


























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

```
1 spring.application.name=springdatajpa-pagination-sorting-demo
2
3 server.port=9090
4
5 spring.datasource.url=jdbc:mysql://localhost:3306/wipro
6 spring.datasource.username=root
7 spring.datasource.password=#Mahadev7
8 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
9
10 spring.jpa.hibernate.ddl-auto=update
11 spring.jpa.show-sql=true
12 spring.jpa.properties.hibernate.format_sql=true
13
14 |
```

```

1 package com.wipro.entity;
2
3 import jakarta.persistence.Entity;
4
5
6 @Entity
7 public class Employee {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private Long id;
12
13    private String name;
14    private String department;
15
16    // Default constructor
17    public Employee() {}
18
19    // Parameterized constructor (without ID since it's auto-generated)
20    public Employee(String name, String department) {
21        this.name = name;
22        this.department = department;
23    }
24
25    // Getters and Setters
26    public Long getId() {
27        return id;
28    }
29
30    public void setId(Long id) {
31        this.id = id;
32    }
33
34    public String getName() {
35        return name;
36    }
37
38    public void setName(String name) {
39        this.name = name;
40    }
41
42    public String getDepartment() {
43        return department;
44    }
45
46    public void setDepartment(String department) {
47        this.department = department;
48    }
49
50 }
51
52

```

```

1 package com.wipro.controller;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.data.domain.Page;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6 import org.springframework.web.bind.annotation.RestController;
7
8 import com.wipro.entity.Employee;
9 import com.wipro.service.EmployeeService;
10
11 @RestController
12 public class EmployeeController {
13
14     @Autowired
15     private EmployeeService employeeService;
16
17     @GetMapping("/pagination") // This mapping is correct
18     public Page<Employee> getEmployeesWithPagination(
19         @RequestParam(defaultValue = "0") int page,
20         @RequestParam(defaultValue = "5") int size) {
21
22         return employeeService.getEmployeeWithPagination(page, size);
23     }
24 }
25

```

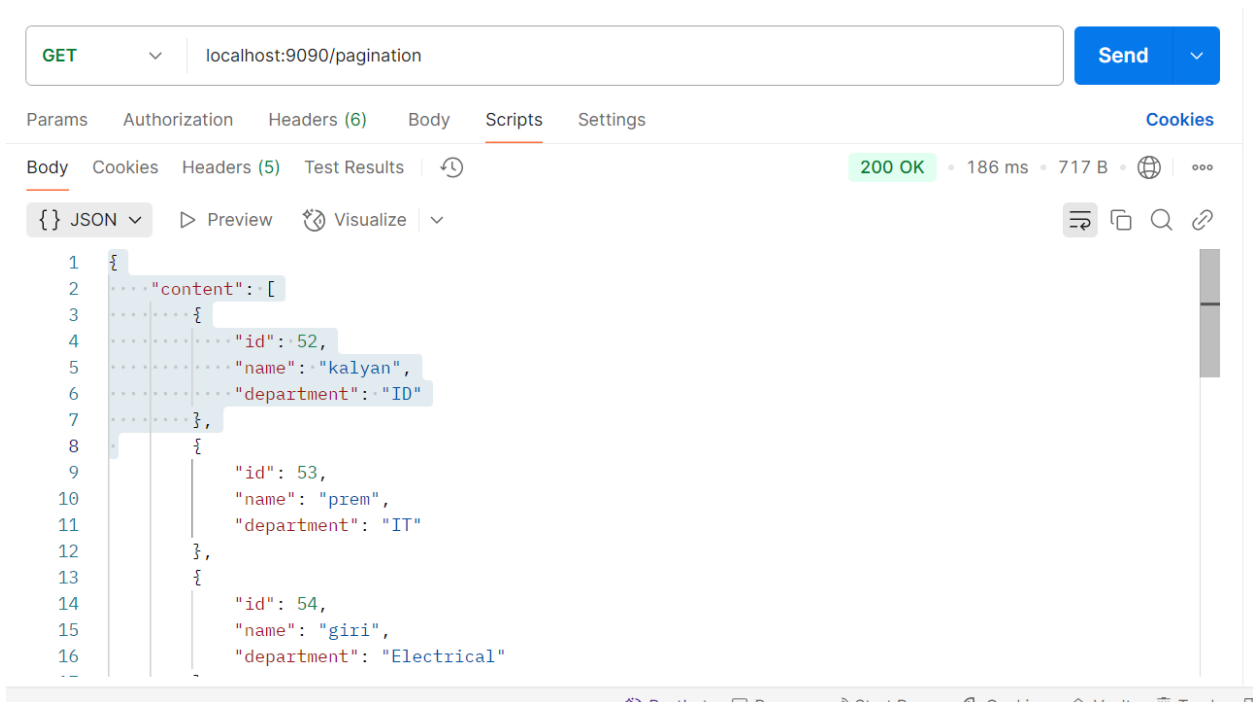
```

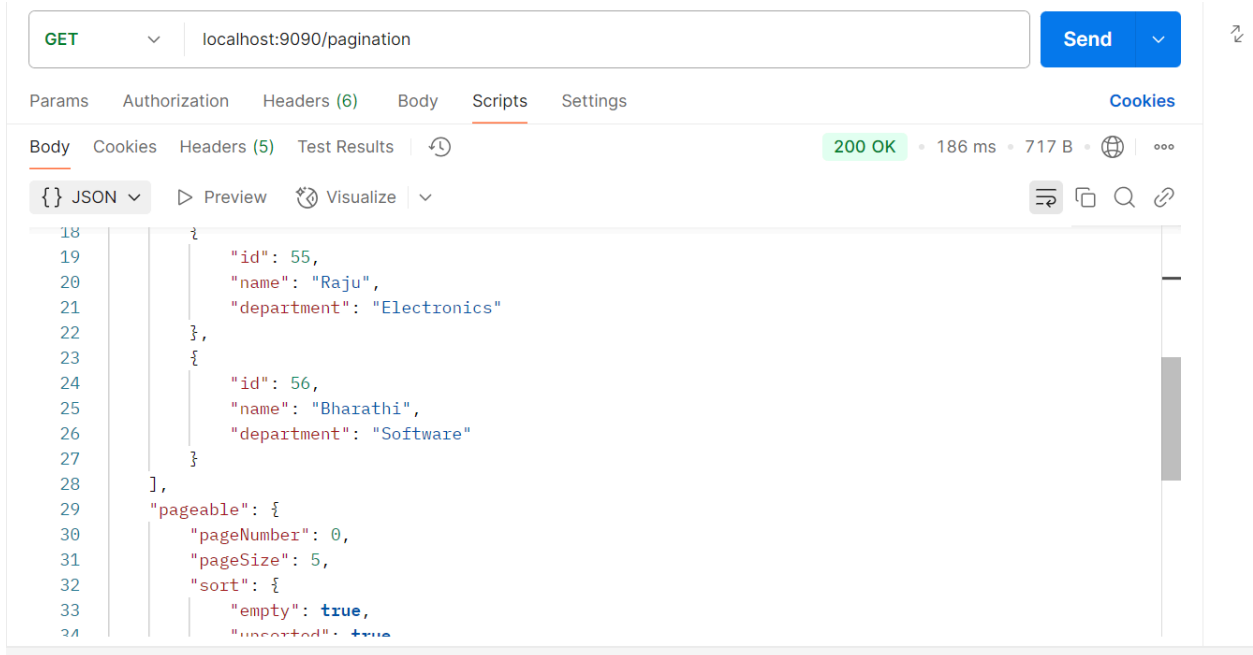
1 package com.wipro.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6
7
8
9
10
11
12 @Service
13 public class EmployeeService {
14
15     //get all employees with pagination
16     @Autowired
17     private EmployeeRepository employeeRepository;
18     public Page<Employee> getEmployeeWithPagination(int page, int size) {
19         Pageable pageable = PageRequest.of(page, size);
20         return employeeRepository.findAll(pageable);
21     }
22 }
23
24 }
25

```

```
SpringdatajpaPagi... EmployeeController.j... Employee.java EmployeeRepository.... X
1 package com.wipro.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
8
9 }
10
```

```
1 package com.wipro;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class SpringdatajpaPaginationSortingDemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringdatajpaPaginationSortingDemoApplication.class, args);
11     }
12 }
13
14
```





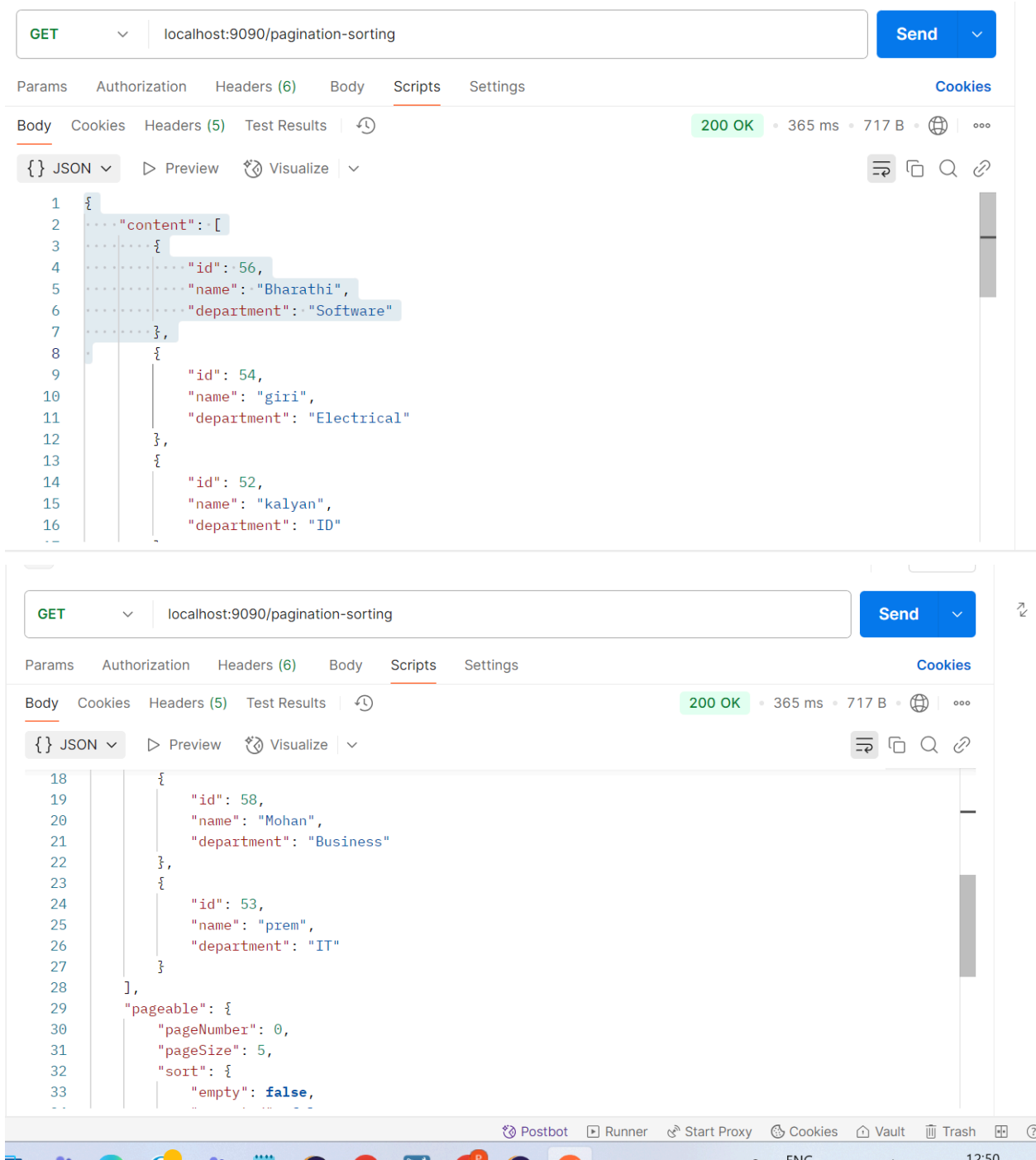
Sort the data by name

Now all the codes is same i changed only few

```
SpringdatajpaPagi... EmployeeController.j... Employee.java EmployeeRepository... EmployeeService.java x applica
1 package com.wipro.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
12
13 @Service
14 public class EmployeeService {
15
16     //get all employees with pagination
17     @Autowired
18     private EmployeeRepository employeeRepository;
19     public Page<Employee> getEmployeeWithPagination(int page, int size) {
20         Pageable pageable = PageRequest.of(page, size);
21         return employeeRepository.findAll(pageable);
22     }
23
24     public Page<Employee> getEmployeesWithSorting(int page, int size, String sortField) {
25
26         Pageable pageable = PageRequest.of(page, size, Sort.by(sortField).ascending());
27         return employeeRepository.findAll(pageable);
28     }
29
30 }
31
```

```
1 package com.wipro.controller;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.data.domain.Page;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6 import org.springframework.web.bind.annotation.RestController;
7
8 import com.wipro.entity.Employee;
9 import com.wipro.service.EmployeeService;
10
11 @RestController
12 public class EmployeeController {
13
14     @Autowired
15     private EmployeeService employeeService;
16
17     @GetMapping("/pagination") // This mapping is correct
18     public Page<Employee> getEmployeesWithPagination(
19         @RequestParam(defaultValue = "0") int page,
20         @RequestParam(defaultValue = "5") int size) {
21
22         return employeeService.getEmployeeWithPagination(page, size);
23     }
24
25     //sort the data by name
26     @GetMapping("pagination-sorting")
27     public Page<Employee> getEmployeesWithPaginationAndSorting(
28
29         @RequestParam(defaultValue = "0") int page,
30         @RequestParam(defaultValue = "5") int size,
31         @RequestParam(defaultValue = "name") String sortField){
32
33         return employeeService.getEmployeesWithSorting(page,size,sortField);
34
35     }
36 }
37
38 }
39
```

Writable	Smart Insert	17 · 18 · 548
----------	--------------	---------------



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.

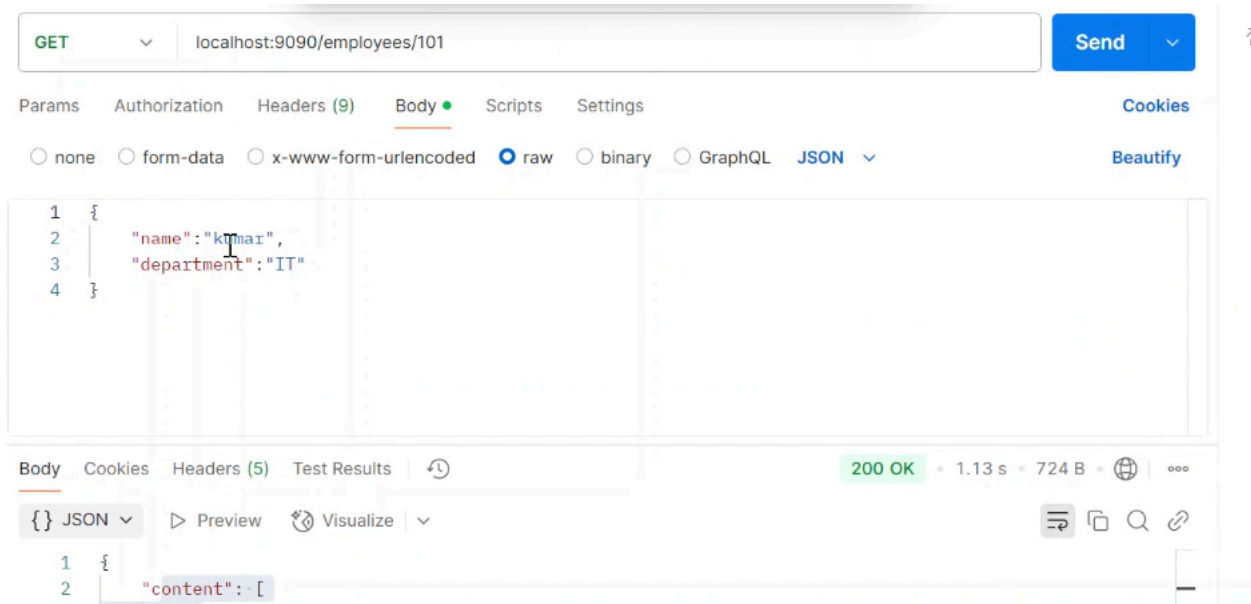
6. Handling 404 Errors (@ResponseStatus)

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



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

```

1  {
2    "name": "",
3    "department": "456"
4  }

```

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

```
@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

```
1 package com.wipro.controller;
2
3 import java.util.List;
4
20
21 @RestController
22 @RequestMapping("/employee")
23 public class EmployeeController {
24
25     @Autowired
26     private EmployeeService service;
27
28     // Create Employee
29     @PostMapping
30     public Employee createEmployee(@Valid @RequestBody Employee employee) {
31         return service.addEmployee(employee);
32     }
33
34     // Get All Employees
35     @GetMapping
36     public List<Employee> getAllEmployees() {
37         return service.getEmployees();
38     }
39
40     // Get Employee by ID
41     @GetMapping("/{id}")
42     public Optional<Employee> getEmployeeById(@PathVariable Long id) {
43         return service.getEmployeeById(id);
44     }
45
46     // Update Employee
47     @PutMapping("/{id}")
48     public Employee updateEmployee(@PathVariable Long id, @Valid @RequestBody Employee updatedEmployee) {
49         return service.updateEmployee(id, updatedEmployee);
50     }
51
52     // Delete Employee
53     @DeleteMapping("/{id}")
54 }
```

Now run the application

POST localhost:9090/employee

Send

Params Authorization Headers (8) Body Scripts Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON





















```
1 {
2   "name": "",
3   "department": ""
4 }
```

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

{ } JSON Preview Visualize

```
1 {
2   "timestamp": "2025-02-27T09:35:40.013+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "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 ×
1 package com.wipro;
2
3+ import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class SpringdatajpademoApplication {
8-     public static void main(String[] args) {
9         SpringApplication.run(SpringdatajpademoApplication.class, args);
10    }
11 }
12
```

SpringdatajpademoApplication.java × EmployeeController.java ×

```
1 package com.wipro.controller;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/employee")
7 public class EmployeeController {
8
9     @Autowired
10     private EmployeeService service;
11
12     // Create Employee
13     @PostMapping
14     public Employee createEmployee(@Valid @RequestBody Employee employee) {
15         return service.addEmployee(employee);
16     }
17
18     // Get All Employees
19     @GetMapping
20     public List<Employee> getAllEmployees() {
21         return service.getEmployees();
22     }
23
24     // Get Employee by ID
25     @GetMapping("/{id}")
26     public Optional<Employee> getEmployeeById(@PathVariable Long id) {
27         return service.getEmployeeById(id);
28     }
29
30     // Update Employee
31     @PutMapping("/{id}")
32     public Employee updateEmployee(@PathVariable Long id,@Valid @RequestBody Employee updatedEmployee) {
33         return service.updateEmployee(id, updatedEmployee);
34     }
35
36     // Delete Employee
37     @DeleteMapping("/{id}")
38     public String deleteEmployee(@PathVariable Long id) {
39         return service.deleteEmployeeById(id);
40     }
41
42     // Get Employees by Name
43     @GetMapping("/name/{name}")
44     public List<Employee> getEmployeesByName(@PathVariable String name) {
45         return service.getEmployeesByName(name);
46     }
47
48     // Get Employees by Department
49     @GetMapping("/department/{department}")
50     public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
51         return service.getEmployeesByDepartment(department);
52     }
53 }
54
55 }
```

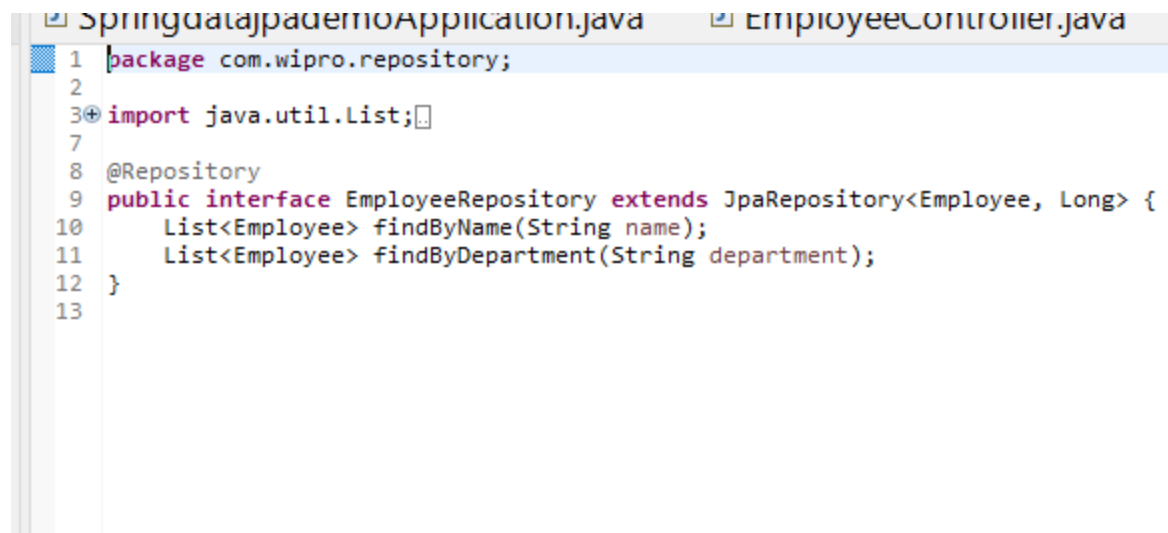
```
1 package com.wipro.entity;
2
3 import jakarta.persistence.Entity;
4
5
6
7
8
9
10 @Entity
11 public class Employee {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO)
15     private Long id;
16
17
18
19     @NotBlank(message = "Name is required")
20     @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
21
22     private String name;
23
24     @NotBlank(message="DepartmentName is required")
25     private String department;
26
27     // Default constructor
28     public Employee() {}
29
30     // Parameterized constructor (without ID since it's auto-generated)
31     public Employee(String name, String department) {
32         this.name = name;
33         this.department = department;
34     }
35
36     // Getters and Setters
37     public Long getId() {
38         return id;
39     }
40
41     public void setId(Long id) {
42         this.id = id;
43     }
44
45     public String getName() {
46         return name;
47     }
48
49     public void setName(String name) {
50         this.name = name;
51     }
52
53     public String getDepartment() {
54         return department;
55     }
56
57     public void setDepartment(String department) {
58         this.department = department;
59     }
60 }
61
```



```

1 package com.wipro.service;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 @Service
12 public class EmployeeService {
13
14     @Autowired
15     private EmployeeRepository repository;
16
17     // Add Employee
18     public Employee addEmployee(Employee employee) {
19         return repository.save(employee);
20     }
21
22     // Get All Employees
23     public List<Employee> getEmployees() {
24         return repository.findAll();
25     }
26
27     // Get Employee By ID
28     public Optional<Employee> getEmployeeById(Long id) {
29         return repository.findById(id);
30     }
31
32     // Update Employee
33     public Employee updateEmployee(Long id, Employee updatedEmployee) {
34         return repository.findById(id).map(employee -> {
35             employee.setName(updatedEmployee.getName());
36             employee.setDepartment(updatedEmployee.getDepartment());
37             return repository.save(employee);
38         }).orElse(null);
39     }
40
41     // Delete Employee
42     public String deleteEmployeeById(Long id) {
43         if (repository.existsById(id)) {
44             repository.deleteById(id);
45             return "Employee with ID " + id + " deleted successfully.";
46         } else {
47             return "Employee not found.";
48         }
49     }
50
51     // Get Employees by Name
52     public List<Employee> getEmployeesByName(String name) {
53         return repository.findByName(name);
54     }
55
56     // Get Employees by Department
57     public List<Employee> getEmployeesByDepartment(String department) {
58         return repository.findByDepartment(department);
59     }
60 }
61

```



```

1 package com.wipro.repository;
2
3 import java.util.List;
4
5
6
7
8 @Repository
9 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
10     List<Employee> findByName(String name);
11     List<Employee> findByDepartment(String department);
12 }
13

```

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

Now we need to handle that method

We don't 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

```

1 package com.wipro.exception;
2 import java.util.ArrayList;
3 import java.util.HashMap;
4 import java.util.List;
5 import java.util.Map;
6
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.MethodArgumentNotValidException;
10 import org.springframework.web.bind.annotation.ExceptionHandler;
11 import org.springframework.web.bind.annotation.RestControllerAdvice;
12
13 @RestControllerAdvice
14 public class GlobalExceptionHandler {
15
16     @ExceptionHandler(MethodArgumentNotValidException.class) //we have to handle the exception for this method..so we mentioned
17     public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
18         Map<String, Object> response = new HashMap<>();
19         response.put("status", HttpStatus.BAD_REQUEST.value()); //value =400 bad request
20         response.put("error", "Validation Failed"); //error :validation failed
21
22         List<Map<String, String>> errorList = new ArrayList<>();
23         ex.getBindingResult().getFieldErrors().forEach(error -> {
24             Map<String, String> errorMap = new HashMap<>();
25             errorMap.put("field", error.getField());
26             errorMap.put("message", error.getDefaultMessage());
27             errorList.add(errorMap);
28         });
29
30         response.put("errors", errorList);
31         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
32     }
33
34
35 }
36 }

```

now run the application

The screenshot shows a Postman interface with a POST request to `localhost:9090/employee`. The request body is a JSON object: `{ "name": "", "department": "" }`. The response is a `400 Bad Request` with a status of `400`, a time of `435 ms`, and a size of `375 B`. The response body is a JSON object: `{ "error": "Validation Failed", "errors": [{ "field": "name", "message": "Name is required" }, { "field": "department", "message": "Name is required" }] }`.

If we mention name:a

POST localhost:9090/employee Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

```

1 {
2   "name": "a",
3   "department": ""
4 }

```

Body Cookies Headers (4) Test Results 400 Bad Request 13 ms 329 B

{ } JSON Preview Visualize

```

6   "message": "DepartmentName is required"
7 },
8 {
9   "field": "name",
10  "message": "Name must be between 2 and 50 characters"
11 }
12 ],
13 "status": 400

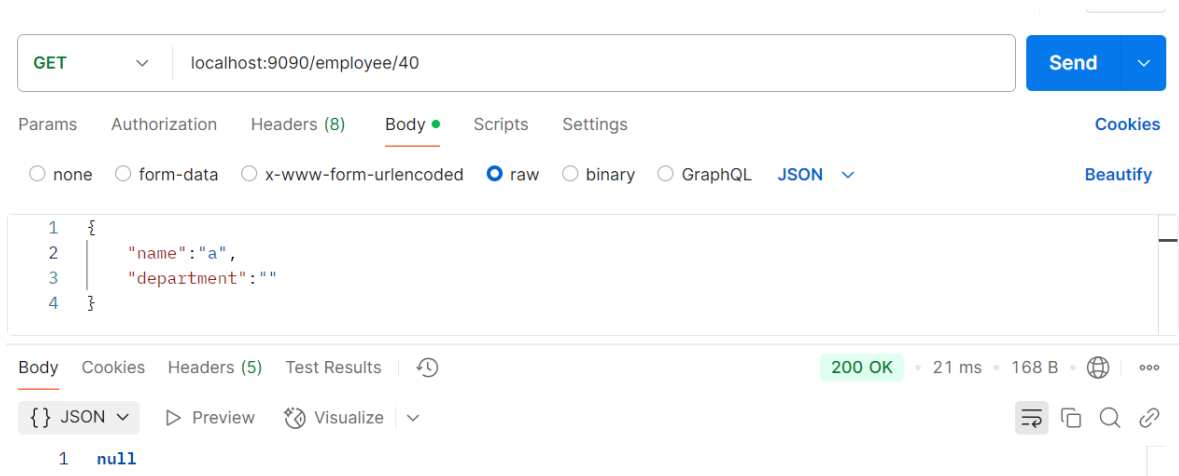
```

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

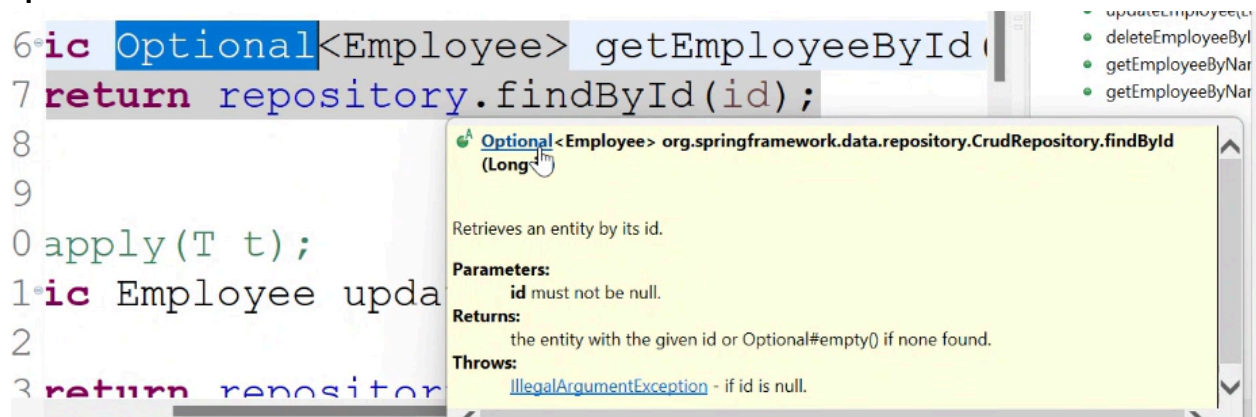
Now we r having this data in my database

Result Grid Filter Rows:			
	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



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

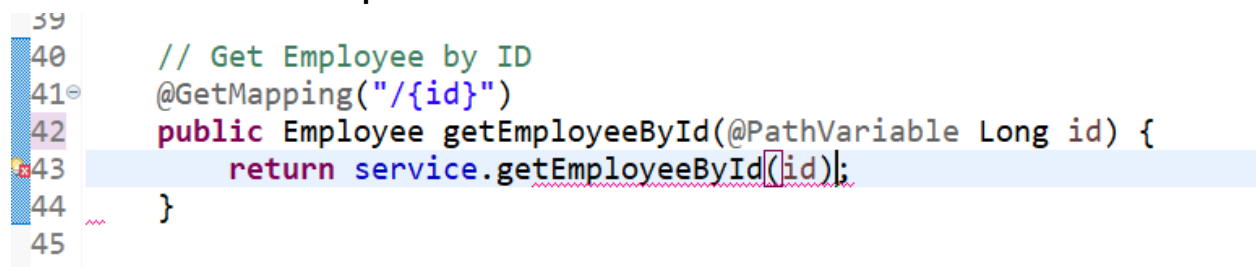


If the data is there then it returns ,if the data is not there then it returns the null

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



In the controller class

```
// 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

The screenshot shows a REST client interface. The request is a GET to `localhost:9090/employee/40`. The response status is **500 Internal Server Error**. The response body is a JSON object:

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

Feature	ResourceNotFoundException	NoSuchElementException
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 FOUND	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

```

1 package com.wipro.exception;
2
3 public class ResourceNotFoundException extends Exception //handles this by GlobalExceptionHandler
4 {
5
6     public ResourceNotFoundException(String message) {
7         super(message);
8     }
9
10
11 }
12

```

```

@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"));
}

```

GET localhost:9090/employee/40 Send

Params Authorization Headers (8) Body • Scripts Settings Cookies

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

```
1 {
2   "name": "a",
3   "department": ""
4 }
```

Body Cookies Headers (5) Test Results 404 Not Found • 20 ms • 258 B •

{ } JSON Preview Visualize

```
1 {
2   "error": "Not Found",
3   "message": "Employee with given id 40 is not present",
4   "status": 404
5 }
```

Now i will give u the overall codes

```
1 package com.wipro;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class SpringdatajpademoApplication {
8     public static void main(String[] args) {
9         SpringApplication.run(SpringdatajpademoApplication.class, args);
10    }
11 }
12
```



```
1 package com.wipro.controller;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/employee")
7 public class EmployeeController {
8
9     @Autowired
10     private EmployeeService service;
11
12     // Create Employee
13     @PostMapping
14     public Employee createEmployee(@Valid @RequestBody Employee employee) {
15         return service.addEmployee(employee);
16     }
17
18     // Get All Employees
19     @GetMapping
20     public List<Employee> getAllEmployees() {
21         return service.getEmployees();
22     }
23
24     // Get Employee by ID
25     @GetMapping("/{id}")
26     public Employee getEmployeeById(@PathVariable Long id) throws ResourceNotFoundException {
27         return service.getEmployeeById(id);
28     }
29
30     // Update Employee
31     @PutMapping("/{id}")
32     public Employee updateEmployee(@PathVariable Long id, @Valid @RequestBody Employee updatedEmployee) {
33         return service.updateEmployee(id, updatedEmployee);
34     }
35
36     // Delete Employee
37     @DeleteMapping("/{id}")
38     public String deleteEmployee(@PathVariable Long id) {
39         return service.deleteEmployeeById(id);
40     }
41
42     // Get Employees by Name
43     @GetMapping("/name/{name}")
44     public List<Employee> getEmployeesByName(@PathVariable String name) {
45         return service.getEmployeesByName(name);
46     }
47
48     // Get Employees by Department
49     @GetMapping("/department/{department}")
50     public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
51         return service.getEmployeesByDepartment(department);
52     }
53 }
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

```
1 package com.wipro.entity;
2
3 import jakarta.persistence.Entity;
4
5
6
7
8
9
10 @Entity
11 public class Employee {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO)
15     private Long id;
16
17
18
19     @NotBlank(message = "Name is required")
20     @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
21
22     private String name;
23
24     @NotBlank(message="DepartmentName is required")
25     private String department;
26
27     // Default constructor
28     public Employee() {}
29
30     // Parameterized constructor (without ID since it's auto-generated)
31     public Employee(String name, String department) {
32         this.name = name;
33         this.department = department;
34     }
35
36     // Getters and Setters
37     public Long getId() {
38         return id;
39     }
40
41     public void setId(Long id) {
42         this.id = id;
43     }
44
45     public String getName() {
46         return name;
47     }
48
49     public void setName(String name) {
50         this.name = name;
51     }
52
53     public String getDepartment() {
54         return department;
55     }
56
57     public void setDepartment(String department) {
58         this.department = department;
59     }
60 }
61
```

```

SpringdatajpademoApplication.java EmployeeController.java Employee.java GlobalExceptionHandler.java ×
1 package com.wipro.exception;
2 import java.util.ArrayList;
12
13 @RestControllerAdvice
14 public class GlobalExceptionHandler {
15
16     @ExceptionHandler(MethodArgumentNotValidException.class) //we have to handle the exception for this method..so we mentioned
17     public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
18         Map<String, Object> response = new HashMap<>();
19         response.put("status", HttpStatus.BAD_REQUEST.value()); //value =400 bad request
20         response.put("error", "Validation Failed"); //error :validation failed
21
22         List<Map<String, String>> errorList = new ArrayList<>();
23         ex.getBindingResult().getFieldErrors().forEach(error -> {
24             Map<String, String> errorMap = new HashMap<>();
25             errorMap.put("field", error.getField());
26             errorMap.put("message", error.getDefaultMessage());
27             errorList.add(errorMap);
28         });
29
30         response.put("errors", errorList);
31         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
32     }
33
34     @ExceptionHandler(ResourceNotFoundException.class)
35     public ResponseEntity<Map<String, Object>> handleResourceNotFound(ResourceNotFoundException ex) //resource not found exception
36     {
37         Map<String, Object> response = new HashMap<>();
38         response.put("status", HttpStatus.NOT_FOUND.value());
39         response.put("error", "Not Found");
40         response.put("message", ex.getMessage());
41         return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
42     }
43
44
45
46
47 }

```

```

SpringdatajpademoApplica... EmployeeController.java Employee.java GIC
1 package com.wipro.exception;
2
3 public class ResourceNotFoundException extends Exception //handles this by Globalexceptionhandler
4 {
5
6     public ResourceNotFoundException(String message) {
7         super(message);
8     }
9
10
11 }
12

```






















```

Springdatajpademo... EmployeeController... Employee.java GLOB
1 package com.wipro.repository;
2
3 import java.util.List;
7
8 @Repository
9 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
10     List<Employee> findByName(String name);
11     List<Employee> findByDepartment(String department);
12 }
13

```

Springdatajpa... EmployeeContr... Employee.java GlobalExcepti... ResourceNotF... EmployeeRepos...

```
1 package com.wipro.service;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12 @Service
13 public class EmployeeService {
14
15     @Autowired
16     private EmployeeRepository repository;
17
18     // Add Employee
19     public Employee addEmployee(Employee employee) {
20         return repository.save(employee);
21     }
22
23     // Get All Employees
24     public List<Employee> getEmployees() {
25         return repository.findAll();
26     }
27
28     // Get Employee By ID
29     public Employee getEmployeeById(Long id) throws ResourceNotFoundException {
30         return repository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee with given id "+id+" is not present"));
31     }
32
33     // Update Employee
34     public Employee updateEmployee(Long id, Employee updatedEmployee) {
35         return repository.findById(id).map(employee -> {
36             employee.setName(updatedEmployee.getName());
37             employee.setDepartment(updatedEmployee.getDepartment());
38             return repository.save(employee);
39         }).orElse(null);
40     }
41
42     // Delete Employee
43     public String deleteEmployeeById(Long id) {
44         if (repository.existsById(id)) {
45             repository.deleteById(id);
46             return "Employee with ID " + id + " deleted successfully.";
47         } else {
48             return "Employee not found.";
49         }
50     }
51
52     // Get Employees by Name
53     public List<Employee> getEmployeesByName(String name) {
54         return repository.findByName(name);
55     }
56
57     // Get Employees by Department
58     public List<Employee> getEmployeesByDepartment(String department) {
59         return repository.findByDepartment(department);
60     }
61 }
62
```

- ✓  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.

```
EmployeeCont... Employee.java GlobalExcep... ResourceNotFou... Employee.java
1 package com.wipro.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.DeleteMapping;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.PutMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RestController;
16
17 import com.wipro.dto.EmployeeDTO;
18 import com.wipro.entity.Employee;
19 import com.wipro.exception.ResourceNotFoundException;
20 import com.wipro.service.EmployeeService;
21
22 import jakarta.validation.Valid;
23
24 @RestController
25 @RequestMapping("/employee")
26 public class EmployeeController {
27
28     @Autowired
29     private EmployeeService service;
30
31     // Create Employee
32     //create Employee
33     @PostMapping
34     public ResponseEntity<EmployeeDTO> createEmployee(@Valid @RequestBody EmployeeDTO dto) {
35         return new ResponseEntity<EmployeeDTO>(service.addEmployee(dto), HttpStatus.CREATED);
36     }
37
38
39     // Get All Employees
40     @GetMapping
41     public List<Employee> getAllEmployees() {
42         return service.getEmployees();
43     }
44
45     // Get Employee by ID
46     @GetMapping("/{id}")
47     public Employee getEmployeeById(@PathVariable Long id) throws ResourceNotFoundException {
48         return service.getEmployeeById(id);
49     }
50
51     // Update Employee
52     @PutMapping("/{id}")
53     public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
54         return service.updateEmployee(id, updatedEmployee);
55     }
56
57     // Delete Employee
58     @DeleteMapping("/{id}")
59     public String deleteEmployee(@PathVariable Long id) {
```

```
49     }
50
51     // Update Employee
52     @PutMapping("/{id}")
53     public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
54         return service.updateEmployee(id, updatedEmployee);
55     }
56
57     // Delete Employee
58     @DeleteMapping("/{id}")
59     public String deleteEmployee(@PathVariable Long id) {
60         return service.deleteEmployeeById(id);
61     }
62
63     // Get Employees by Name
64     @GetMapping("/name/{name}")
65     public List<Employee> getEmployeesByName(@PathVariable String name) {
66         return service.getEmployeesByName(name);
67     }
68
69     // Get Employees by Department
70     @GetMapping("/department/{department}")
71     public List<Employee> getEmployeesByDepartment(@PathVariable String department) {
72         return service.getEmployeesByDepartment(department);
73     }
74 }
75
```

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

```
1 package com.wipro.entity;
2
3 import jakarta.persistence.*;
4 import jakarta.validation.constraints.*;
5
6 @Entity
7 public class Employee {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private Long id;
12
13    @NotBlank(message = "Name is required")
14    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
15    private String name;
16
17    @NotBlank(message = "Department Name is required")
18    private String department;
19
20    // Default constructor
21    public Employee() {}
22
23    // Parameterized constructor
24    public Employee(String name, String department) {
25        this.name = name;
26        this.department = department;
27    }
28
29    // Getters and Setters
30    public Long getId() {
31        return id;
32    }
33
34    public void setId(Long id) {
35        this.id = id;
36    }
37
38    public String getName() {
39        return name;
40    }
41
42    public void setName(String name) {
43        this.name = name;
44    }
45
46    public String getDepartment() {
47        return department;
48    }
49
50    public void setDepartment(String department) {
51        this.department = department;
52    }
53 }
54
```



```

EmployeeCon... Employee.java GlobalExcep... ResourceNotF... Employeeserv...
1 package com.wipro.exception;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.List;
6 import java.util.Map;
7
8 import org.springframework.http.HttpStatus;
9 import org.springframework.http.ResponseEntity;
10 import org.springframework.web.bind.MethodArgumentNotValidException;
11 import org.springframework.web.bind.annotation.ExceptionHandler;
12 import org.springframework.web.bind.annotation.RestControllerAdvice;
13
14 @RestControllerAdvice
15 public class GlobalExceptionHandler {
16
17     @ExceptionHandler(MethodArgumentNotValidException.class)
18     public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
19         Map<String, Object> response = new HashMap<>();
20         response.put("status", HttpStatus.BAD_REQUEST.value());
21         response.put("error", "Validation Failed");
22
23         List<Map<String, String>> errorList = new ArrayList<>();
24         ex.getBindingResult().getFieldErrors().forEach(error -> {
25             Map<String, String> errorMap = new HashMap<>();
26             errorMap.put("field", error.getField());
27             errorMap.put("message", error.getDefaultMessage());
28             errorList.add(errorMap);
29         });
30
31         response.put("errors", errorList);
32         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
33     }
34
35     @ExceptionHandler(ResourceNotFoundException.class)
36     public ResponseEntity<Map<String, Object>> handleResourceNotFound(ResourceNotFoundException ex) {
37         Map<String, Object> response = new HashMap<>();
38         response.put("status", HttpStatus.NOT_FOUND.value());
39         response.put("error", "Resource not found");
40         response.put("message", ex.getMessage());
41         return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
42     }
43 }
44

```

```

EmployeeCon... Employee.java GlobalExcep... ResourceNotF...
1 package com.wipro.exception;
2
3 public class ResourceNotFoundException extends Exception {
4
5     public ResourceNotFoundException(String message) {
6         super(message);
7     }
8 }
9

```

```
EmployeeCon... Employee.java GlobalExcep... ResourceNotF... EmployeeServ... ×
1 package com.wipro.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.wipro.dto.EmployeeDTO;
9 import com.wipro.entity.Employee;
10 import com.wipro.exception.ResourceNotFoundException;
11 import com.wipro.repository.EmployeeRepository;
12
13 @Service
14 public class EmployeeService {
15
16     @Autowired
17     private EmployeeRepository repository;
18
19     // Add Employee
20     public EmployeeDTO addEmployee(EmployeeDTO dto) {
21         //convert dto to entity
22         Employee employee=new Employee(dto.getName(),dto.getDepartment());
23         Employee savedEmployee=repository.save(employee);//entity class
24
25         //convert entity to dto object again
26
27         EmployeeDTO employeeDTO=new EmployeeDTO(savedEmployee.getId(),savedEmployee.getName(),savedEmployee.getDepartme
28         return employeeDTO;
29     }
30
31     // Get All Employees
32     public List<Employee> getEmployees() {
33         return repository.findAll();
34     }
35
36     // Get Employee by ID
37     public Employee getEmployeeById(Long id) throws ResourceNotFoundException {
38         return repository.findById(id)
39             .orElseThrow(() -> new ResourceNotFoundException("Employee with given id " + id + " is not present"));
40     }
41
42     // Update Employee
43     public Employee updateEmployee(Long id, Employee updatedEmployee) {
44         return repository.findById(id).map(employee -> {
45             employee.setName(updatedEmployee.getName());
46             employee.setDepartment(updatedEmployee.getDepartment());
47             return repository.save(employee);
48         }).orElse(null);
49     }
50
51     // Delete Employee
52     public String deleteEmployeeById(Long id) {
53         if (repository.existsById(id)) {
54             repository.deleteById(id);
55             return "Employee with ID " + id + " deleted successfully.";
56         } else {
57             return "Employee not found.";
58         }
59     }
60 }
```

```

45     return repository.findById(id).map(employee -> {
46         employee.setName(updatedEmployee.getName());
47         employee.setDepartment(updatedEmployee.getDepartment());
48         return repository.save(employee);
49     }).orElse(null);
50 }
51
52 // Delete Employee
53 public String deleteEmployeeById(Long id) {
54     if (repository.existsById(id)) {
55         repository.deleteById(id);
56         return "Employee with ID " + id + " deleted successfully.";
57     } else {
58         return "Employee not found.";
59     }
60 }
61
62 // Get Employees by Name
63 public List<Employee> getEmployeesByName(String name) {
64     return repository.findByName(name);
65 }
66
67 // Get Employees by Department
68 public List<Employee> getEmployeesByDepartment(String department) {
69     return repository.findByDepartment(department);
70 }
71 }
72

```

Writable

Smart Insert

24:8:711


























```

1 package com.wipro.repository;
2
3 import java.util.List;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6 import com.wipro.entity.Employee;
7
8 @Repository
9 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
10     List<Employee> findByName(String name);
11     List<Employee> findByDepartment(String department);
12 }
13

```

EmployeeCon... Employee.java GlobalExcep... Reso

```
1 package com.wipro.dto;
2
3 public class EmployeeDTO
4 {
5     private Long id;
6     private String name;
7     private String department;
8     public Long getId() {
9         return id;
10    }
11    public void setId(Long id) {
12        this.id = id;
13    }
14    public String getName() {
15        return name;
16    }
17    public void setName(String name) {
18        this.name = name;
19    }
20    public String getDepartment() {
21        return department;
22    }
23    public void setDepartment(String department) {
24        this.department = department;
25    }
26    public EmployeeDTO(Long id, String name, String department) {
27        super();
28        this.id = id;
29        this.name = name;
30        this.department = department;
31    }
32
33    public EmployeeDTO()
34    {
35
36    }
37
38
39
40 }
41
```

-  springbootdemo
- ✓  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
 -  target

POST

localhost:9090/employee

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

Beautify

```
1  {
2    "name": "Rupesh",
3    "department": "Civil"
4  }
```

Body

Cookies

Headers (5)

Test Results

201 Created

132 ms

216 B

{ } JSON

Preview

Visualize

```
1  {
2    "id": 102,
3    "name": "Rupesh",
4    "department": "Civil"
5  }
```