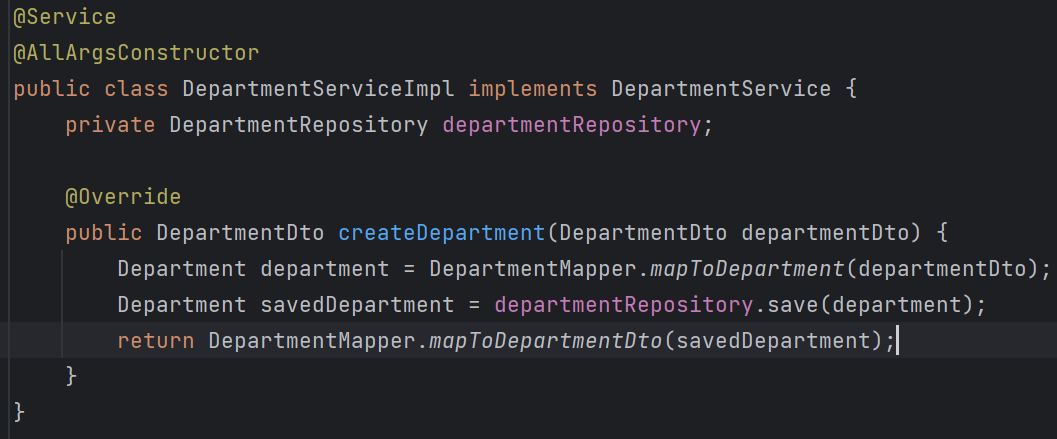
**Create DepartmentDto & DepartmentMapper**

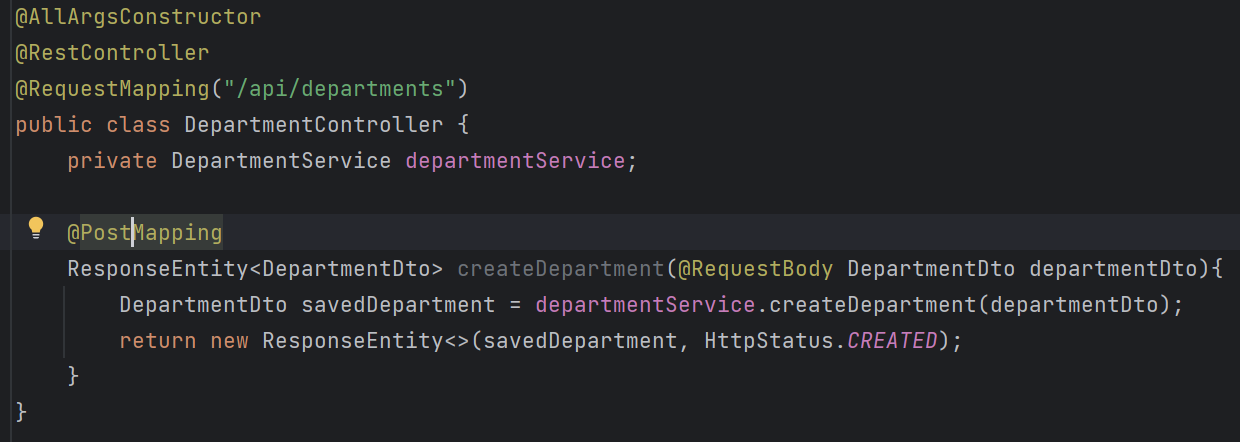




**Build Create Department REST API**







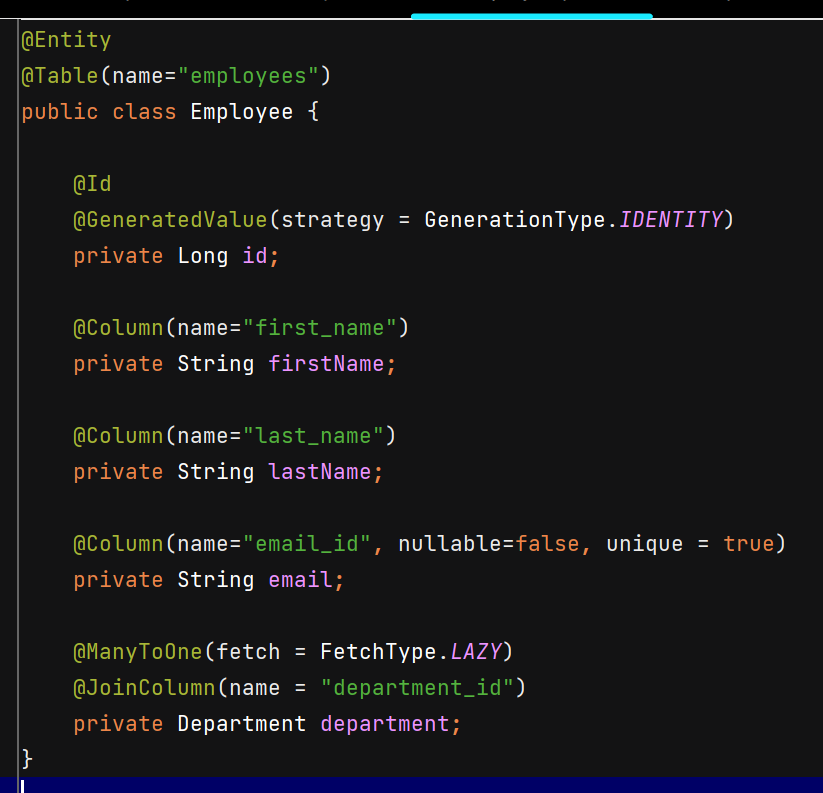
Similarly make REST APIs to get dept by Id, get all dept, update dept and delete dept.

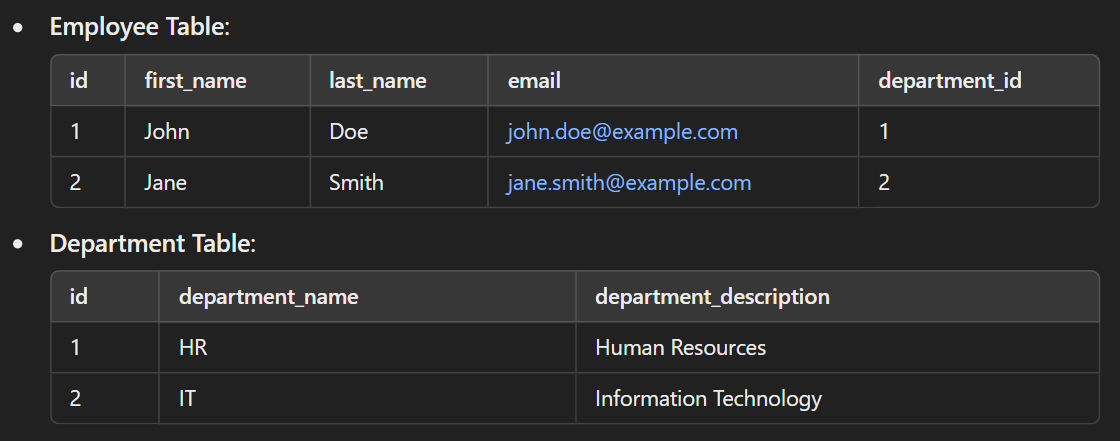
**Many To One Relationship between Employee & Deptarment JPA entities**

So go to Employee JPA entity and add Department instance variable.

Specify FetchType.LAZY becoz whenever we a get employee entity object from DB, the hibernate won’t load the department object immediately. We can get this department object lazily or on demand.

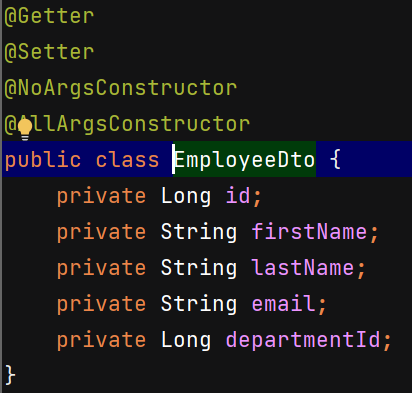
We have to maintain a foreign key in a employee table, specify it using @JoinColumn annotation. name attribute specifies the **name of the foreign key column** in the Employee table that will reference the primary key of the Department table.





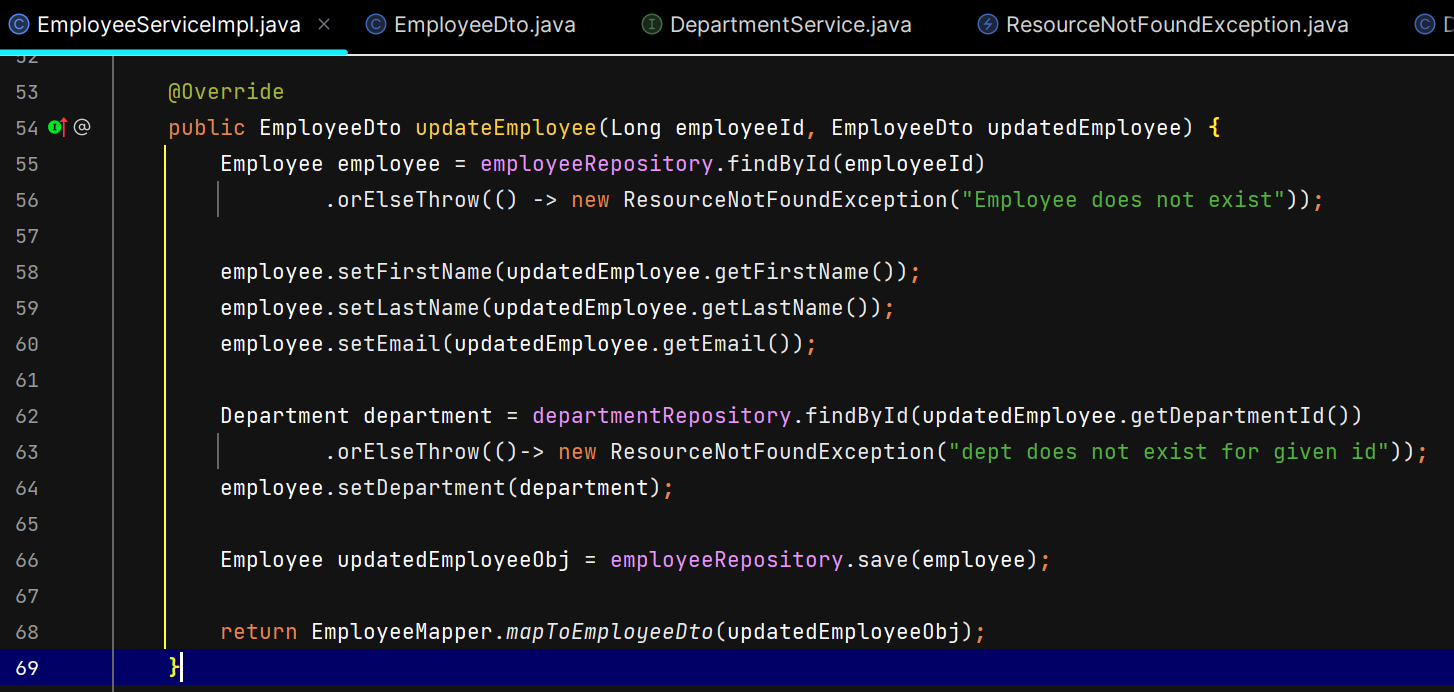
Whenever we save the Employee object in DB, we have to add Department to that Employee object, changes in EmployeeServiceImpl –

Add departmentId to EmployeeDto –





User can change department for particular employee –



Change implementation of mapToEmployee from constructor to setters, and for mapToEmployeeDto pass the department Id –

