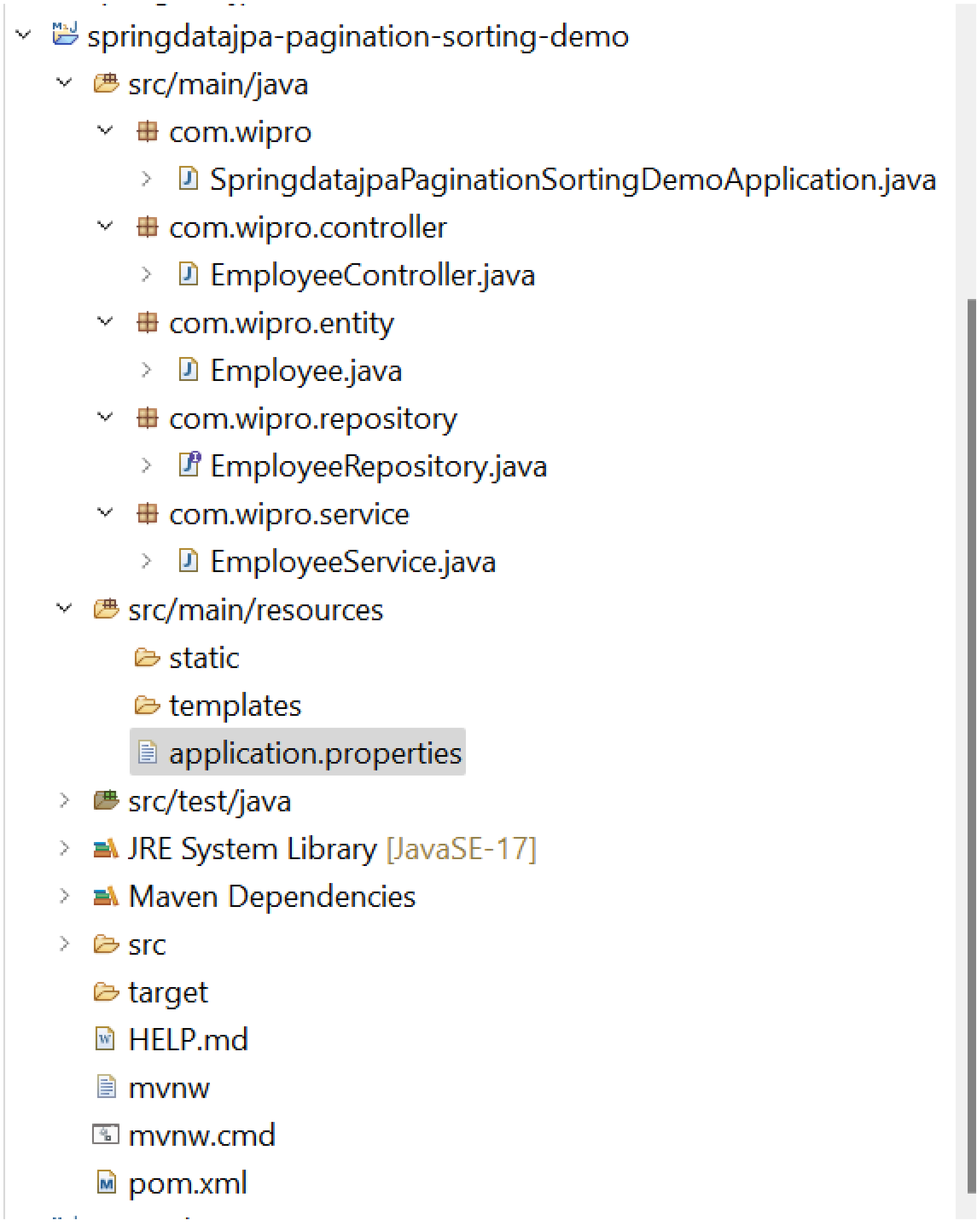
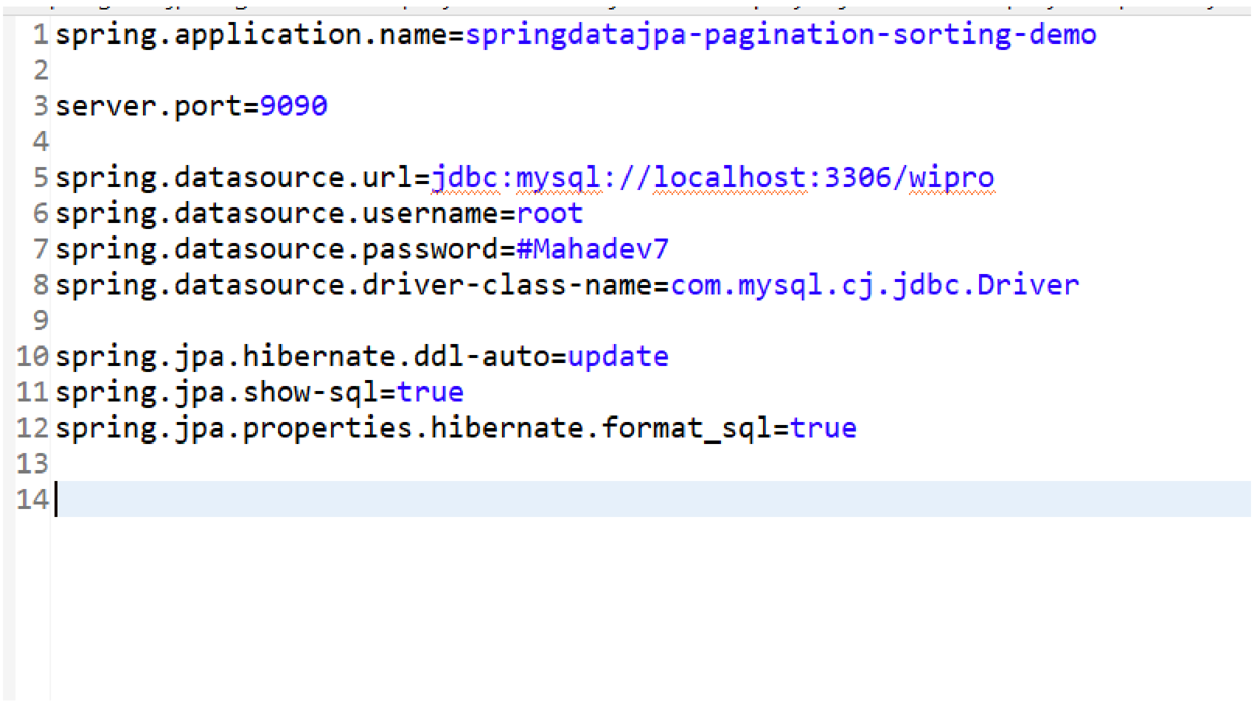
27-02-2025 **Pagination and sorting**

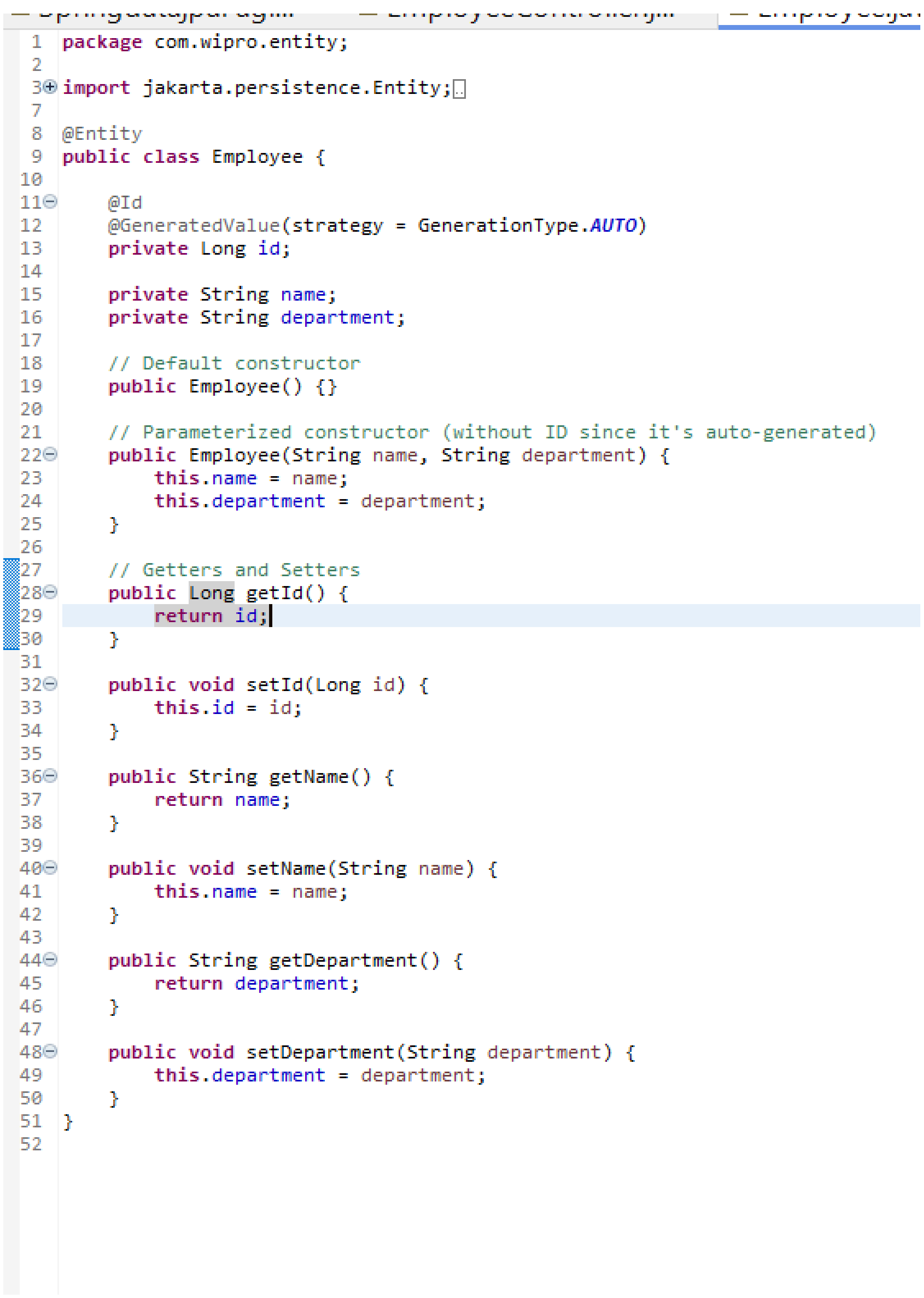
Pagination demo by sir



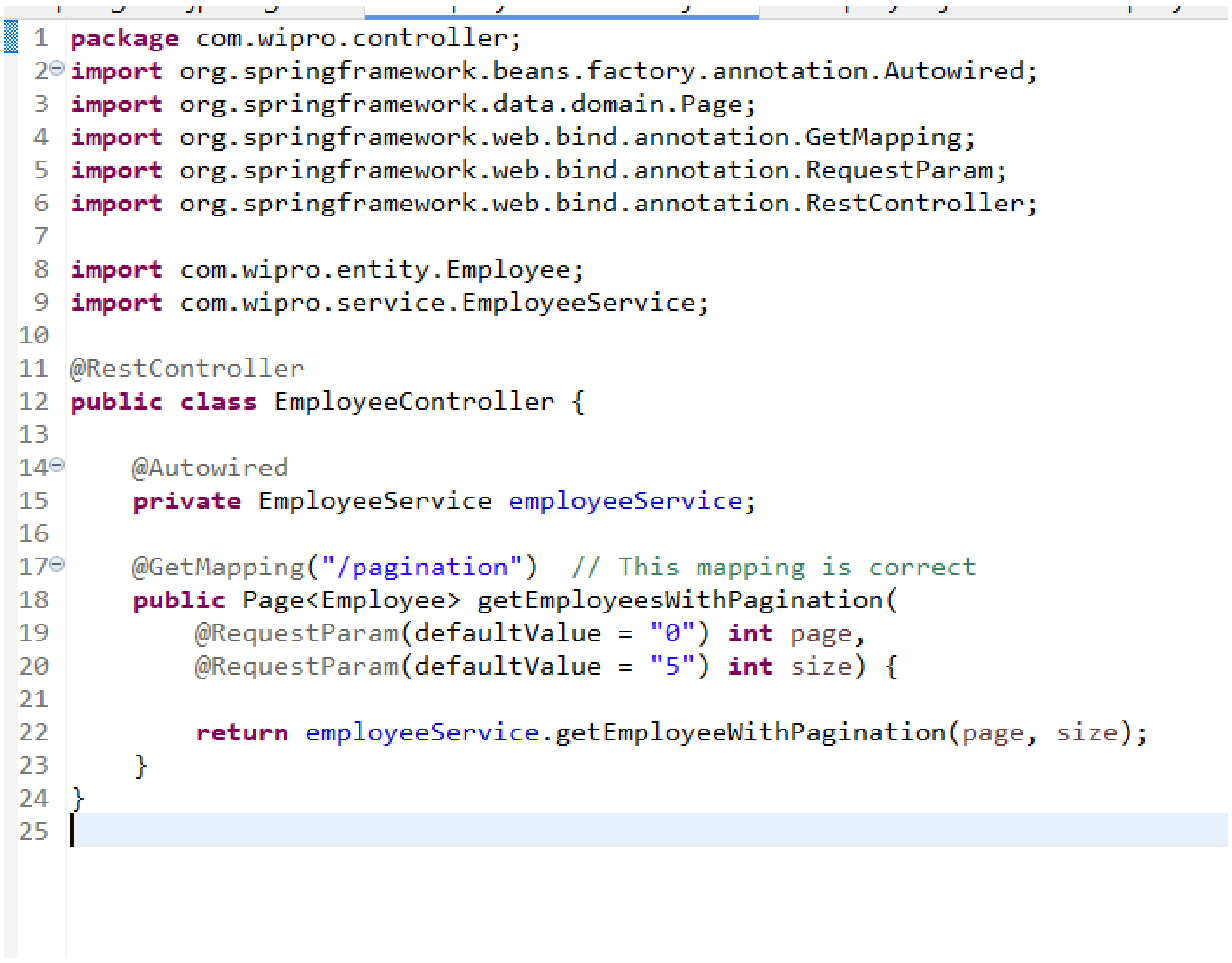
```
springdatajpa-pagination-sorting-demo
  src/main/java
    com.wipro
      SpringdatajpaPaginationSortingDemoApplication.java
    com.wipro.controller
      EmployeeController.java
    com.wipro.entity
      Employee.java
    com.wipro.repository
      EmployeeRepository.java
    com.wipro.service
      EmployeeService.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```



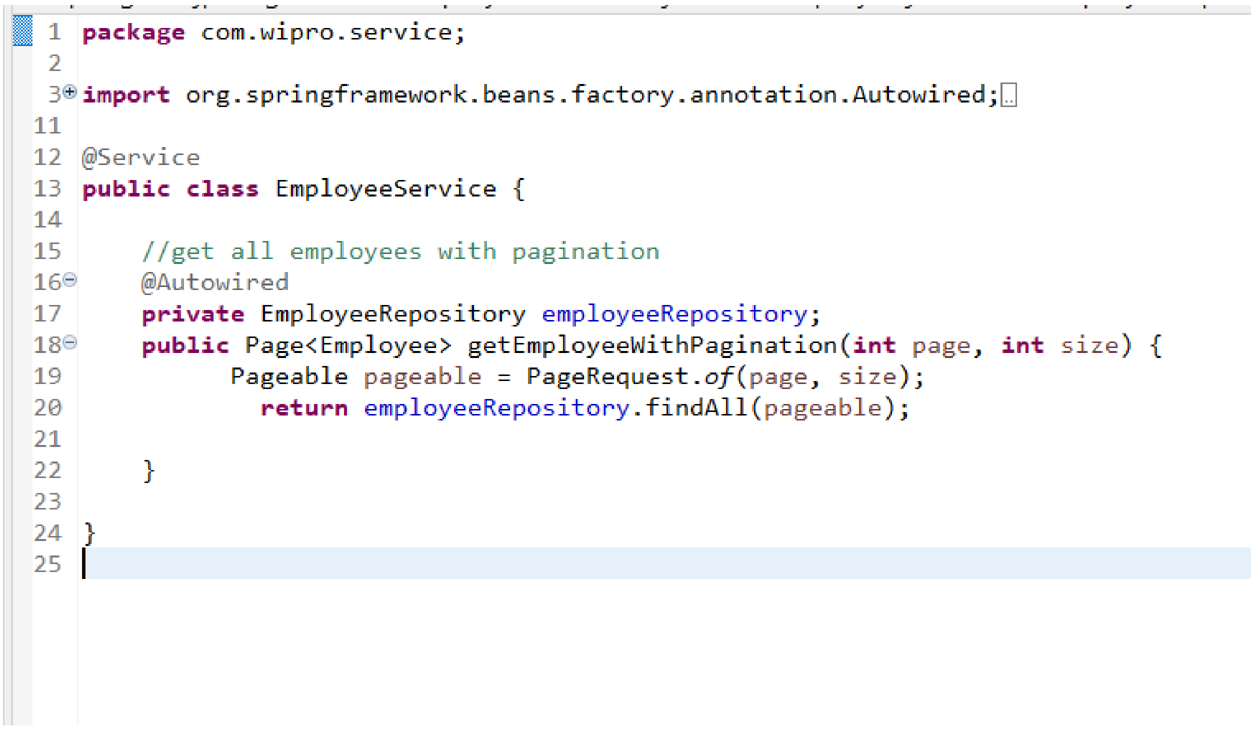
```
spring.application.name=springdatajpa-pagination-sorting-demo

server.port=9090

spring.datasource.url=jdbc:mysql://localhost:3306/wipro
spring.datasource.username=root
spring.datasource.password=#Mahadev7
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

```



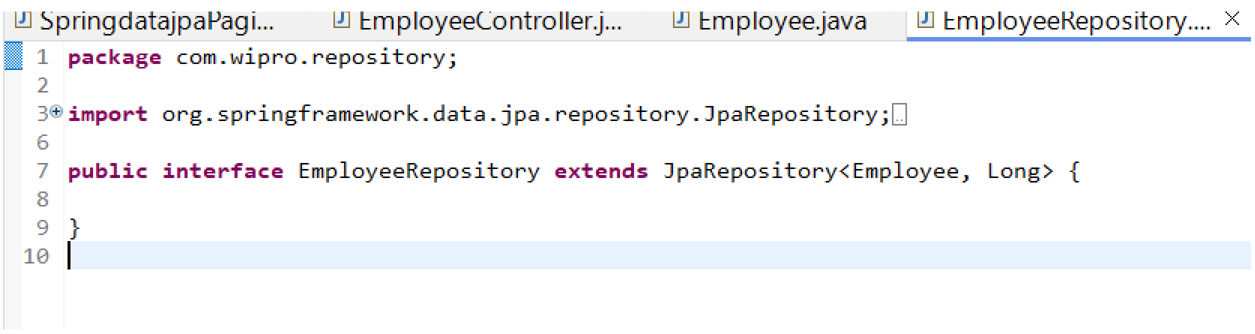
```java
package com.wipro.entity;

import jakarta.persistence.Entity;

@Entity
public class Employee {

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

    private String name;
    private String department;

    // Default constructor
    public Employee() {}

    // Parameterized constructor (without ID since it's auto-generated)
    public Employee(String name, String department) {
        this.name = name;
        this.department = department;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }
}
```



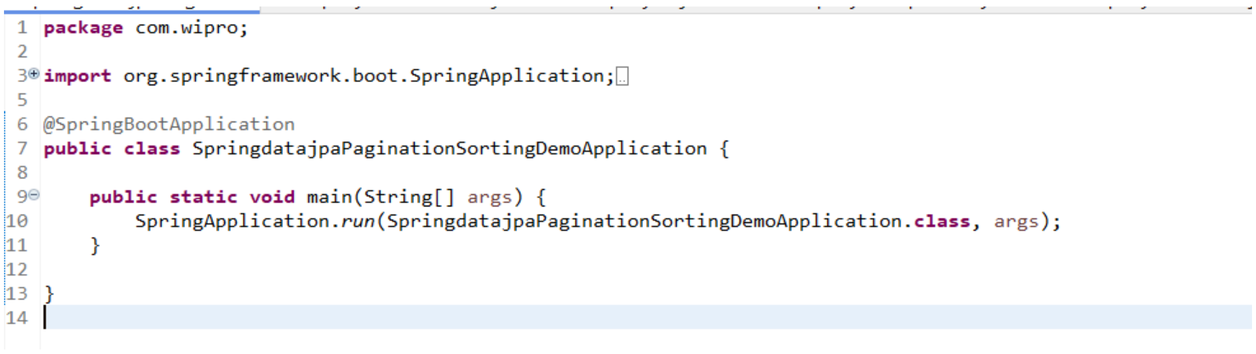
```java
package com.wipro.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.wipro.entity.Employee;
import com.wipro.service.EmployeeService;

@RestController
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @GetMapping("/pagination")  // This mapping is correct
    public Page<Employee> getEmployeesWithPagination(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "5") int size) {

        return employeeService.getEmployeeWithPagination(page, size);
    }
}

```



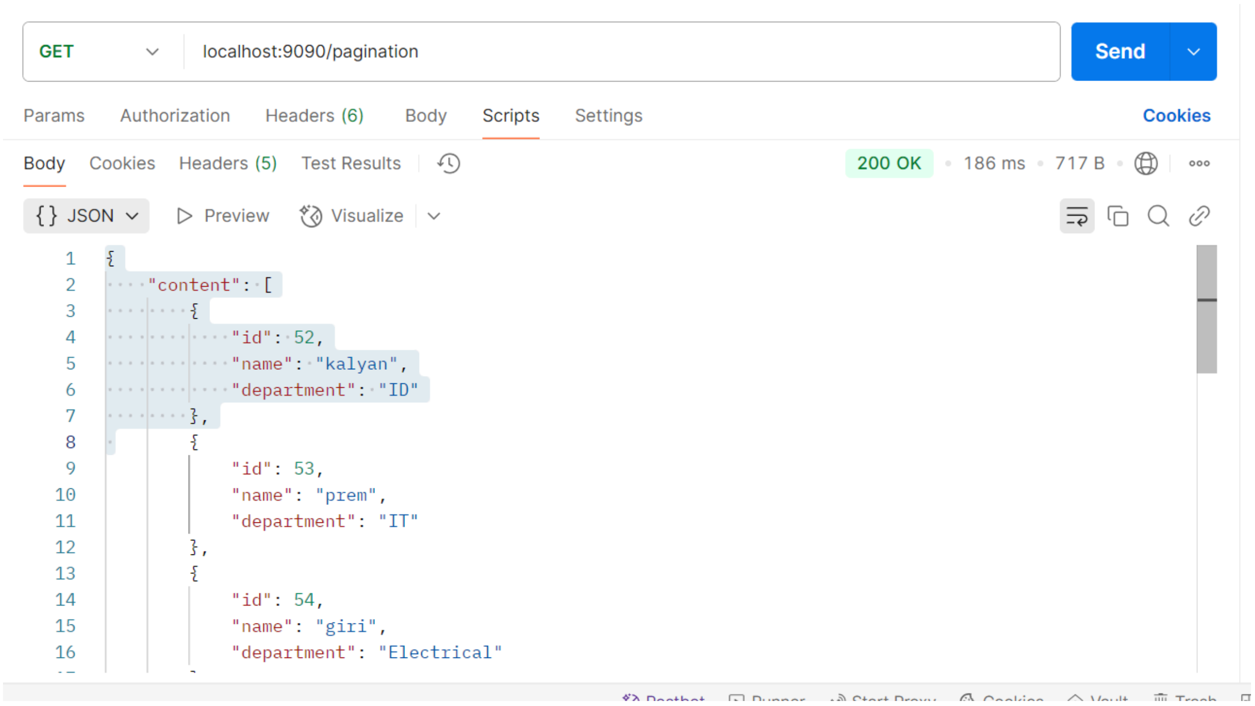
```java
package com.wipro.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class EmployeeService {

    //get all employees with pagination
    @Autowired
    private EmployeeRepository employeeRepository;
    public Page<Employee> getEmployeeWithPagination(int page, int size) {
          Pageable pageable = PageRequest.of(page, size);
            return employeeRepository.findAll(pageable);

    }

}

```



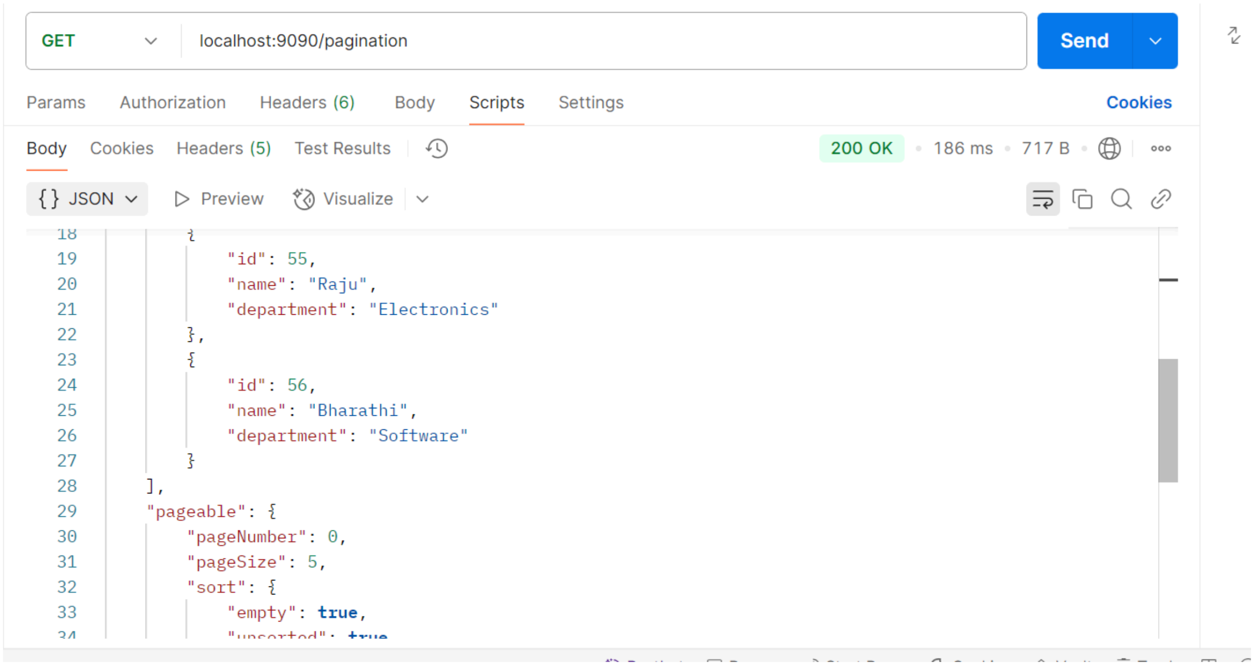
```java
package com.wipro.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```



```java
package com.wipro;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringdatajpaPaginationSortingDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringdatajpaPaginationSortingDemoApplication.class, args);
    }

}
```



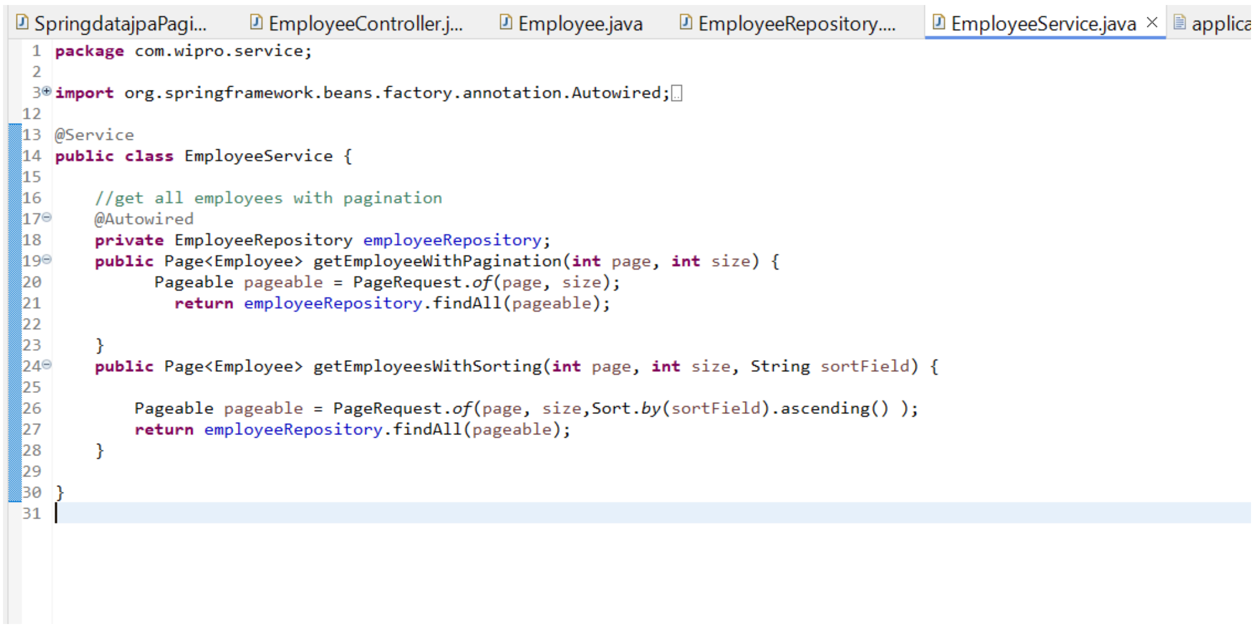
```json
{
    "content": [
        {
            "id": 52,
            "name": "kalyan",
            "department": "ID"
        },
        {
            "id": 53,
            "name": "prem",
            "department": "IT"
        },
        {
            "id": 54,
            "name": "giri",
            "department": "Electrical"
```



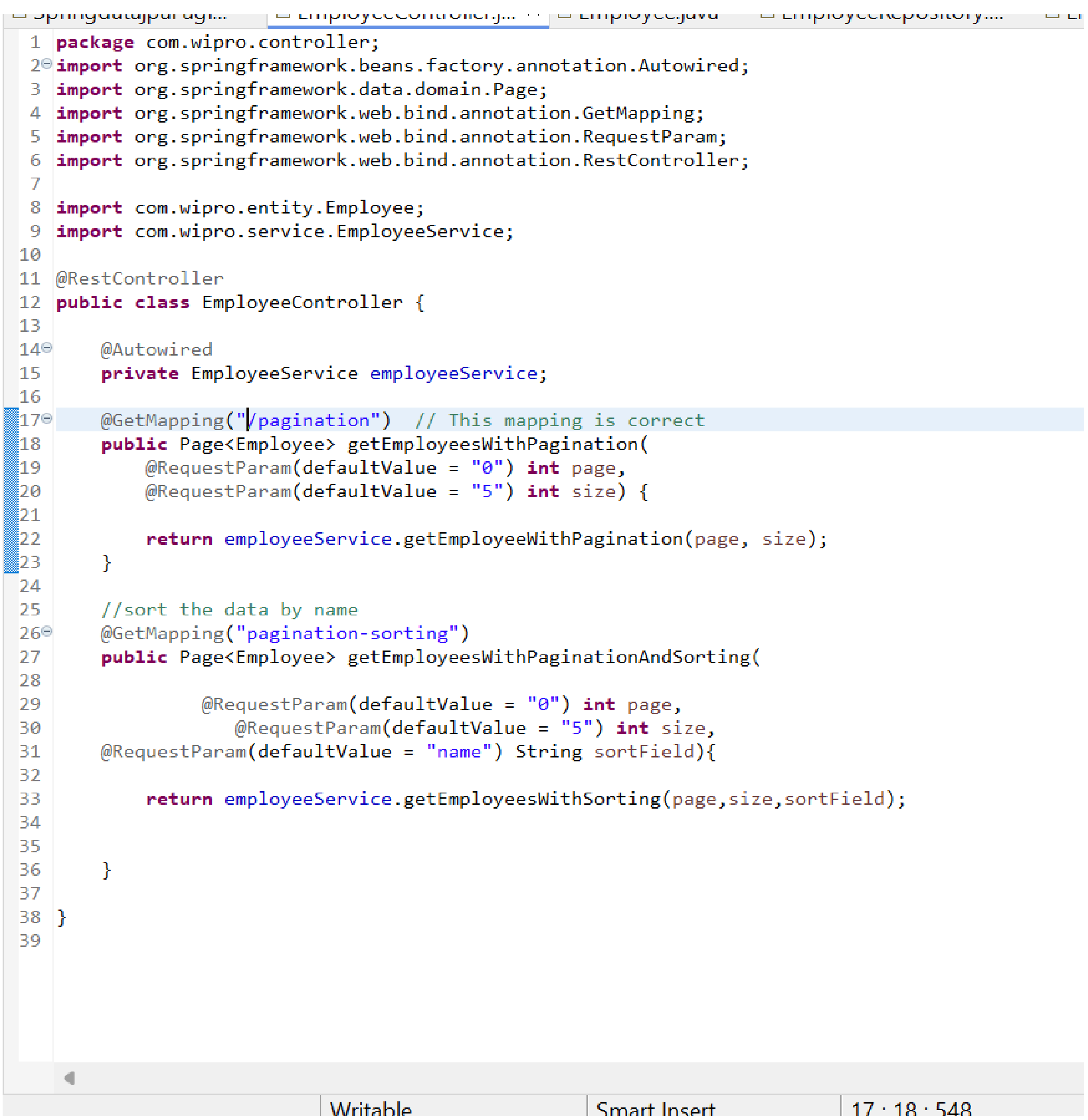
```
        {
            "id": 55,
            "name": "Raju",
            "department": "Electronics"
        },
        {
            "id": 56,
            "name": "Bharathi",
            "department": "Software"
        }
    ],
    "pageable": {
        "pageNumber": 0,
        "pageSize": 5,
        "sort": {
            "empty": true,
```

Sort the data by name

Now all the codes is same i changed only few

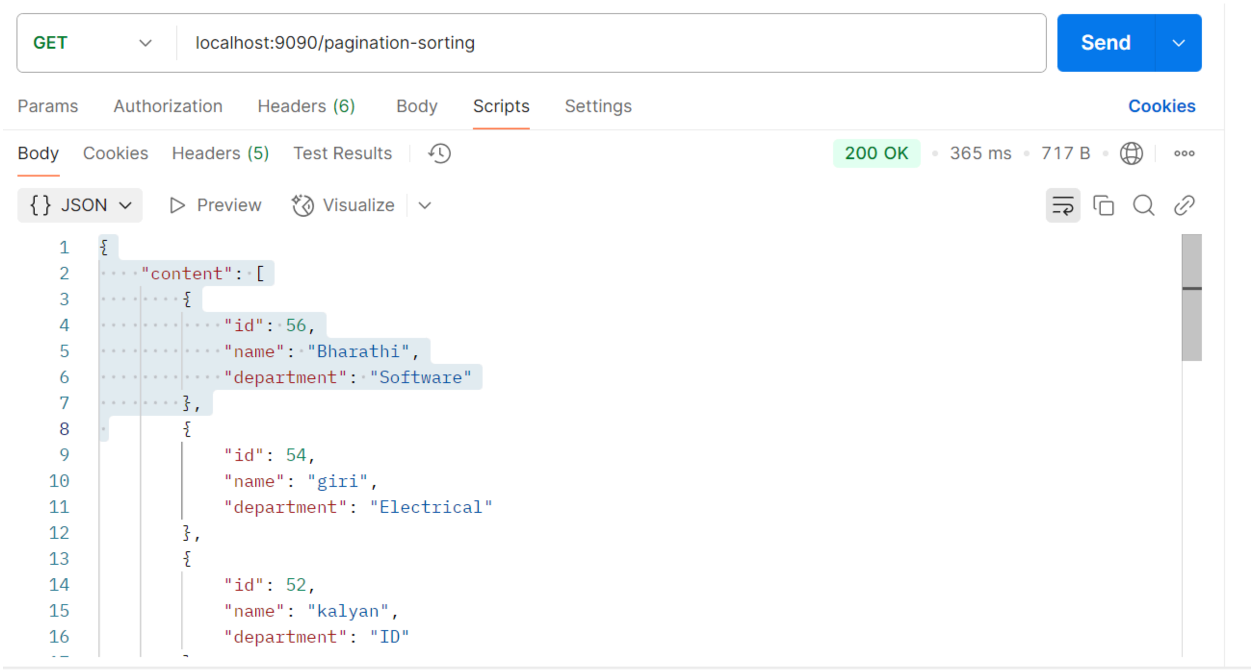


```java
package com.wipro.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class EmployeeService {

    //get all employees with pagination
    @Autowired
    private EmployeeRepository employeeRepository;
    public Page<Employee> getEmployeeWithPagination(int page, int size) {
          Pageable pageable = PageRequest.of(page, size);
            return employeeRepository.findAll(pageable);

    }
    public Page<Employee> getEmployeesWithSorting(int page, int size, String sortField) {

        Pageable pageable = PageRequest.of(page, size,Sort.by(sortField).ascending() );
        return employeeRepository.findAll(pageable);
    }

}
```



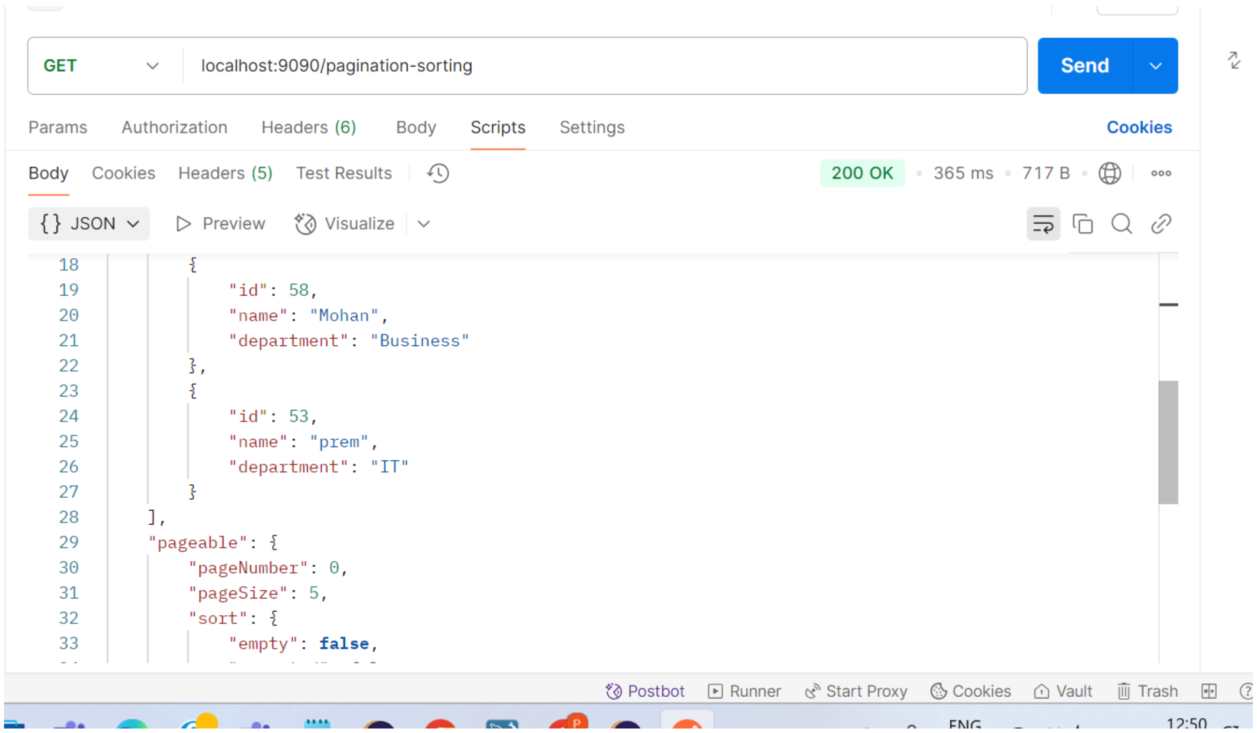
```
package com.wipro.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.wipro.entity.Employee;
import com.wipro.service.EmployeeService;

@RestController
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @GetMapping("/pagination")  // This mapping is correct
    public Page<Employee> getEmployeesWithPagination(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "5") int size) {

        return employeeService.getEmployeeWithPagination(page, size);
    }

    //sort the data by name
    @GetMapping("pagination-sorting")
    public Page<Employee> getEmployeesWithPaginationAndSorting(

             @RequestParam(defaultValue = "0") int page,
                @RequestParam(defaultValue = "5") int size,
    @RequestParam(defaultValue = "name") String sortField){

        return employeeService.getEmployeesWithSorting(page,size,sortField);


    }

}
```

Writable | Smart Insert | 17 : 18 : 548


GET localhost:9090/pagination-sorting

200 OK · 365 ms · 717 B

```
{
    "content": [
        {
            "id": 56,
            "name": "Bharathi",
            "department": "Software"
        },
        {
            "id": 54,
            "name": "giri",
            "department": "Electrical"
        },
        {
            "id": 52,
            "name": "kalyan",
            "department": "ID"
```


GET localhost:9090/pagination-sorting

200 OK · 365 ms · 717 B

```
        {
            "id": 58,
            "name": "Mohan",
            "department": "Business"
        },
        {
            "id": 53,
            "name": "prem",
            "department": "IT"
        }
    ],
    "pageable": {
        "pageNumber": 0,
        "pageSize": 5,
        "sort": {
            "empty": false,
```

Spring Boot Exception handling

Spring Boot provides several ways to handle exceptions in a **structured and user-friendly** manner. Exception handling ensures that **when something goes wrong, the application provides meaningful responses instead of crashing.**

1. Using `@ExceptionHandler` (Method-Level Handling)

You can use the `@ExceptionHandler` annotation in a specific controller to handle exceptions related to that controller's methods.

- **It allows catching specific exceptions and returning custom responses.**
- **Best suited when exception handling logic is unique to a particular controller.**

---

2. Using `@ControllerAdvice` (Global Exception Handling)

To handle exceptions globally across all controllers, use `@ControllerAdvice`.

- **A centralized way to handle exceptions across multiple controllers.**
- **It allows defining exception handlers in a separate class, making the code cleaner.**
- **Ensures uniform error responses for the entire application.**

---

3. Using `ResponseStatusException` (Quick Exception Handling)

You can also throw exceptions directly using `ResponseStatusException`.

- **A simple way to throw exceptions with an associated HTTP status code.**
- **Useful for handling cases where you want to return an error response without writing separate exception handler methods.**
- **Often used for standard HTTP errors like `404 NOT FOUND` or `400 BAD REQUEST`.**

---

4. Customizing Error Responses with `ErrorDetails`

Instead of returning plain strings, you can return structured error responses.

- **Instead of returning plain error messages, you can define a structured format with fields like timestamp, error code, and details.**
- **Helps improve API documentation and debugging.**
- **Ensures consistency in error handling across different endpoints.**

5. Handling Validation Errors (`@Valid` & `@Validated`)

If you are using `@Valid` for request validation, you can handle `MethodArgumentNotValidException`.

- **This allows returning detailed field-specific validation messages instead of generic error responses.**

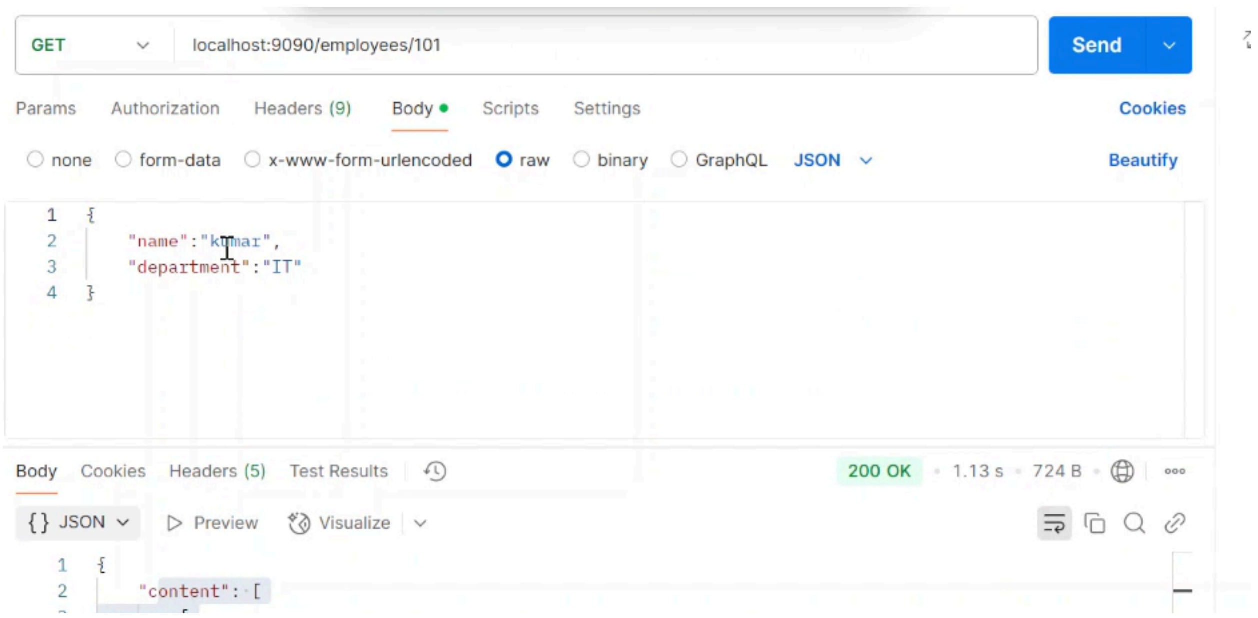
---

6. Handling 404 Errors (`@ResponseStatus`)

You can create a custom exception with `@ResponseStatus`.

- **This ensures that when a specific exception is thrown, it automatically returns the appropriate HTTP status without requiring additional handler methods.**

Spring boot exception handling afternoon


GET localhost:9090/employees/101 Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

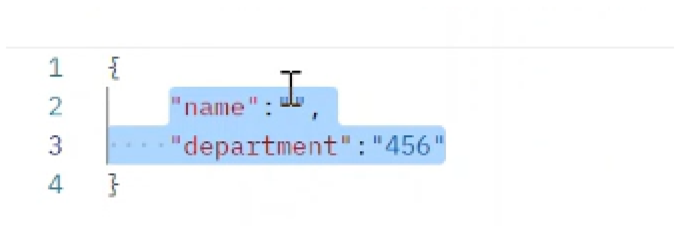
```
{
    "name":"kumar",
    "department":"IT"
}
```

Body Cookies Headers (5) Test Results 200 OK 1.13 s 724 B

{} JSON Preview Visualize

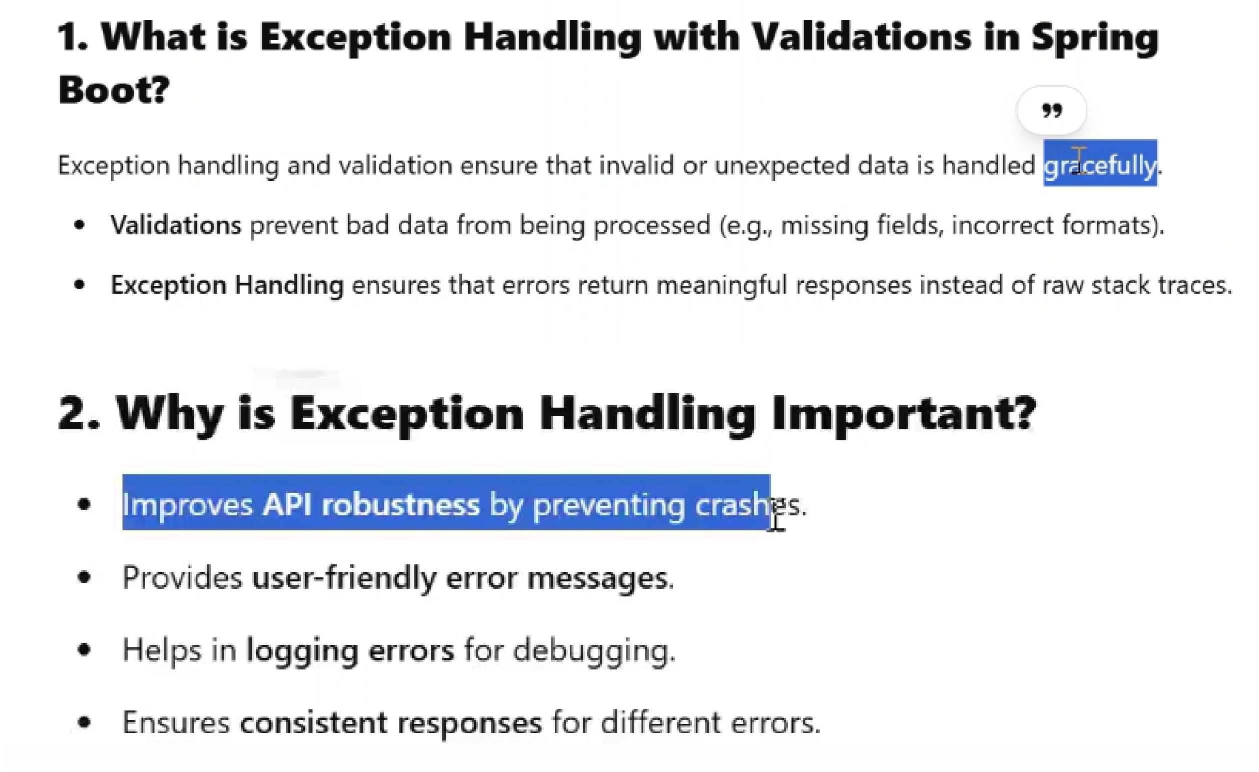
```
{
    "content": [
```

For suppose at the time of updating the data ...if we miss the name and dept by leaving empty then it leads to an error ..so at that time we need to mention the error in the console clearly to the end user



```
{
    "name":"",
    "department":"456"
}
```

It is also the invalid data..it should not hit the server..before hitting the server we need to mention the error like name cannot be blank and dept should be alphabetical to the end user during the rest api calls



1. What is Exception Handling with Validations in Spring Boot?

Exception handling and validation ensure that invalid or unexpected data is handled gracefully.

- **Validations** prevent bad data from being processed (e.g., missing fields, incorrect formats).
- **Exception Handling** ensures that errors return meaningful responses instead of raw stack traces.

2. Why is Exception Handling Important?

- Improves **API robustness** by preventing crashes.
- Provides **user-friendly error messages**.
- Helps in **logging errors** for debugging.
- Ensures **consistent responses** for different errors.

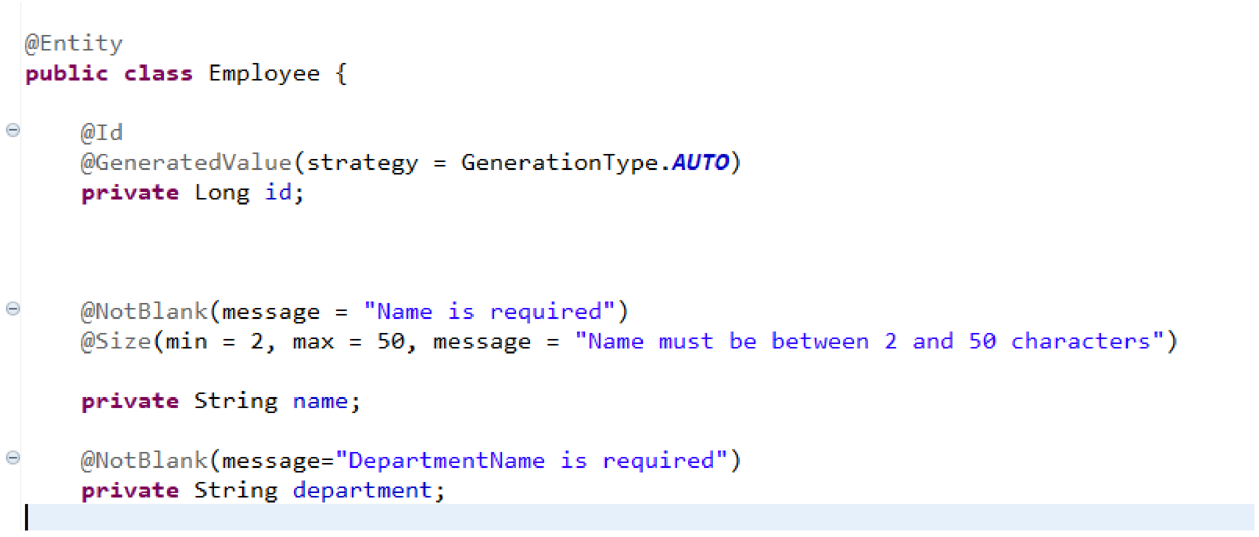
We have to add the spring boot starter validation into our pom.xml file

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
```

Now add the below code into the Entity class

@NotBlank(message = "Name is required")
@Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

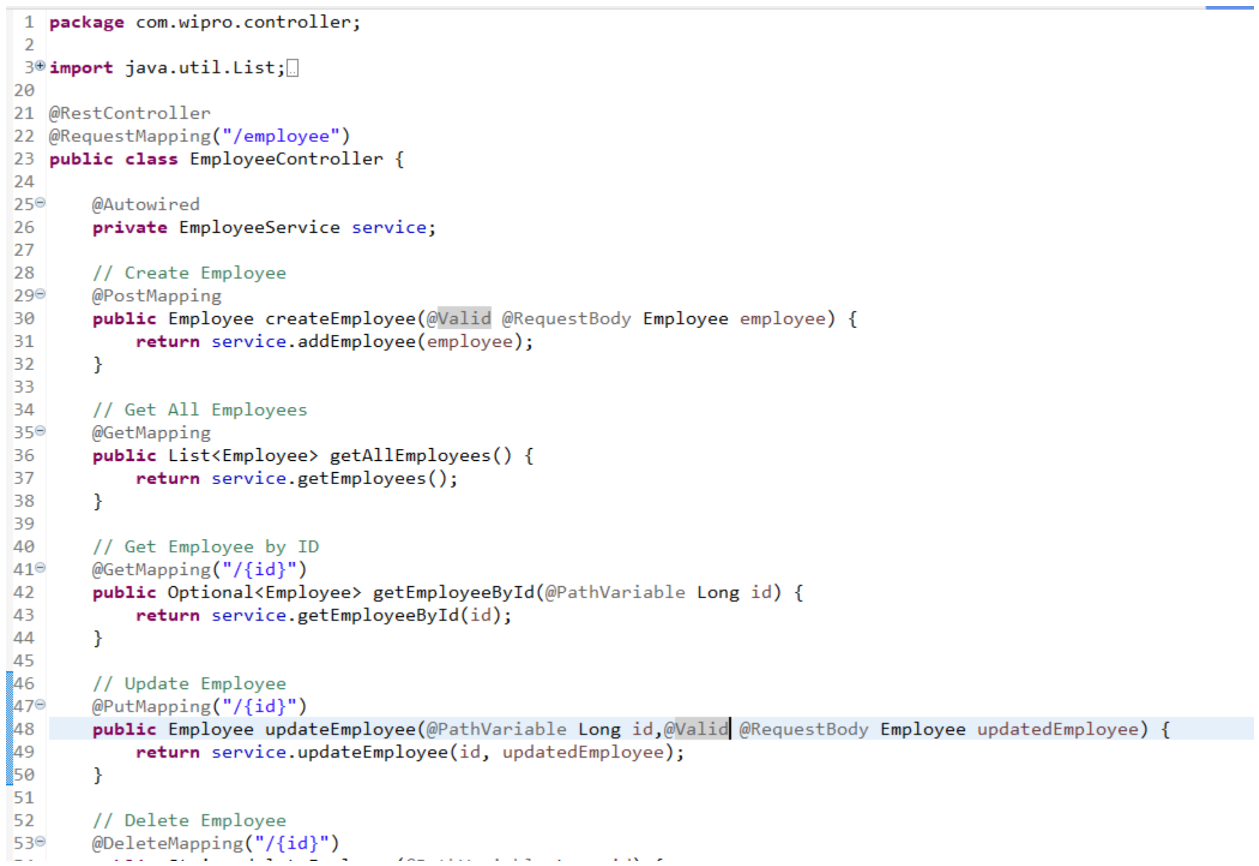
Successfully added



```java
@Entity
public class Employee {

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



    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

    private String name;

    @NotBlank(message="DepartmentName is required")
    private String department;
```

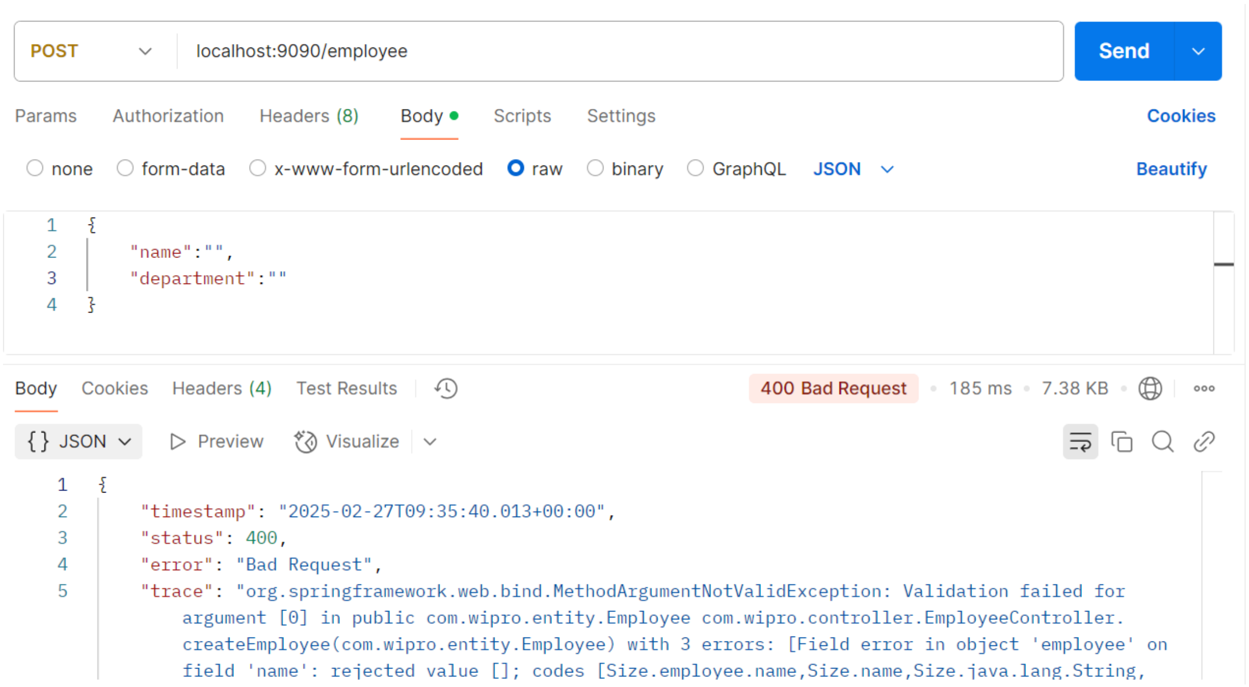
Name and dept we have added the annotations

Now apply the @valid in the employeecontroller



```java
package com.wipro.controller;

import java.util.List;

@RestController
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService service;

    // Create Employee
    @PostMapping
    public Employee createEmployee(@Valid @RequestBody Employee employee) {
        return service.addEmployee(employee);
    }

    // Get All Employees
    @GetMapping
    public List<Employee> getAllEmployees() {
        return service.getEmployees();
    }

    // Get Employee by ID
    @GetMapping("/{id}")
    public Optional<Employee> getEmployeeById(@PathVariable Long id) {
        return service.getEmployeeById(id);
    }

    // Update Employee
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id,@Valid @RequestBody Employee updatedEmployee) {
        return service.updateEmployee(id, updatedEmployee);
    }

    // Delete Employee
    @DeleteMapping("/{id}")
```

Now run the application


POST localhost:9090/employee

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON Beautify

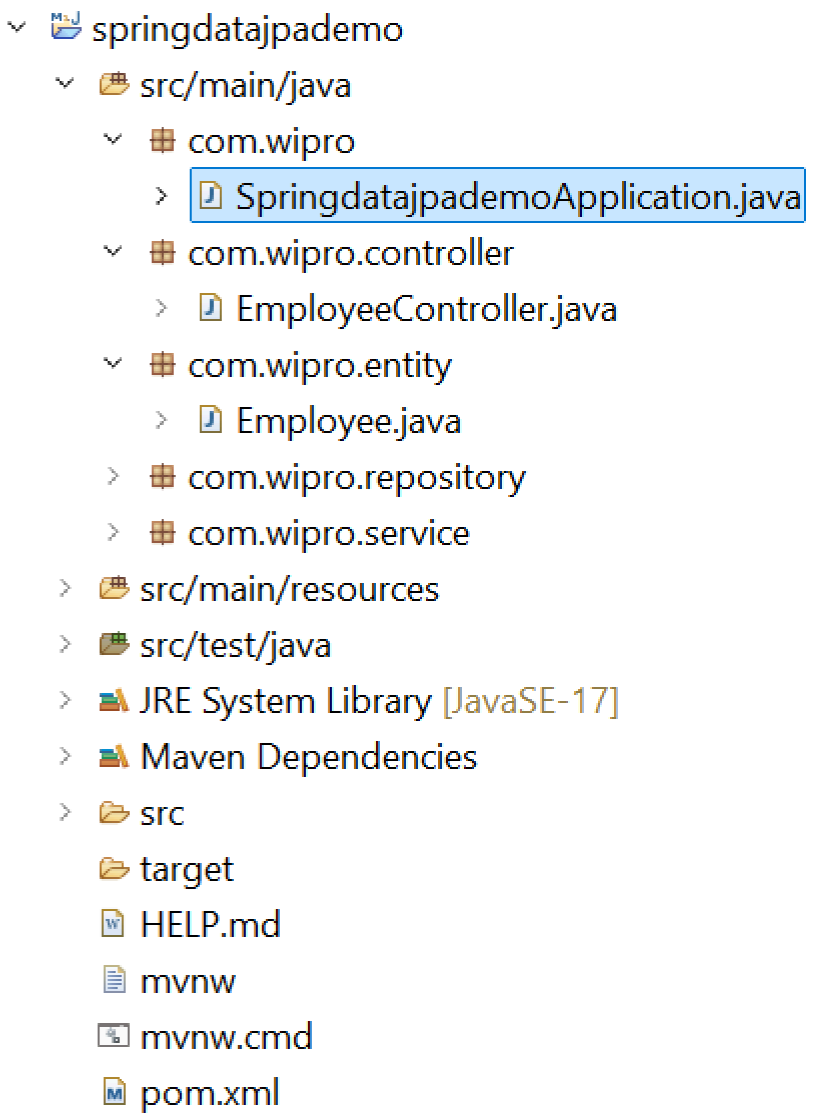
```
{
    "name":"",
    "department":""
}
```

Body Cookies Headers (4) Test Results 400 Bad Request 185 ms 7.38 KB

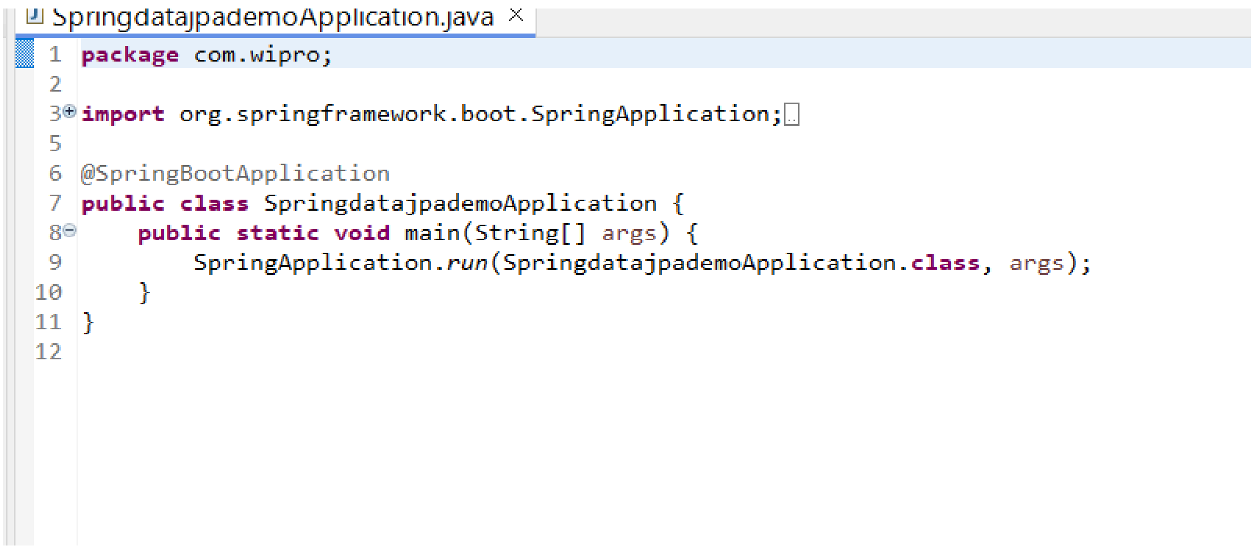
{ } JSON Preview Visualize

```
{
    "timestamp": "2025-02-27T09:35:40.013+00:00",
    "status": 400,
    "error": "Bad Request",
    "trace": "org.springframework.web.bind.MethodArgumentNotValidException: Validation failed for
        argument [0] in public com.wipro.entity.Employee com.wipro.controller.EmployeeController.
        createEmployee(com.wipro.entity.Employee) with 3 errors: [Field error in object 'employee' on
        field 'name': rejected value []; codes [Size.employee.name,Size.name,Size.java.lang.String,
```

Now i will provide u the overall codes

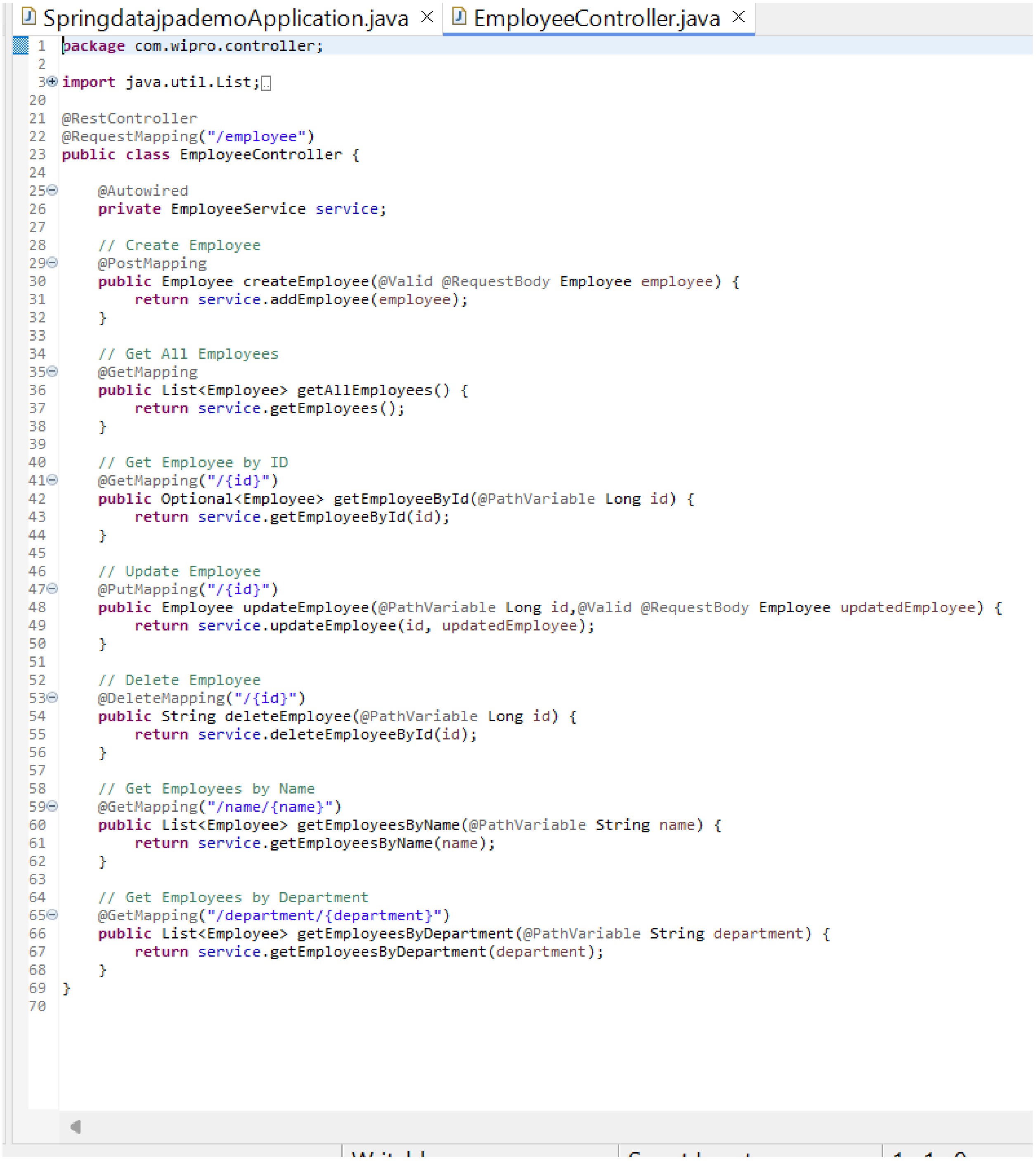


```
springdatajpademo
  src/main/java
    com.wipro
      SpringdatajpademoApplication.java
    com.wipro.controller
      EmployeeController.java
    com.wipro.entity
      Employee.java
    com.wipro.repository
    com.wipro.service
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

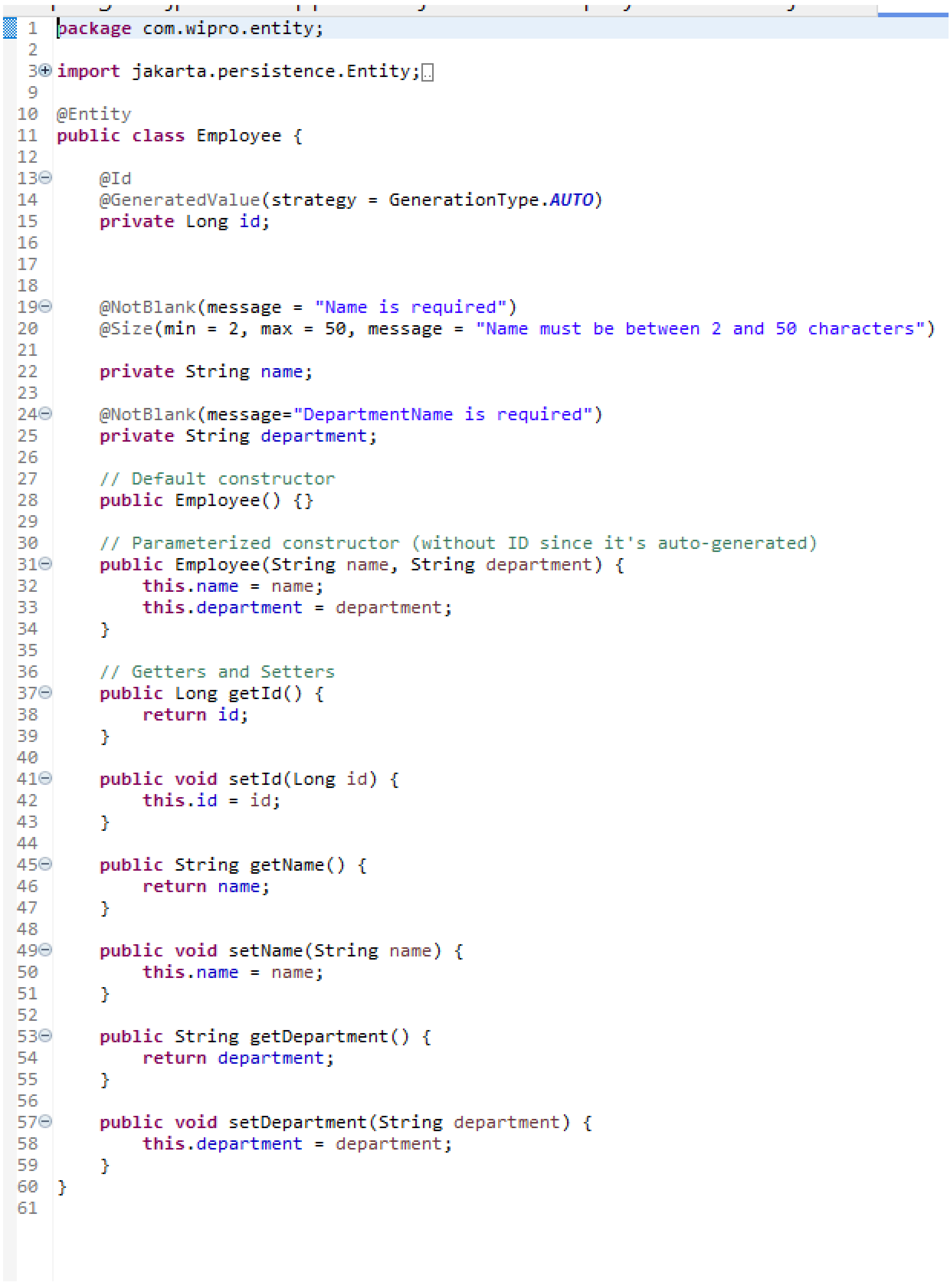


SpringdatajpademoApplication.java

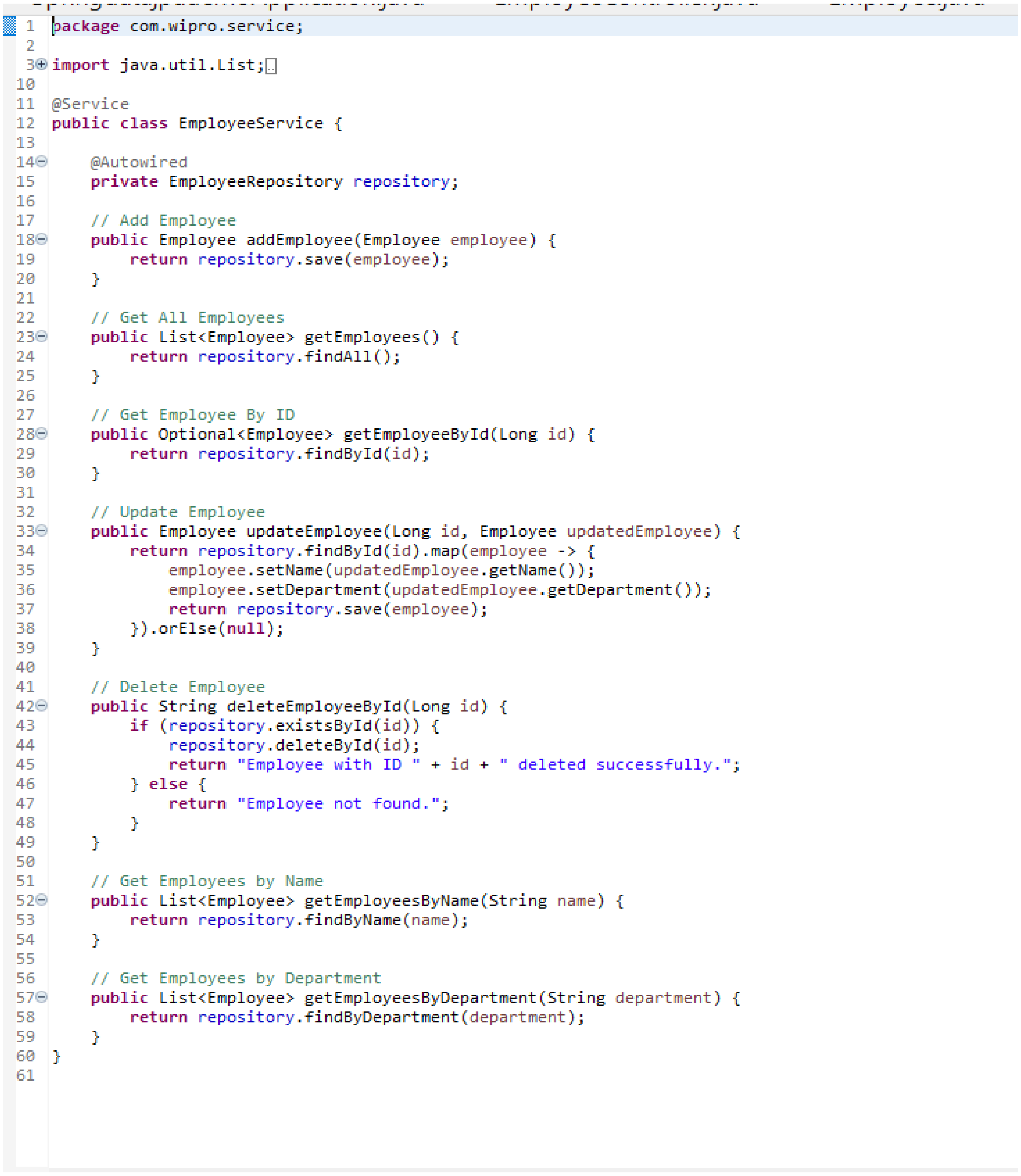
```java
package com.wipro;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringdatajpademoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringdatajpademoApplication.class, args);
    }
}

```



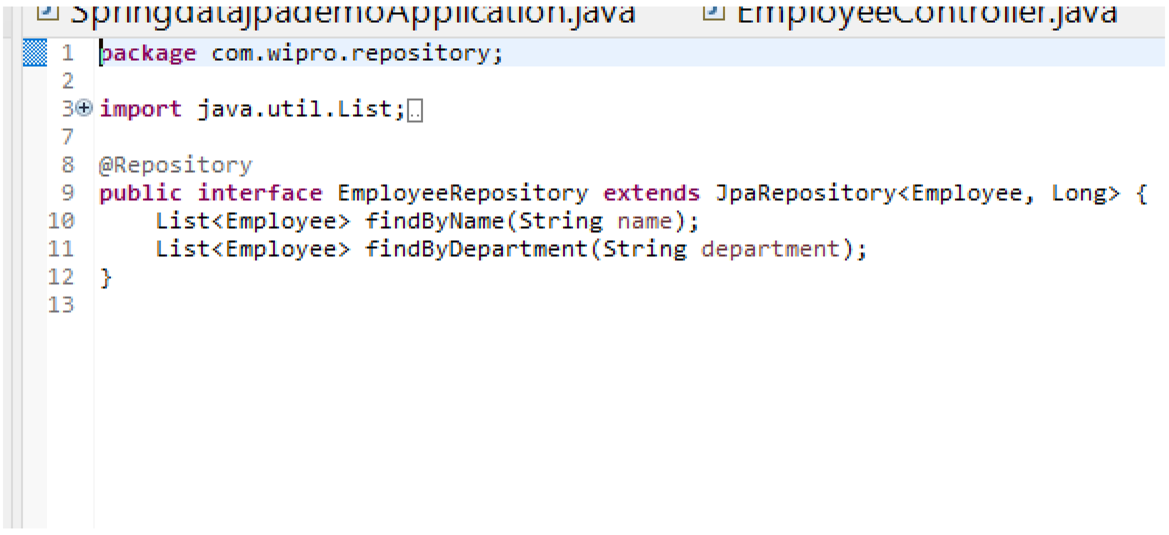
```java
package com.wipro.controller;

import java.util.List;

@RestController
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService service;

    // Create Employee
    @PostMapping
    public Employee createEmployee(@Valid @RequestBody Employee employee) {
        return service.addEmployee(employee);
    }

    // Get All Employees
    @GetMapping
    public List<Employee> getAllEmployees() {
        return service.getEmployees();
    }

    // Get Employee by ID
    @GetMapping("/{id}")
    public Optional<Employee> getEmployeeById(@PathVariable Long id) {
        return service.getEmployeeById(id);
    }

    // Update Employee
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id,@Valid @RequestBody Employee updatedEmployee) {
        return service.updateEmployee(id, updatedEmployee);
    }

    // Delete Employee
    @DeleteMapping("/{id}")
    public String deleteEmployee(@PathVariable Long id) {
        return service.deleteEmployeeById(id);
    }

    // Get Employees by Name
    @GetMapping("/name/{name}")
    public List<Employee> getEmployeesByName(@PathVariable String name) {
        return service.getEmployeesByName(name);
    }

    // Get Employees by Department
    @GetMapping("/department/{department}")
    public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
        return service.getEmployeesByDepartment(department);
    }
}
```



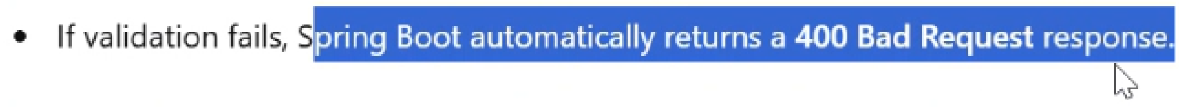
```java
package com.wipro.entity;

import jakarta.persistence.Entity;

@Entity
public class Employee {

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



    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

    private String name;

    @NotBlank(message="DepartmentName is required")
    private String department;

    // Default constructor
    public Employee() {}

    // Parameterized constructor (without ID since it's auto-generated)
    public Employee(String name, String department) {
        this.name = name;
        this.department = department;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }
}
```



```java
package com.wipro.service;

import java.util.List;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    // Add Employee
    public Employee addEmployee(Employee employee) {
        return repository.save(employee);
    }

    // Get All Employees
    public List<Employee> getEmployees() {
        return repository.findAll();
    }

    // Get Employee By ID
    public Optional<Employee> getEmployeeById(Long id) {
        return repository.findById(id);
    }

    // Update Employee
    public Employee updateEmployee(Long id, Employee updatedEmployee) {
        return repository.findById(id).map(employee -> {
            employee.setName(updatedEmployee.getName());
            employee.setDepartment(updatedEmployee.getDepartment());
            return repository.save(employee);
        }).orElse(null);
    }

    // Delete Employee
    public String deleteEmployeeById(Long id) {
        if (repository.existsById(id)) {
            repository.deleteById(id);
            return "Employee with ID " + id + " deleted successfully.";
        } else {
            return "Employee not found.";
        }
    }

    // Get Employees by Name
    public List<Employee> getEmployeesByName(String name) {
        return repository.findByName(name);
    }

    // Get Employees by Department
    public List<Employee> getEmployeesByDepartment(String department) {
        return repository.findByDepartment(department);
    }
}
```



```
package com.wipro.repository;

import java.util.List;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByName(String name);
    List<Employee> findByDepartment(String department);
}

```



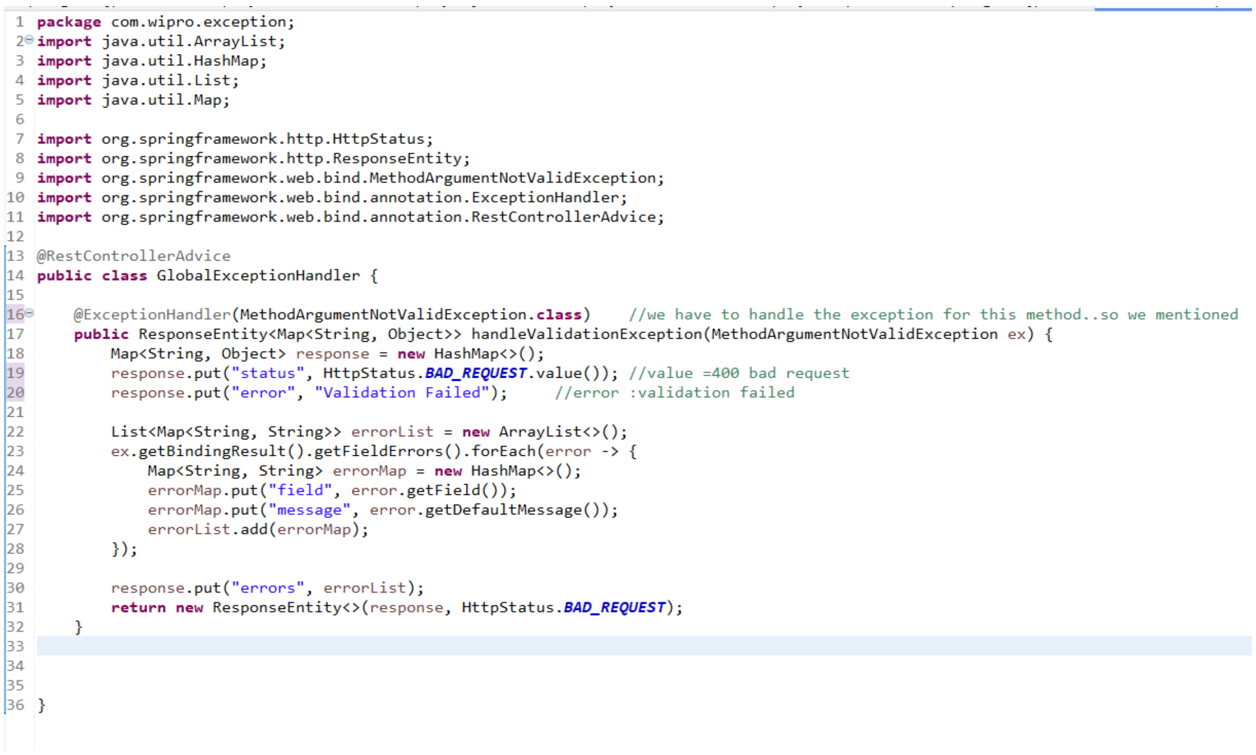
- If validation fails, Spring Boot automatically returns a **400 Bad Request** response.

Now we need to handle that method

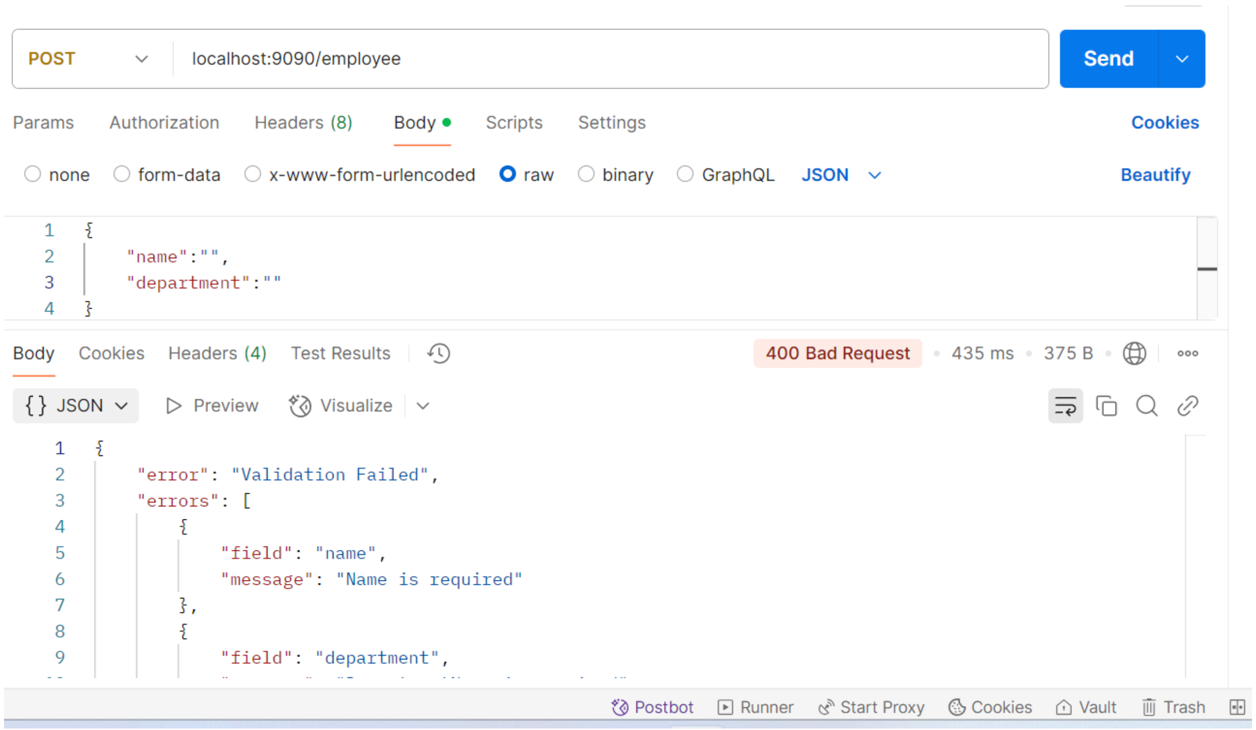
We dont want the error message…as a user needs a friendly msg instead of this

Now we need to create the package named as **exception**

Now we should define the global exception handler class in the above package


```java
package com.wipro.exception;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)    //we have to handle the exception for this method..so we mentioned
    public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("status", HttpStatus.BAD_REQUEST.value()); //value =400 bad request
        response.put("error", "Validation Failed");     //error :validation failed

        List<Map<String, String>> errorList = new ArrayList<>();
        ex.getBindingResult().getFieldErrors().forEach(error -> {
            Map<String, String> errorMap = new HashMap<>();
            errorMap.put("field", error.getField());
            errorMap.put("message", error.getDefaultMessage());
            errorList.add(errorMap);
        });

        response.put("errors", errorList);
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }



}
```

now run the application



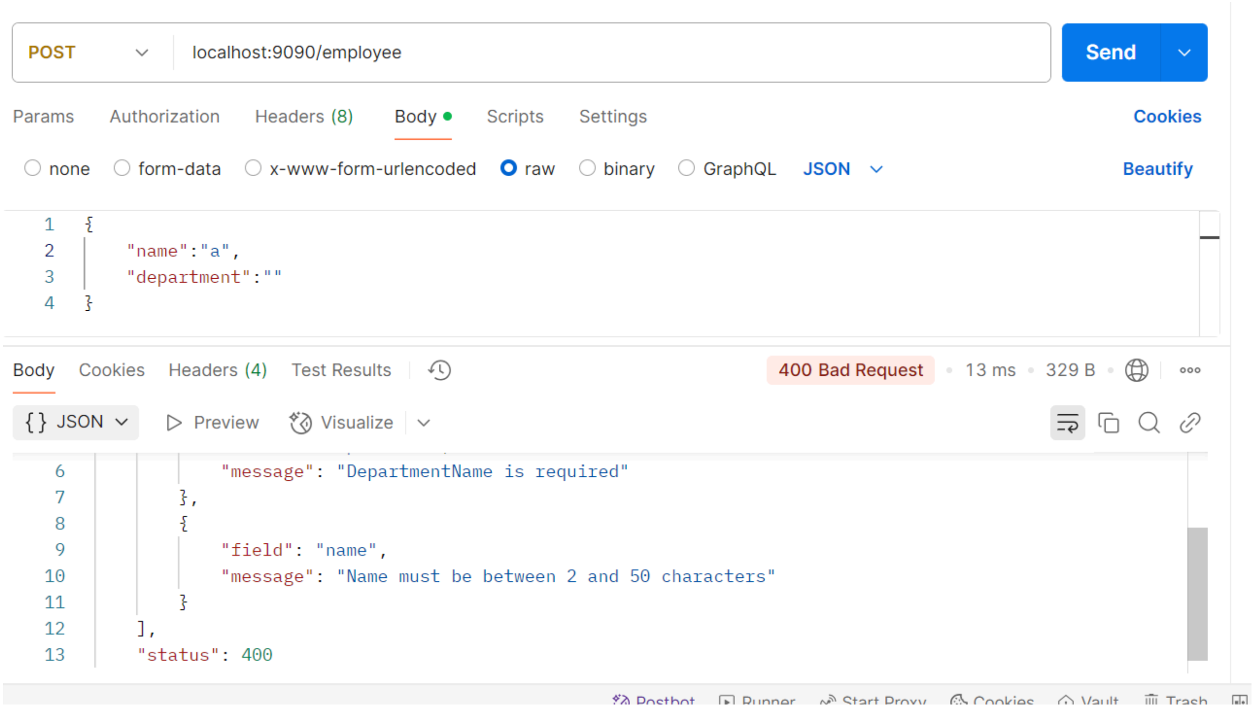
localhost:9090/employee

```json
{
    "name":"",
    "department":""
}
```

400 Bad Request • 435 ms • 375 B

```json
{
    "error": "Validation Failed",
    "errors": [
        {
            "field": "name",
            "message": "Name is required"
        },
        {
            "field": "department",
```

If we mention name:a



localhost:9090/employee

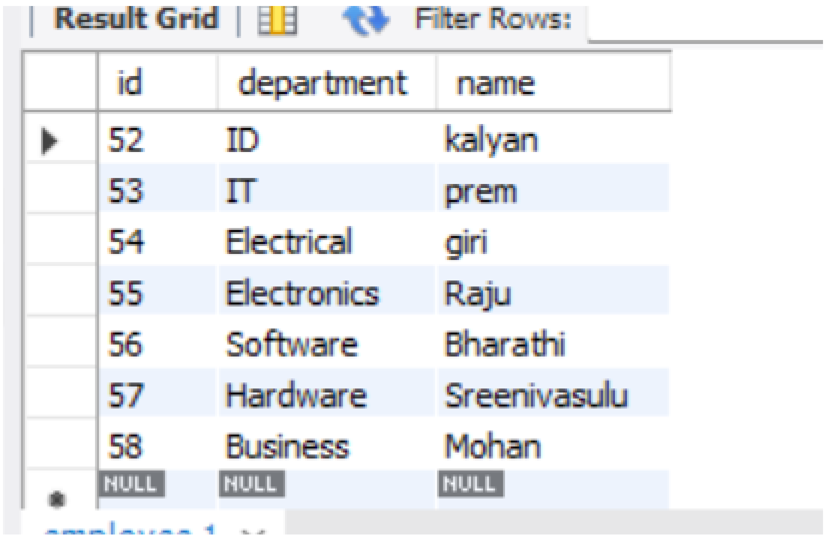
```
{
    "name":"a",
    "department":""
}
```

400 Bad Request

```
            "message": "DepartmentName is required"
        },
        {
            "field": "name",
            "message": "Name must be between 2 and 50 characters"
        }
    ],
    "status": 400
```

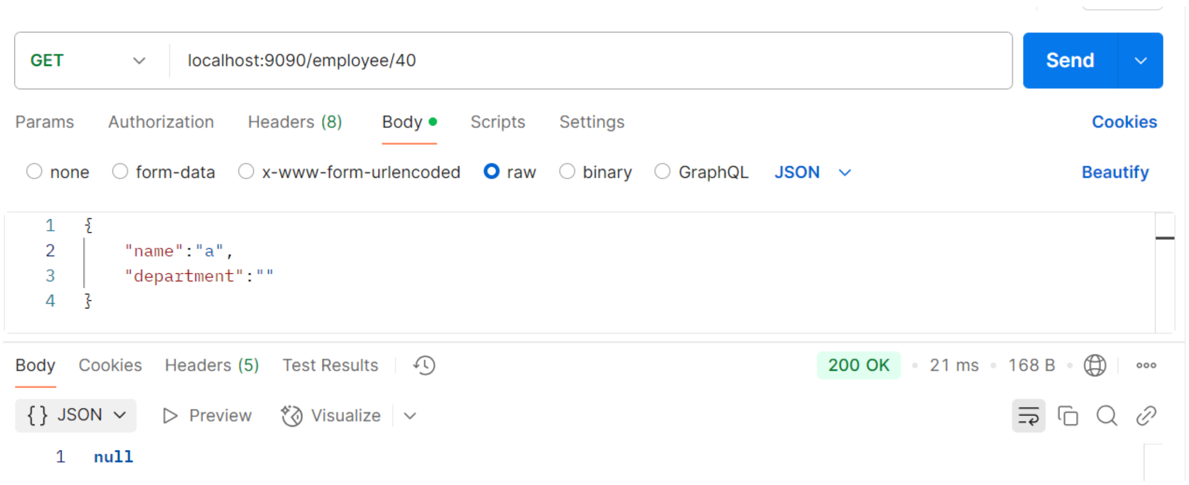
Again it is getting error y bcoz we have mentioned the name min and max size…so one letter cannot be considered

Now we r having this data in my database



id	department	name
52	ID	kalyan
53	IT	prem
54	Electrical	giri
55	Electronics	Raju
56	Software	Bharathi
57	Hardware	Sreenivasulu
58	Business	Mohan
NULL	NULL	NULL

What if i search for the id which is not exist in my database

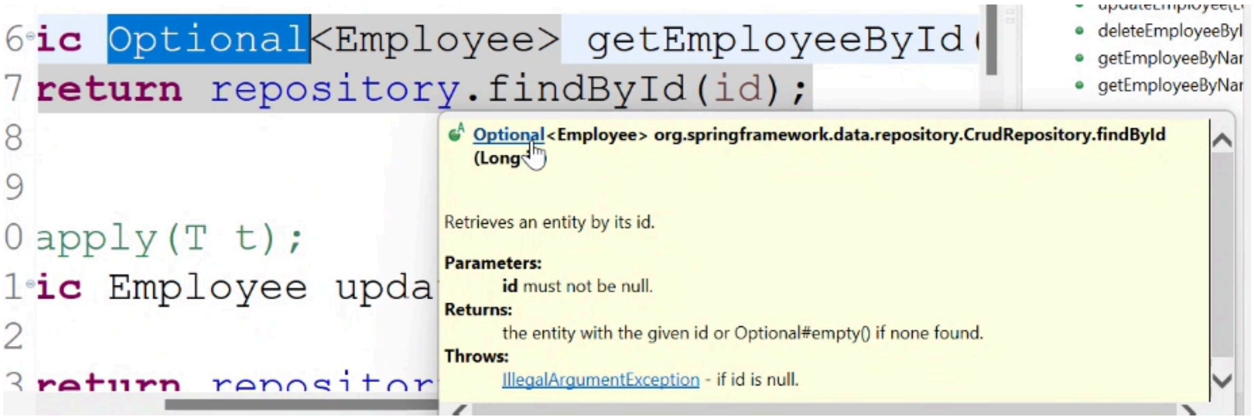


```
GET localhost:9090/employee/40
```

```
{
    "name":"a",
    "department":""
}
```

```
null
```

We r going to get the null **value in the postman…This is because our return type is optional**



```
ic Optional<Employee> getEmployeeById(
return repository.findById(id);


apply(T t);
ic Employee upda

return repositor
```

Optional<Employee> org.springframework.data.repository.CrudRepository.findById(Long

Retrieves an entity by its id.

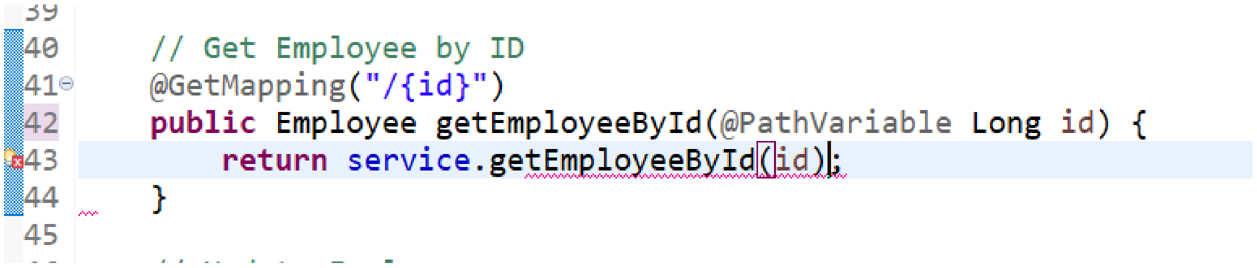
Parameters:
id must not be null.
Returns:
the entity with the given id or Optional#empty() if none found.
Throws:
IllegalArgumentException - if id is null.

If the data is there then it returns ,if the data is not there then it returns the null

It is not the better one to the end user if it is shown as output

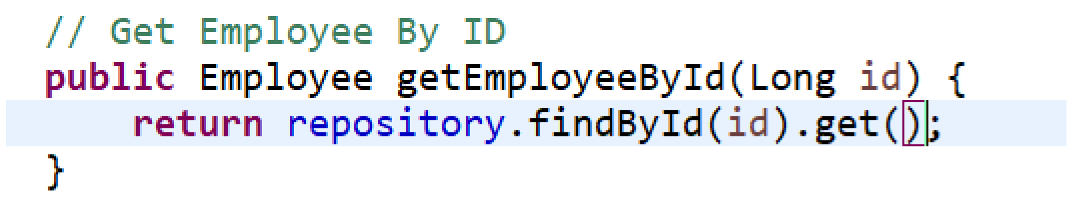
We need to handle it

Now better to remove the **optional**



```
    // Get Employee by ID
    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) {
        return service.getEmployeeById(id);
    }
```

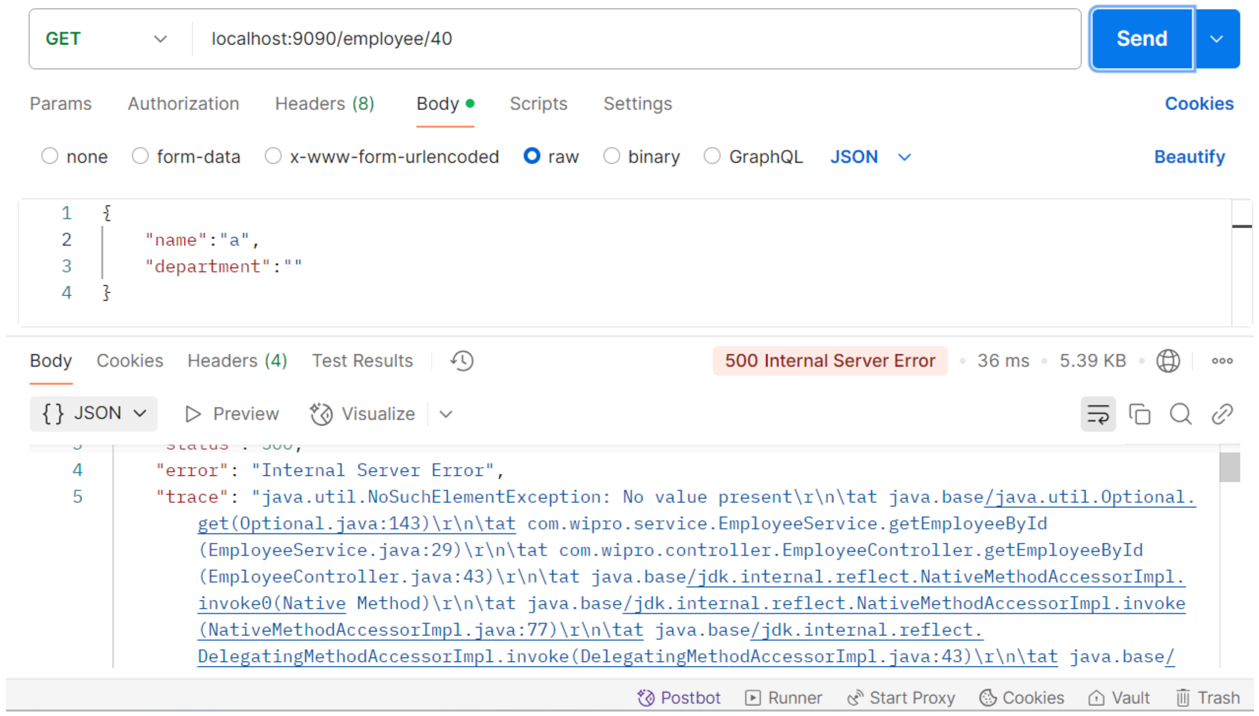
In the controller class



```java
// Get Employee By ID
public Employee getEmployeeById(Long id) {
    return repository.findById(id).get();
}
```

In the service class

Now run the application then we will get the error like this



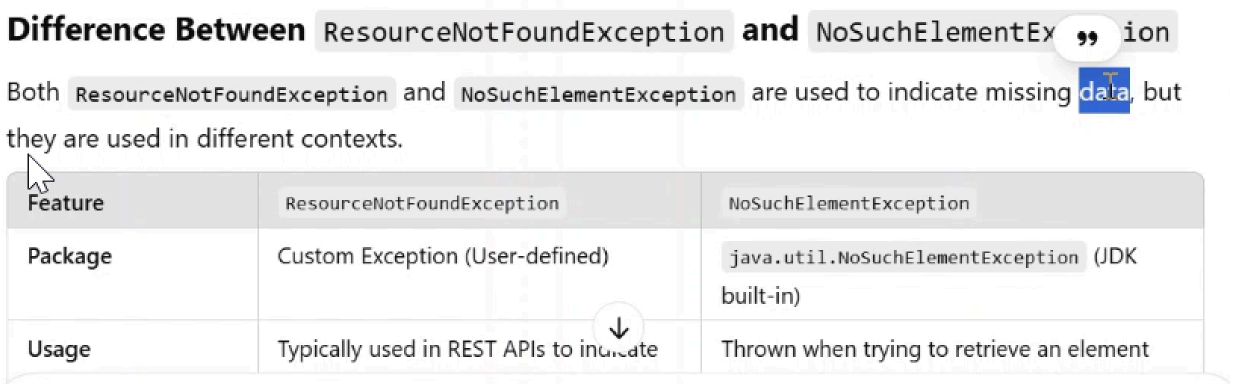
localhost:9090/employee/40

```json
{
    "name":"a",
    "department":""
}
```

500 Internal Server Error • 36 ms • 5.39 KB

```
    "status": 500,
    "error": "Internal Server Error",
    "trace": "java.util.NoSuchElementException: No value present\r\n\tat java.base/java.util.Optional.
        get(Optional.java:143)\r\n\tat com.wipro.service.EmployeeService.getEmployeeById
        (EmployeeService.java:29)\r\n\tat com.wipro.controller.EmployeeController.getEmployeeById
        (EmployeeController.java:43)\r\n\tat java.base/jdk.internal.reflect.NativeMethodAccessorImpl.
        invoke0(Native Method)\r\n\tat java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke
        (NativeMethodAccessorImpl.java:77)\r\n\tat java.base/jdk.internal.reflect.
        DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\r\n\tat java.base/
```

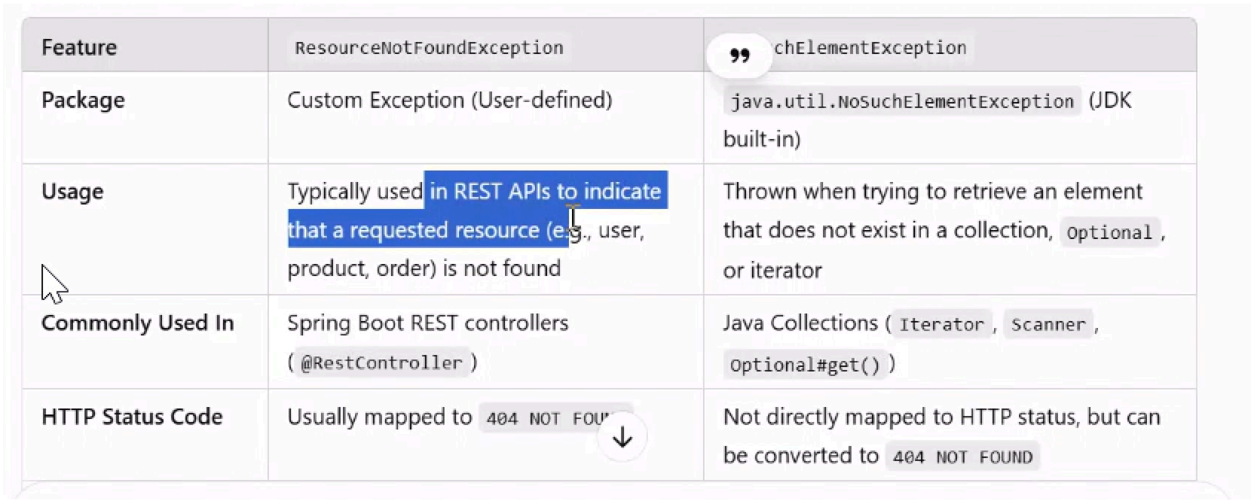
Y bcoz we removed optional so we r getting the error



Difference Between `ResourceNotFoundException` and `NoSuchElementException`

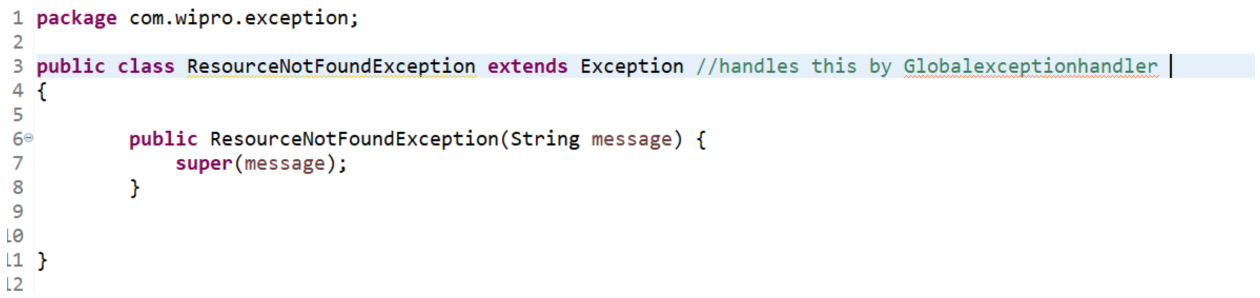
Both `ResourceNotFoundException` and `NoSuchElementException` are used to indicate missing data, but they are used in different contexts.

Feature	`ResourceNotFoundException`	`NoSuchElementException`
Package	Custom Exception (User-defined)	`java.util.NoSuchElementException` (JDK built-in)
Usage	Typically used in REST APIs to indicate	Thrown when trying to retrieve an element



Feature	`ResourceNotFoundException`	`chElementException`
Package	Custom Exception (User-defined)	`java.util.NoSuchElementException` (JDK built-in)
Usage	Typically used in REST APIs to indicate that a requested resource (e.g., user, product, order) is not found	Thrown when trying to retrieve an element that does not exist in a collection, `Optional`, or iterator
Commonly Used In	Spring Boot REST controllers (`@RestController`)	Java Collections (`Iterator`, `Scanner`, `Optional#get()`)
HTTP Status Code	Usually mapped to `404 NOT FOU`	Not directly mapped to HTTP status, but can be converted to `404 NOT FOUND`

Now define the class named as **ResourceNotFoundException must be extending from exception** in the exception package

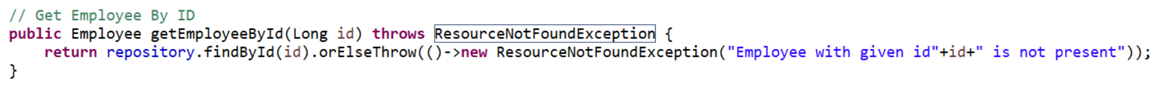


```
package com.wipro.exception;

public class ResourceNotFoundException extends Exception //handles this by Globalexceptionhandler
{

        public ResourceNotFoundException(String message) {
            super(message);
        }


}

```

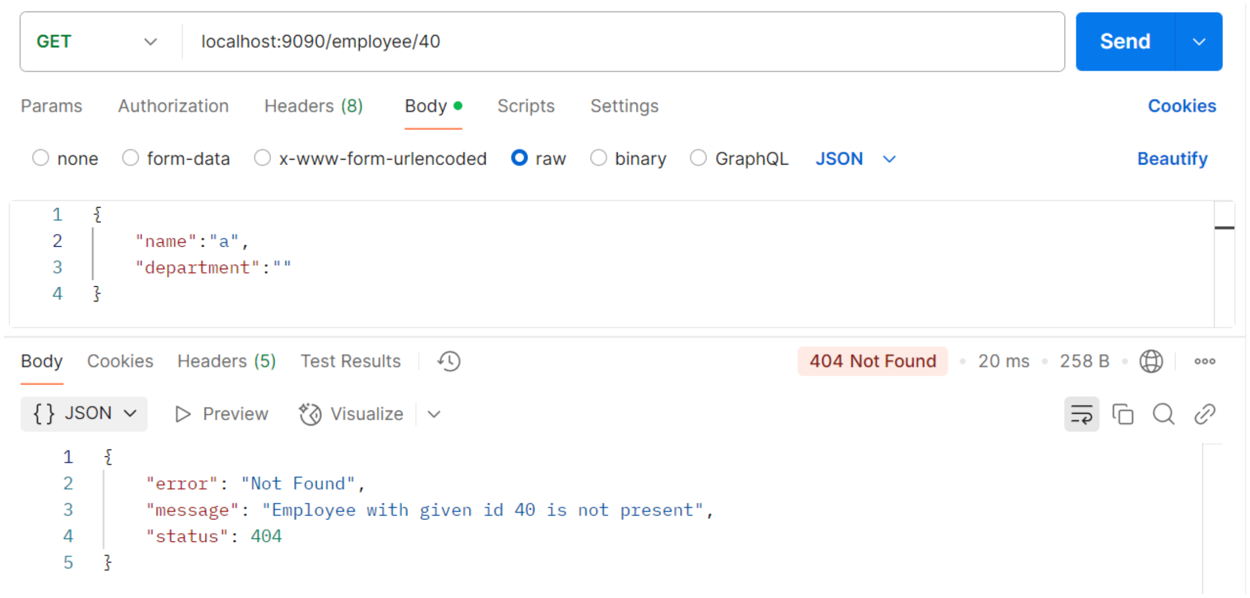
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<Map<String, Object>>
handleResourceNotFound(ResourceNotFoundException ex) {
    Map<String, Object> response = new HashMap<>();
    response.put("status", HttpStatus.NOT_FOUND.value());
    response.put("error", "Not Found");
    response.put("message", ex.getMessage());
    return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

Paste this code in the global exception class to handle the exception

Now come to the service class



```
// Get Employee By ID
public Employee getEmployeeById(Long id) throws ResourceNotFoundException {
    return repository.findById(id).orElseThrow(()->new ResourceNotFoundException("Employee with given id"+id+" is not present"));
}
```



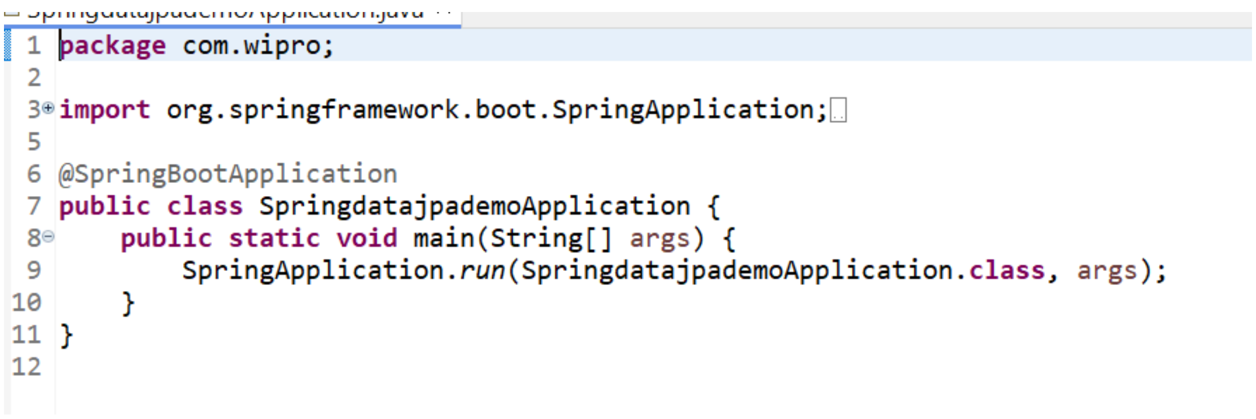
localhost:9090/employee/40

```
{
    "name":"a",
    "department":""
}
```

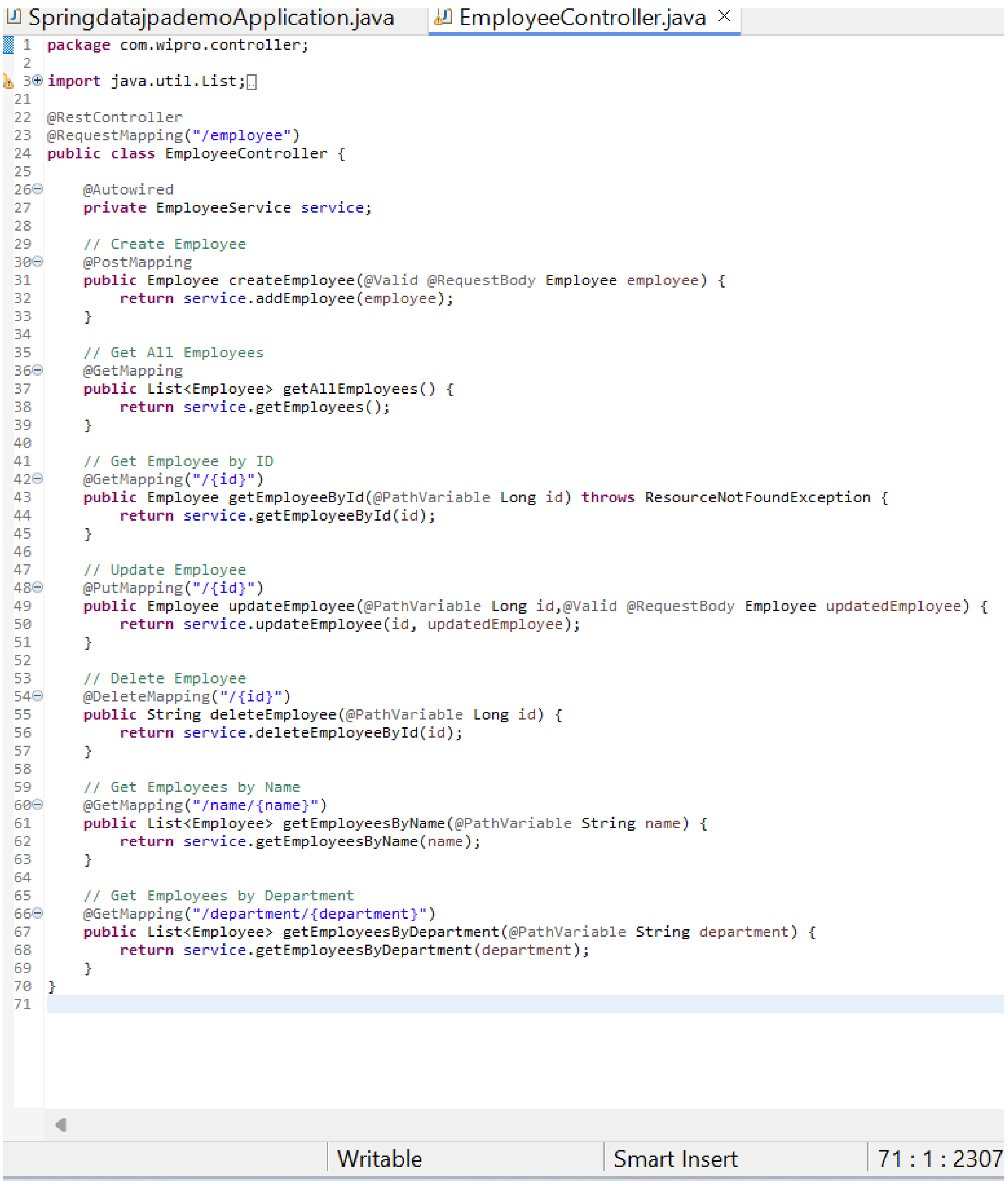
404 Not Found • 20 ms • 258 B

```
{
    "error": "Not Found",
    "message": "Employee with given id 40 is not present",
    "status": 404
}
```

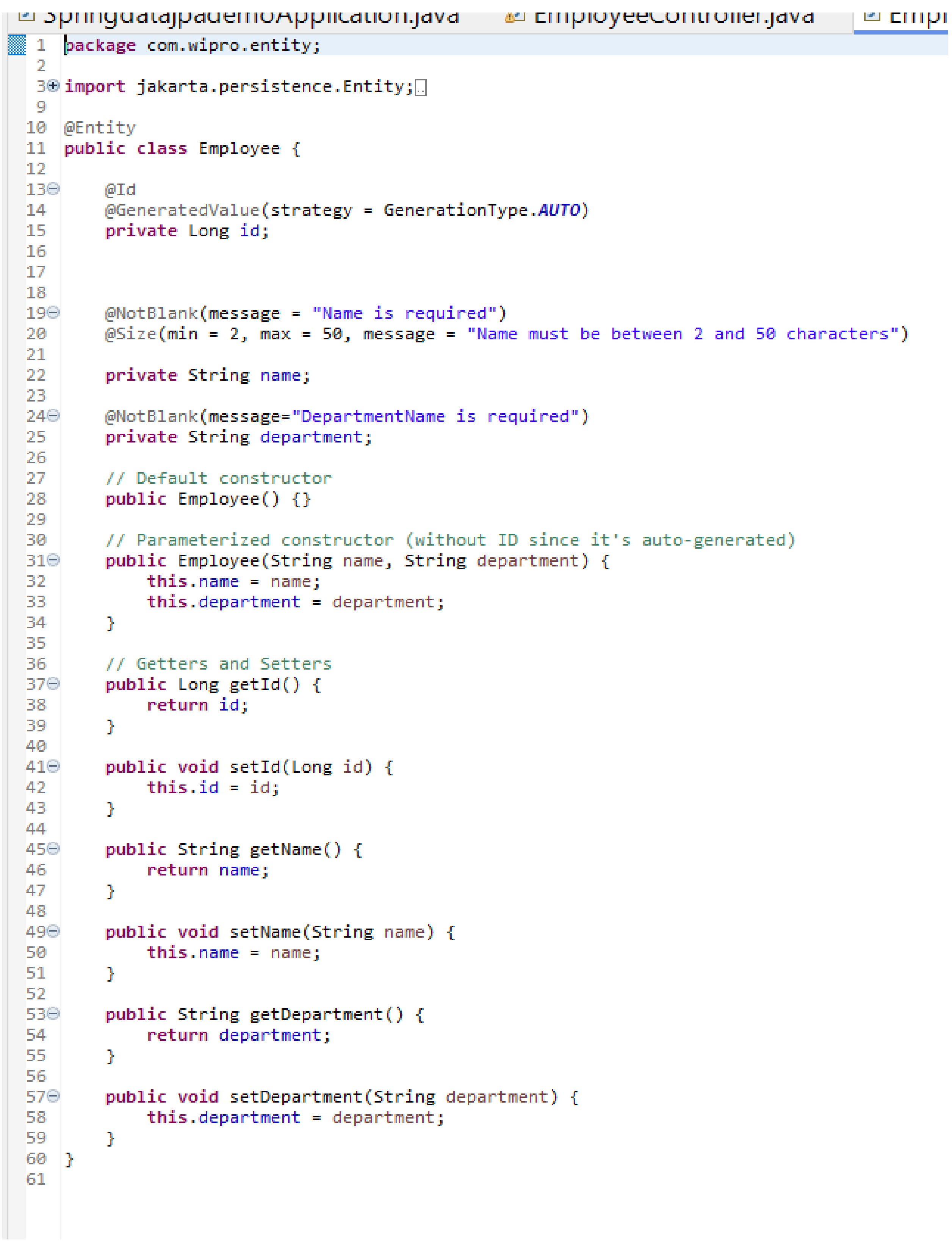
Now i will give u the overall codes



```java
package com.wipro;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringdatajpademoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringdatajpademoApplication.class, args);
    }
}

```

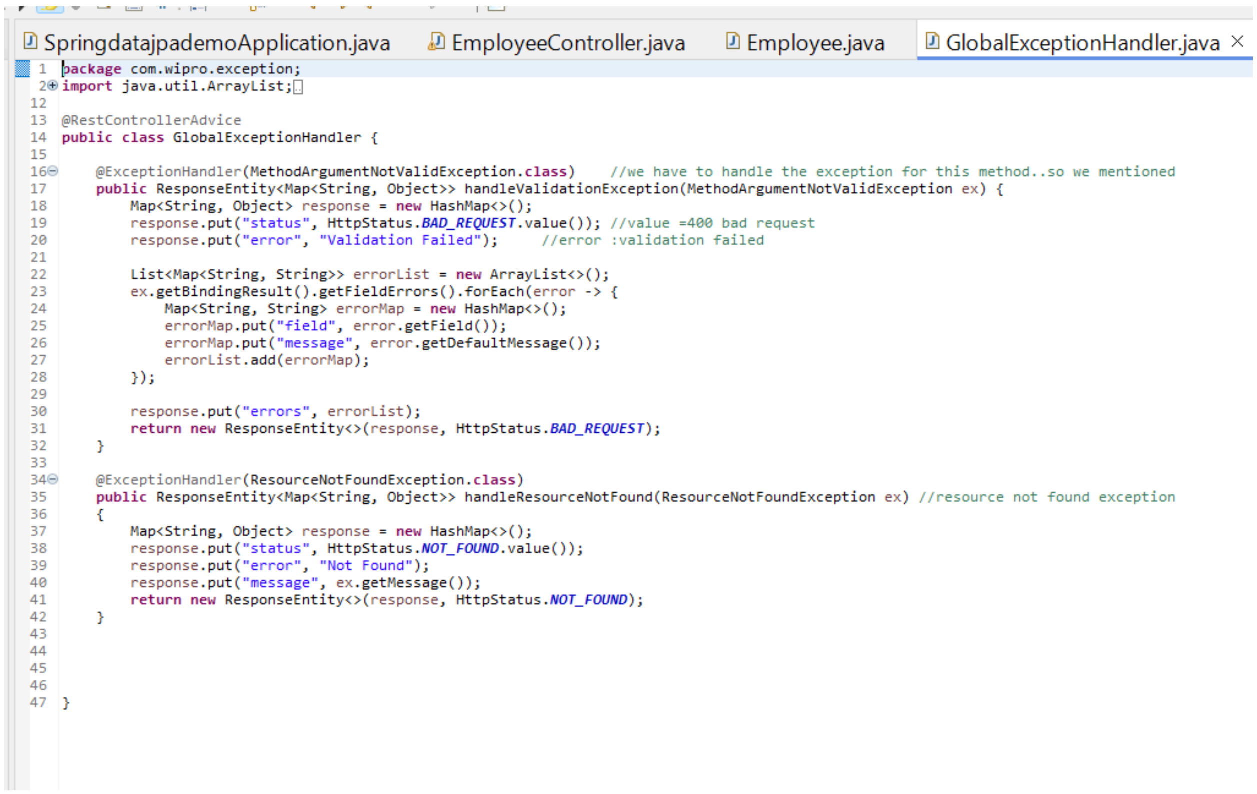


```java
package com.wipro.controller;

import java.util.List;

@RestController
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService service;

    // Create Employee
    @PostMapping
    public Employee createEmployee(@Valid @RequestBody Employee employee) {
        return service.addEmployee(employee);
    }

    // Get All Employees
    @GetMapping
    public List<Employee> getAllEmployees() {
        return service.getEmployees();
    }

    // Get Employee by ID
    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) throws ResourceNotFoundException {
        return service.getEmployeeById(id);
    }

    // Update Employee
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id,@Valid @RequestBody Employee updatedEmployee) {
        return service.updateEmployee(id, updatedEmployee);
    }

    // Delete Employee
    @DeleteMapping("/{id}")
    public String deleteEmployee(@PathVariable Long id) {
        return service.deleteEmployeeById(id);
    }

    // Get Employees by Name
    @GetMapping("/name/{name}")
    public List<Employee> getEmployeesByName(@PathVariable String name) {
        return service.getEmployeesByName(name);
    }

    // Get Employees by Department
    @GetMapping("/department/{department}")
    public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
        return service.getEmployeesByDepartment(department);
    }
}
```

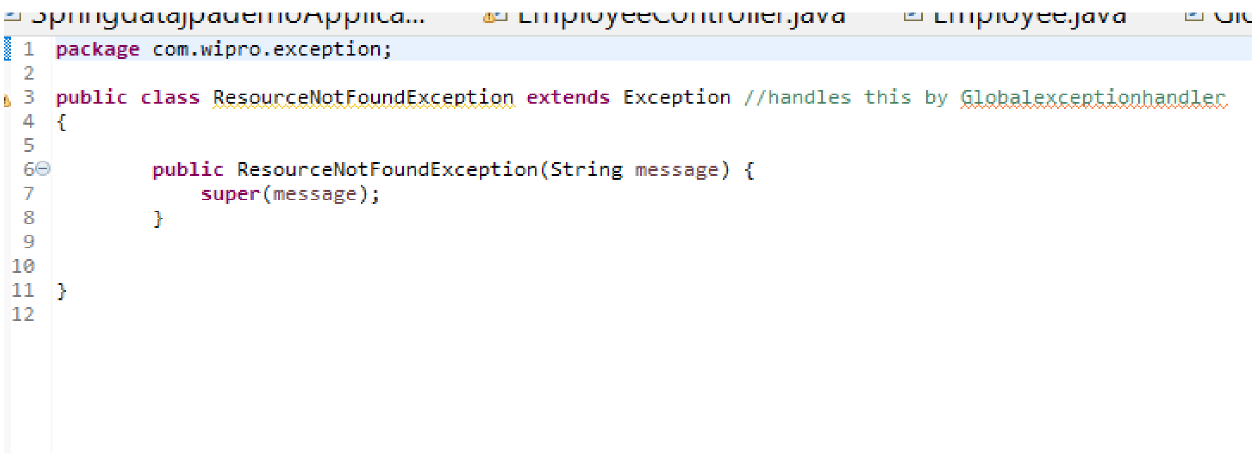


SpringdatajpademoApplication.java EmployeeController.java Empl

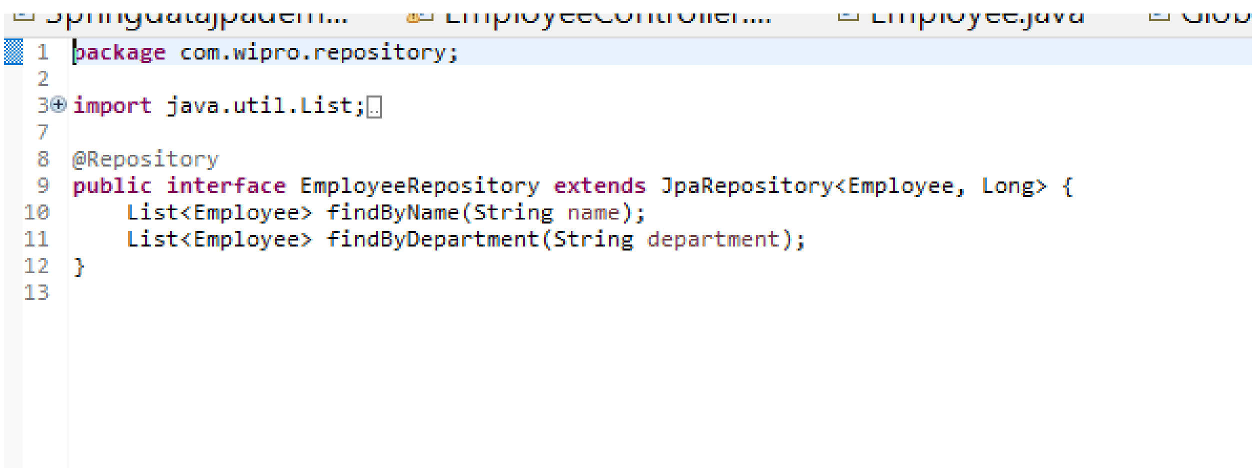
```java
package com.wipro.entity;

import jakarta.persistence.Entity;

@Entity
public class Employee {

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



    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

    private String name;

    @NotBlank(message="DepartmentName is required")
    private String department;

    // Default constructor
    public Employee() {}

    // Parameterized constructor (without ID since it's auto-generated)
    public Employee(String name, String department) {
        this.name = name;
        this.department = department;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }
}
```


SpringdatajpademoApplication.java
EmployeeController.java
Employee.java
GlobalExceptionHandler.java

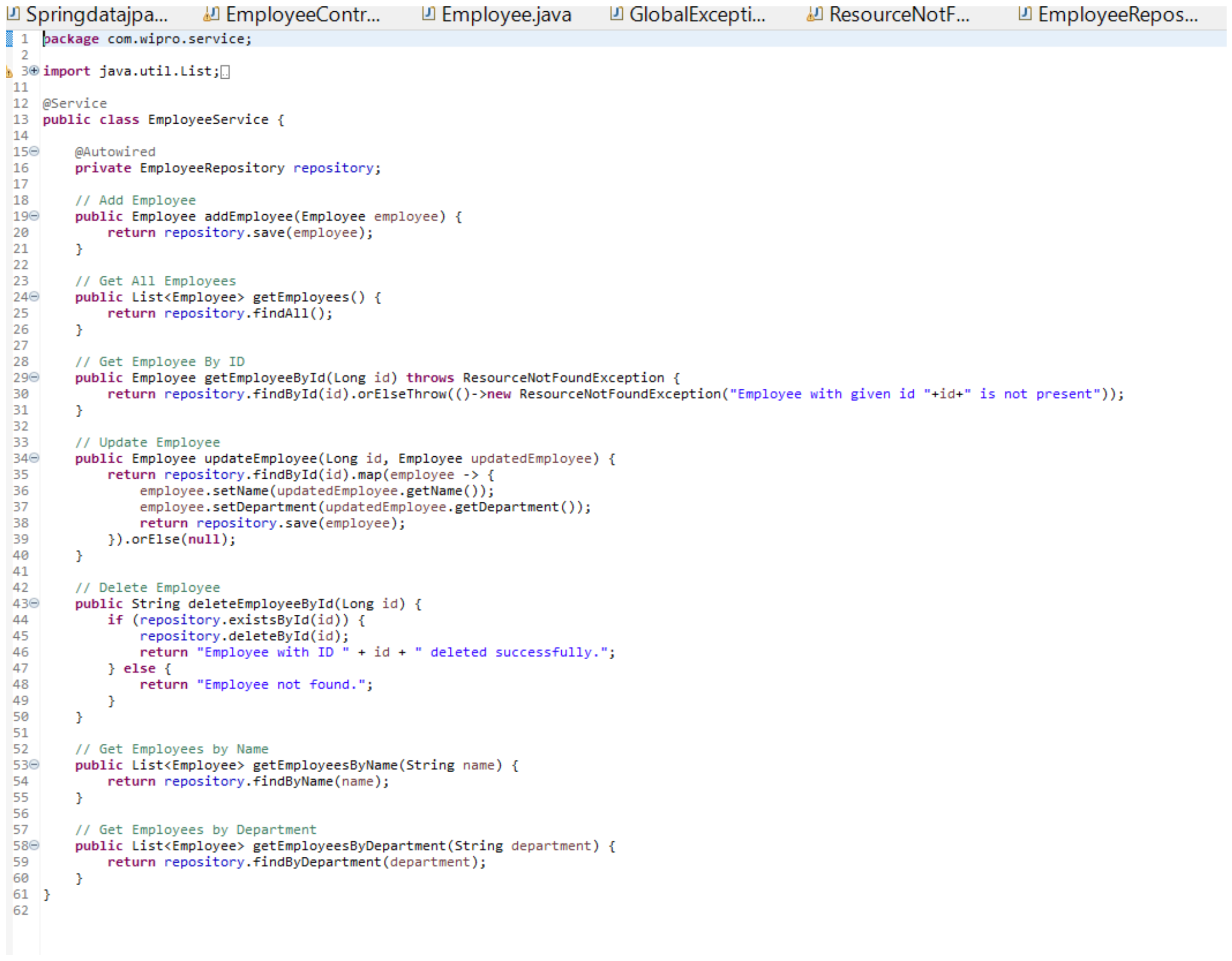
```java
package com.wipro.exception;
import java.util.ArrayList;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)    //we have to handle the exception for this method..so we mentioned
    public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("status", HttpStatus.BAD_REQUEST.value()); //value =400 bad request
        response.put("error", "Validation Failed");     //error :validation failed

        List<Map<String, String>> errorList = new ArrayList<>();
        ex.getBindingResult().getFieldErrors().forEach(error -> {
            Map<String, String> errorMap = new HashMap<>();
            errorMap.put("field", error.getField());
            errorMap.put("message", error.getDefaultMessage());
            errorList.add(errorMap);
        });

        response.put("errors", errorList);
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Map<String, Object>> handleResourceNotFound(ResourceNotFoundException ex) //resource not found exception
    {
        Map<String, Object> response = new HashMap<>();
        response.put("status", HttpStatus.NOT_FOUND.value());
        response.put("error", "Not Found");
        response.put("message", ex.getMessage());
        return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
    }




}
```



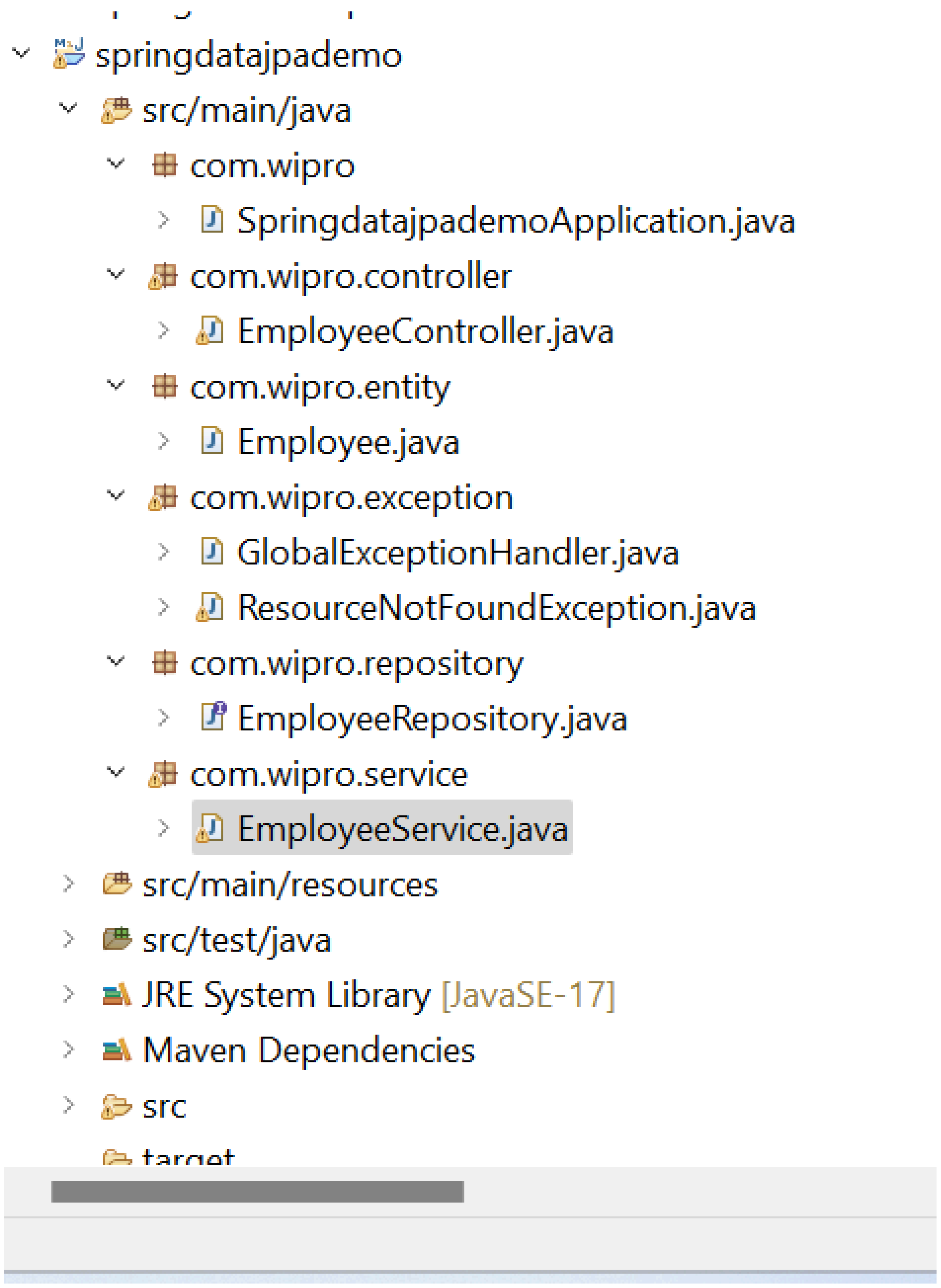
```java
package com.wipro.exception;

public class ResourceNotFoundException extends Exception //handles this by Globalexceptionhandler
{

        public ResourceNotFoundException(String message) {
            super(message);
        }


}

```



```java
package com.wipro.repository;

import java.util.List;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByName(String name);
    List<Employee> findByDepartment(String department);
}

```



```java
package com.wipro.service;

import java.util.List;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    // Add Employee
    public Employee addEmployee(Employee employee) {
        return repository.save(employee);
    }

    // Get All Employees
    public List<Employee> getEmployees() {
        return repository.findAll();
    }

    // Get Employee By ID
    public Employee getEmployeeById(Long id) throws ResourceNotFoundException {
        return repository.findById(id).orElseThrow(()->new ResourceNotFoundException("Employee with given id "+id+" is not present"));
    }

    // Update Employee
    public Employee updateEmployee(Long id, Employee updatedEmployee) {
        return repository.findById(id).map(employee -> {
            employee.setName(updatedEmployee.getName());
            employee.setDepartment(updatedEmployee.getDepartment());
            return repository.save(employee);
        }).orElse(null);
    }

    // Delete Employee
    public String deleteEmployeeById(Long id) {
        if (repository.existsById(id)) {
            repository.deleteById(id);
            return "Employee with ID " + id + " deleted successfully.";
        } else {
            return "Employee not found.";
        }
    }

    // Get Employees by Name
    public List<Employee> getEmployeesByName(String name) {
        return repository.findByName(name);
    }

    // Get Employees by Department
    public List<Employee> getEmployeesByDepartment(String department) {
        return repository.findByDepartment(department);
    }
}
```

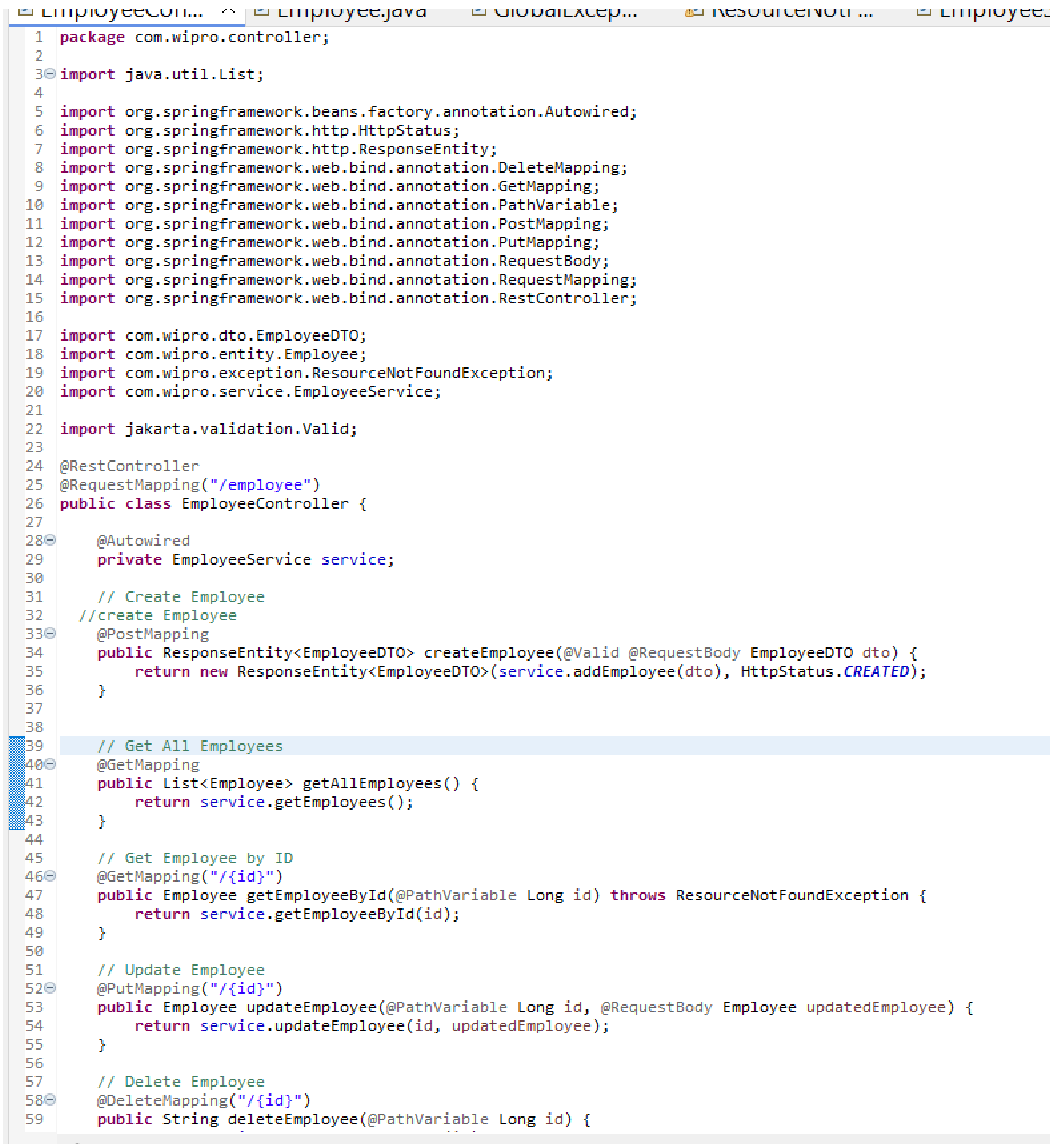


```
springdatajpademo
  src/main/java
    com.wipro
      SpringdatajpademoApplication.java
    com.wipro.controller
      EmployeeController.java
    com.wipro.entity
      Employee.java
    com.wipro.exception
      GlobalExceptionHandler.java
      ResourceNotFoundException.java
    com.wipro.repository
      EmployeeRepository.java
    com.wipro.service
      EmployeeService.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
```

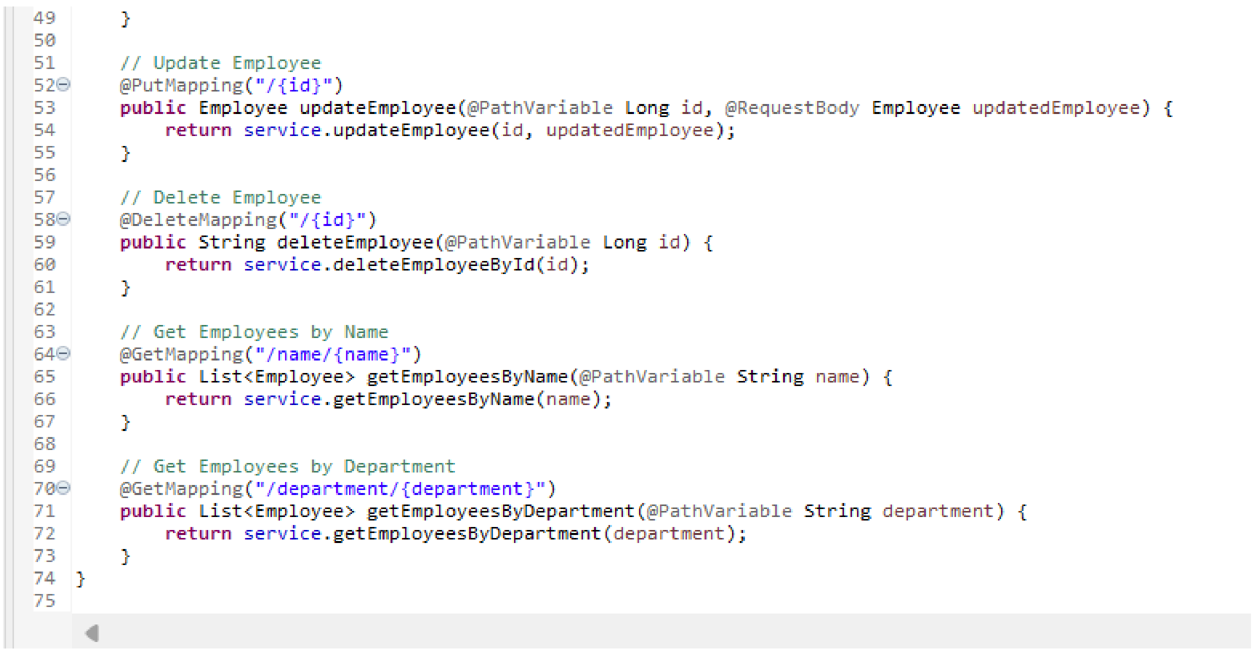
A **Data Transfer Object (DTO)** is a simple Java class used to **transfer data** between different layers of an application, typically between the **service layer and the controller layer** in a Spring Boot application.

Why Use a DTO?

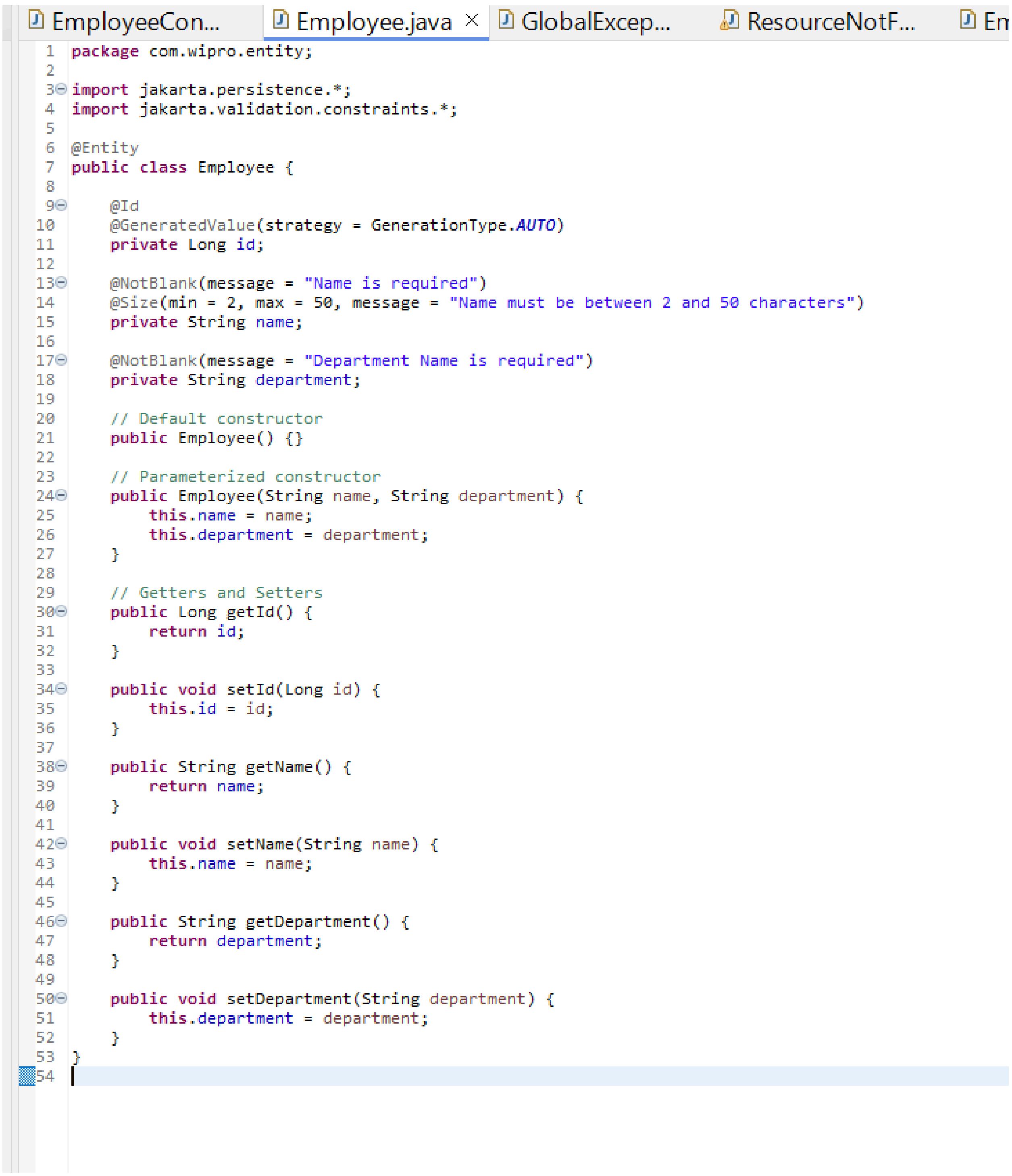
1. **Encapsulation**: Hides unnecessary fields from the client.
2. **Security**: Prevents exposing sensitive database fields.
3. **Performance**: Reduces unnecessary data transfer, improving efficiency.
4. **Validation**: Ensures data integrity before persisting it.
5. **Decoupling**: Separates the entity layer from the API layer.



```java
package com.wipro.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.wipro.dto.EmployeeDTO;
import com.wipro.entity.Employee;
import com.wipro.exception.ResourceNotFoundException;
import com.wipro.service.EmployeeService;

import jakarta.validation.Valid;

@RestController
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService service;

    // Create Employee
  //create Employee
    @PostMapping
    public ResponseEntity<EmployeeDTO> createEmployee(@Valid @RequestBody EmployeeDTO dto) {
        return new ResponseEntity<EmployeeDTO>(service.addEmployee(dto), HttpStatus.CREATED);
    }


    // Get All Employees
    @GetMapping
    public List<Employee> getAllEmployees() {
        return service.getEmployees();
    }

    // Get Employee by ID
    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) throws ResourceNotFoundException {
        return service.getEmployeeById(id);
    }

    // Update Employee
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
        return service.updateEmployee(id, updatedEmployee);
    }

    // Delete Employee
    @DeleteMapping("/{id}")
    public String deleteEmployee(@PathVariable Long id) {
```

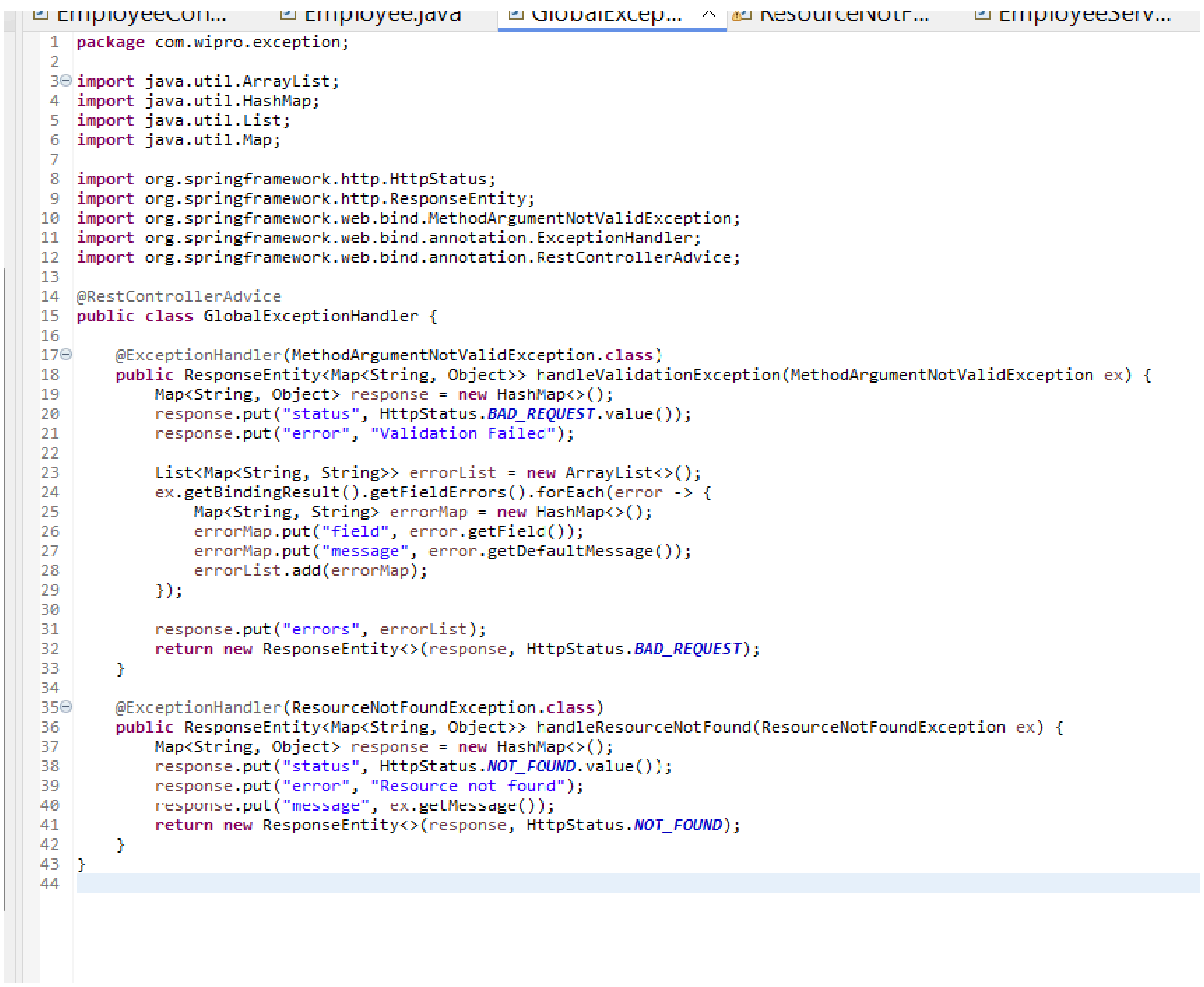


```java
    }

    // Update Employee
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
        return service.updateEmployee(id, updatedEmployee);
    }

    // Delete Employee
    @DeleteMapping("/{id}")
    public String deleteEmployee(@PathVariable Long id) {
        return service.deleteEmployeeById(id);
    }

    // Get Employees by Name
    @GetMapping("/name/{name}")
    public List<Employee> getEmployeesByName(@PathVariable String name) {
        return service.getEmployeesByName(name);
    }

    // Get Employees by Department
    @GetMapping("/department/{department}")
    public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
        return service.getEmployeesByDepartment(department);
    }
}
```

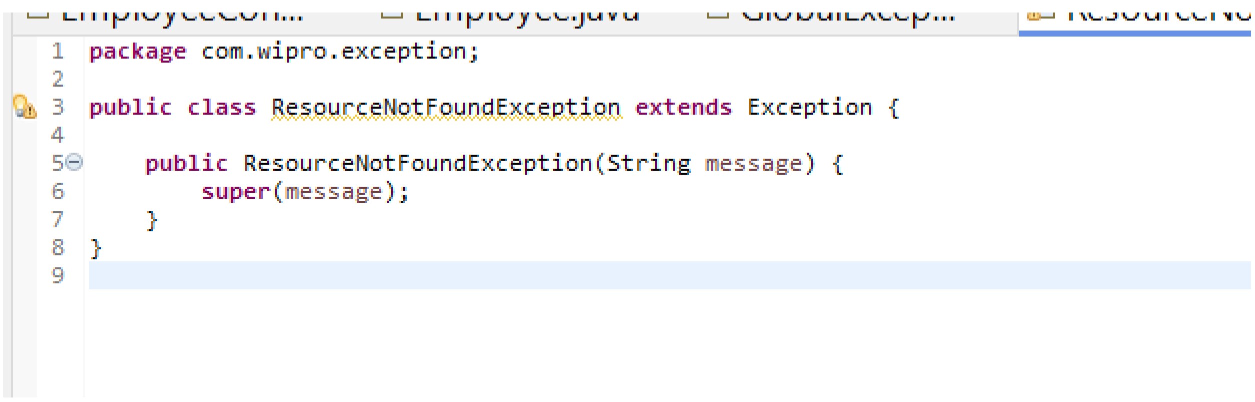


EmployeeCon... Employee.java GlobalExcep... ResourceNotF... En

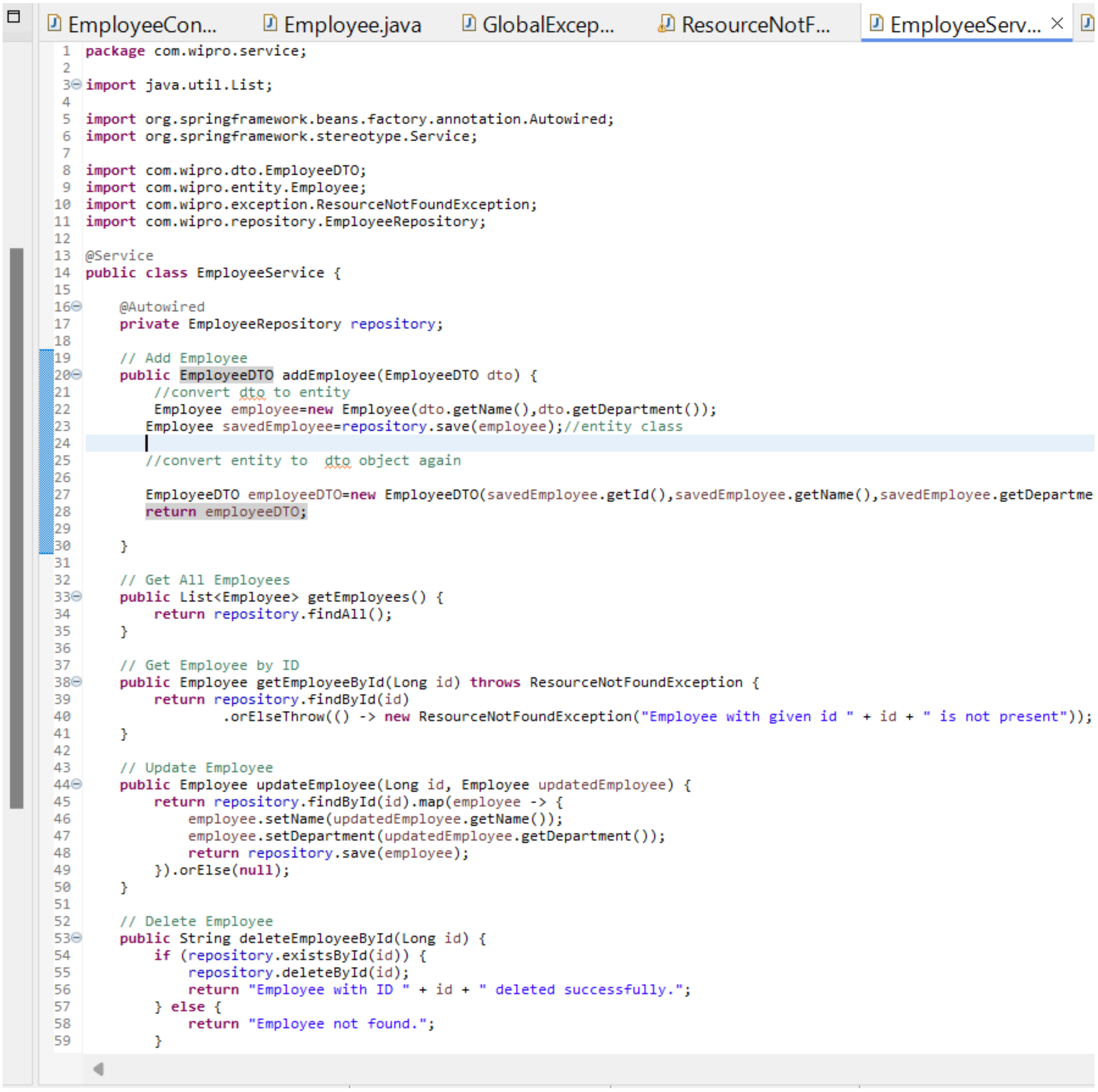
```java
package com.wipro.entity;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;

@Entity
public class Employee {

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

    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
    private String name;

    @NotBlank(message = "Department Name is required")
    private String department;

    // Default constructor
    public Employee() {}

    // Parameterized constructor
    public Employee(String name, String department) {
        this.name = name;
        this.department = department;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }
}

```



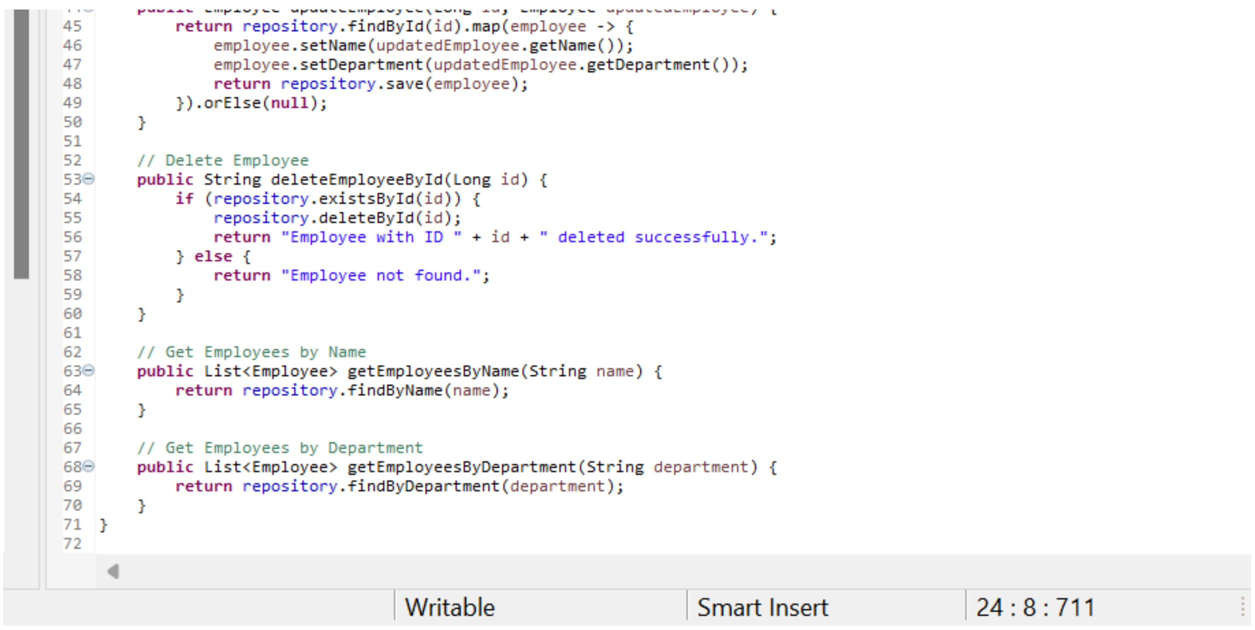
```java
package com.wipro.exception;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("status", HttpStatus.BAD_REQUEST.value());
        response.put("error", "Validation Failed");

        List<Map<String, String>> errorList = new ArrayList<>();
        ex.getBindingResult().getFieldErrors().forEach(error -> {
            Map<String, String> errorMap = new HashMap<>();
            errorMap.put("field", error.getField());
            errorMap.put("message", error.getDefaultMessage());
            errorList.add(errorMap);
        });

        response.put("errors", errorList);
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Map<String, Object>> handleResourceNotFound(ResourceNotFoundException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("status", HttpStatus.NOT_FOUND.value());
        response.put("error", "Resource not found");
        response.put("message", ex.getMessage());
        return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
    }
}
```



```java
package com.wipro.exception;

public class ResourceNotFoundException extends Exception {

    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

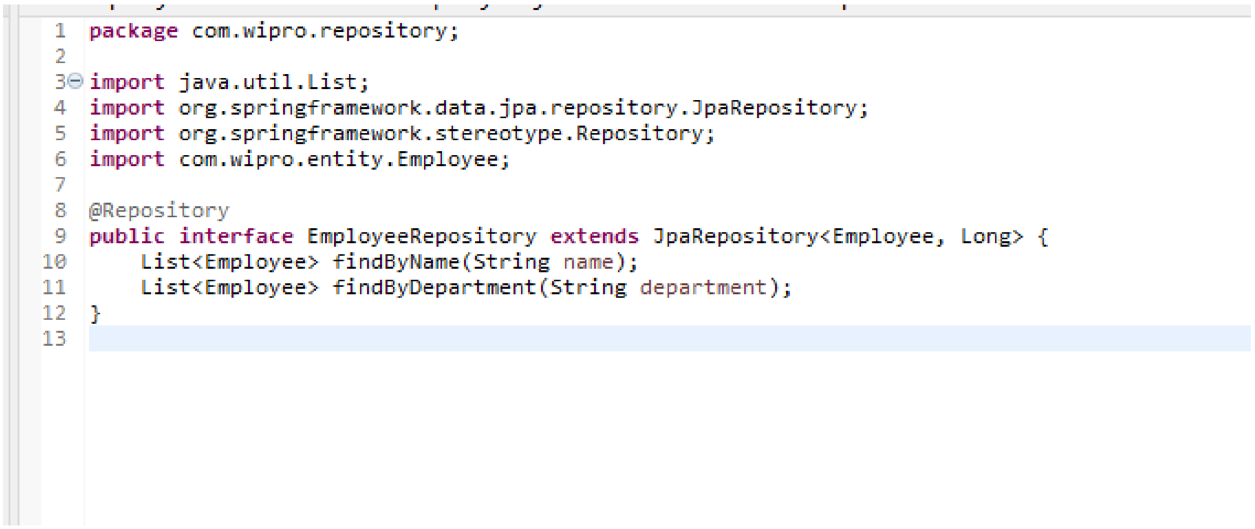


```java
package com.wipro.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.wipro.dto.EmployeeDTO;
import com.wipro.entity.Employee;
import com.wipro.exception.ResourceNotFoundException;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    // Add Employee
    public EmployeeDTO addEmployee(EmployeeDTO dto) {
        //convert dto to entity
        Employee employee=new Employee(dto.getName(),dto.getDepartment());
       Employee savedEmployee=repository.save(employee);//entity class

       //convert entity to  dto object again

       EmployeeDTO employeeDTO=new EmployeeDTO(savedEmployee.getId(),savedEmployee.getName(),savedEmployee.getDepartme
       return employeeDTO;

    }

    // Get All Employees
    public List<Employee> getEmployees() {
        return repository.findAll();
    }

    // Get Employee by ID
    public Employee getEmployeeById(Long id) throws ResourceNotFoundException {
        return repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Employee with given id " + id + " is not present"));
    }

    // Update Employee
    public Employee updateEmployee(Long id, Employee updatedEmployee) {
        return repository.findById(id).map(employee -> {
            employee.setName(updatedEmployee.getName());
            employee.setDepartment(updatedEmployee.getDepartment());
            return repository.save(employee);
        }).orElse(null);
    }

    // Delete Employee
    public String deleteEmployeeById(Long id) {
        if (repository.existsById(id)) {
            repository.deleteById(id);
            return "Employee with ID " + id + " deleted successfully.";
        } else {
            return "Employee not found.";
        }
```

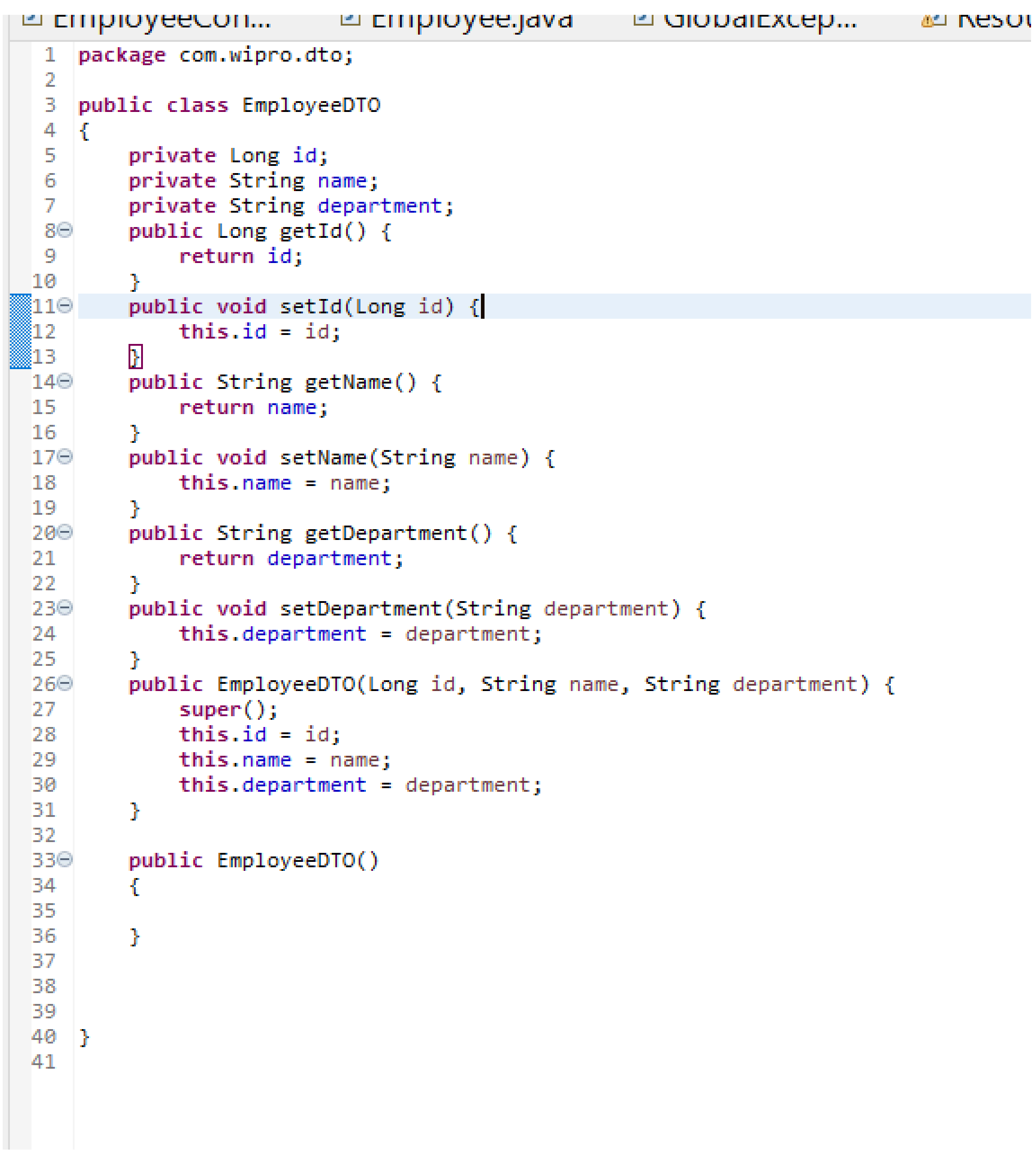


```
        return repository.findById(id).map(employee -> {
            employee.setName(updatedEmployee.getName());
            employee.setDepartment(updatedEmployee.getDepartment());
            return repository.save(employee);
        }).orElse(null);
    }

    // Delete Employee
    public String deleteEmployeeById(Long id) {
        if (repository.existsById(id)) {
            repository.deleteById(id);
            return "Employee with ID " + id + " deleted successfully.";
        } else {
            return "Employee not found.";
        }
    }

    // Get Employees by Name
    public List<Employee> getEmployeesByName(String name) {
        return repository.findByName(name);
    }

    // Get Employees by Department
    public List<Employee> getEmployeesByDepartment(String department) {
        return repository.findByDepartment(department);
    }
}
```

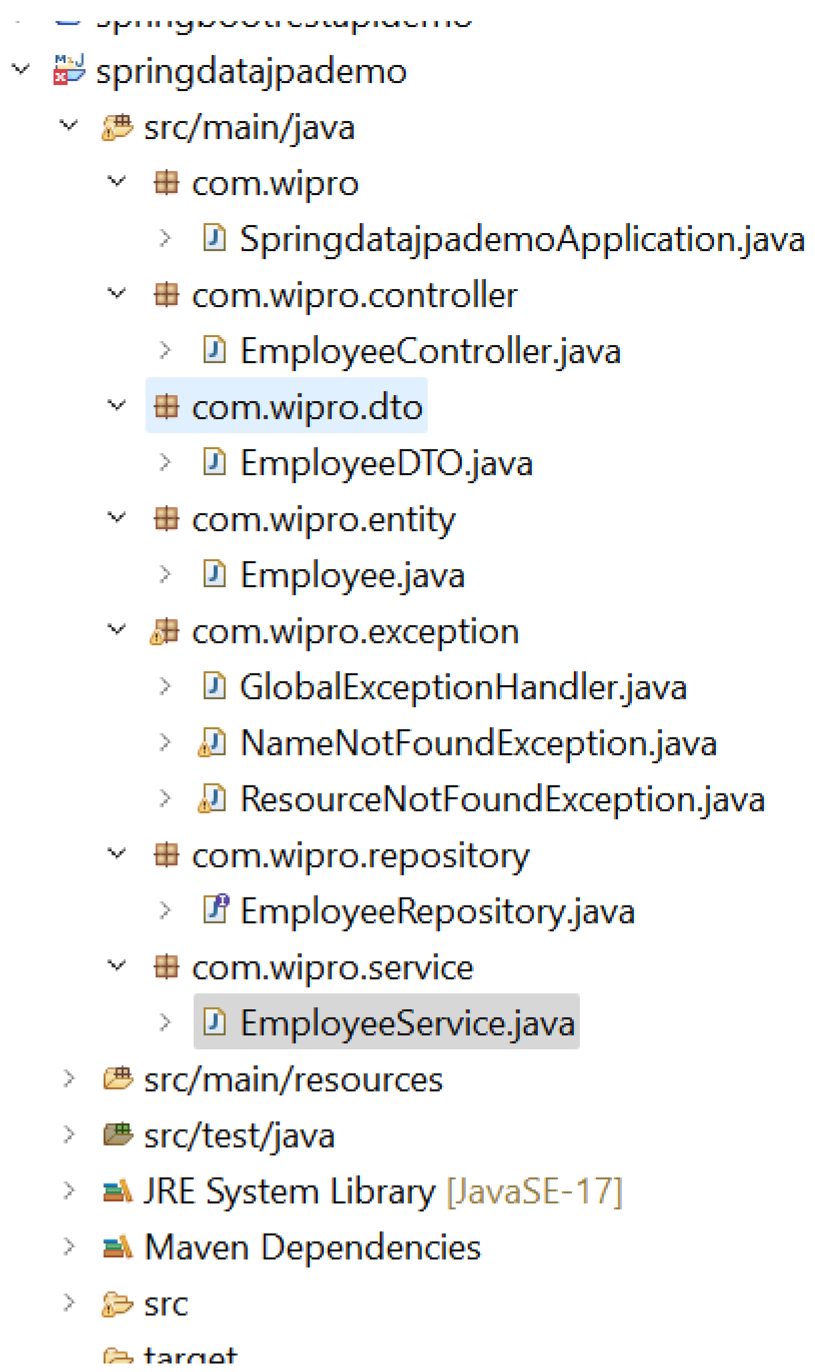
Writable Smart Insert 24 : 8 : 711



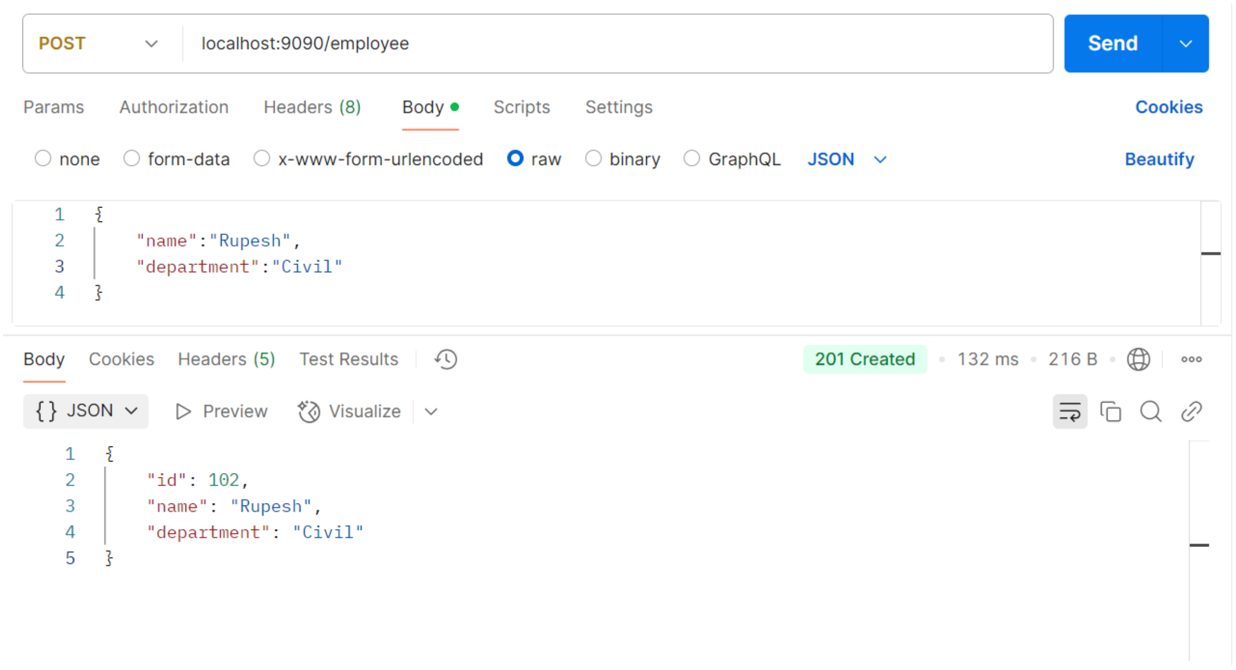
```
package com.wipro.repository;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.wipro.entity.Employee;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByName(String name);
    List<Employee> findByDepartment(String department);
}
```



```java
package com.wipro.dto;

public class EmployeeDTO
{
    private Long id;
    private String name;
    private String department;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
    public EmployeeDTO(Long id, String name, String department) {
        super();
        this.id = id;
        this.name = name;
        this.department = department;
    }

    public EmployeeDTO()
    {

    }



}
```



```
springdatajpademo
  src/main/java
    com.wipro
      SpringdatajpademoApplication.java
    com.wipro.controller
      EmployeeController.java
    com.wipro.dto
      EmployeeDTO.java
    com.wipro.entity
      Employee.java
    com.wipro.exception
      GlobalExceptionHandler.java
      NameNotFoundException.java
      ResourceNotFoundException.java
    com.wipro.repository
      EmployeeRepository.java
    com.wipro.service
      EmployeeService.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
```



POST localhost:9090/employee

```
{
    "name":"Rupesh",
    "department":"Civil"
}
```

201 Created • 132 ms • 216 B

```
{
    "id": 102,
    "name": "Rupesh",
    "department": "Civil"
}
```