26-02-2025

Spring Data JPA

Spring Data JPA is a part of the **Spring Framework** that helps developers work with **databases easily without writing a lot of SQL queries.** It is built on top of **JPA (Java Persistence API)**, which is a standard way to interact with relational databases in Java.

Why Use Spring Data JPA?

✅ **Less Code:** No need to write SQL queries manually
✅ **Automatic CRUD Operations:** Fetch, insert, update, and delete with simple method calls
✅ **Faster Development:** Focus on business logic instead of database queries
✅ **Supports Different Databases:** Works with MySQL, PostgreSQL, Oracle, etc.

Advantages of Using Spring Data JPA

1️⃣ Reduces Boilerplate Code (Less Code, More Work Done)

❌ Traditional JDBC requires writing SQL queries manually.
✅ With Spring Data JPA, we just define methods like `findById()` or `save()`, and Spring generates the SQL automatically.
✔️ **No need for writing long SQL queries** like `SELECT * FROM books WHERE id=1;`

```
List<Book> books = bookRepository.findAll(); // Fetch all books
```

✅ No need for ResultSet, Statement, or handling connections manually.

2️⃣ Built-in CRUD Operations

Spring Data JPA provides a `JpaRepository` interface with common operations:

- `save()` → Insert or update a record
- `findById()` → Get a record by ID
- `findAll()` → Get all records
- `deleteById()` → Delete a record

✔️ Saves **time and effort** by providing **ready-made** CRUD methods.

3️⃣ Automatic Query Generation (Method Naming Conventions)

We can create **custom queries** just by defining method names, and Spring generates SQL automatically!

```
List<Book> findByAuthor(String author);
```

Spring automatically creates

```
SELECT * FROM book WHERE author = 'John Doe';
```

4️⃣ Pagination & Sorting Made Easy

Spring Data JPA provides **built-in** pagination & sorting without writing SQL.

```
Page<Book> books = bookRepository.findAll(PageRequest.of(0, 5,
Sort.by("title")));
```

✔ Fetch the **first 5 books** sorted by **title**.

5️⃣ Easy Integration with Hibernate & Other Databases

Spring Data JPA uses **Hibernate** (default JPA provider) under the hood, but it supports **other JPA providers** too.
It works seamlessly with: ✅ MySQL
✅ PostgreSQL

✅ Oracle
✅ MongoDB (with Spring Data MongoDB)

✔️ No need to write native SQL queries!

6️⃣ Supports Transactions Automatically

✅ Spring Data JPA provides **automatic transaction management**.
✅ We can use `@Transactional` to ensure database consistency.

Example:

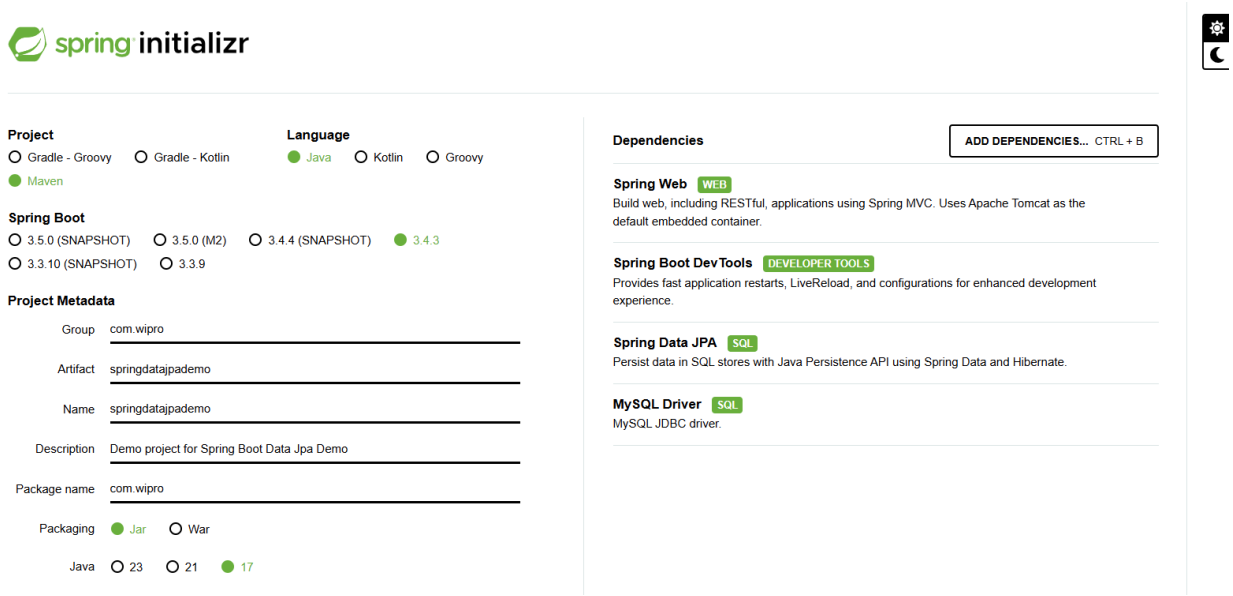
```
@Transactional

public void updateBookDetails(Book book) {

    bookRepository.save(book);

}
```

✔️ Ensures that if an error occurs, changes are **rolled back** automatically.

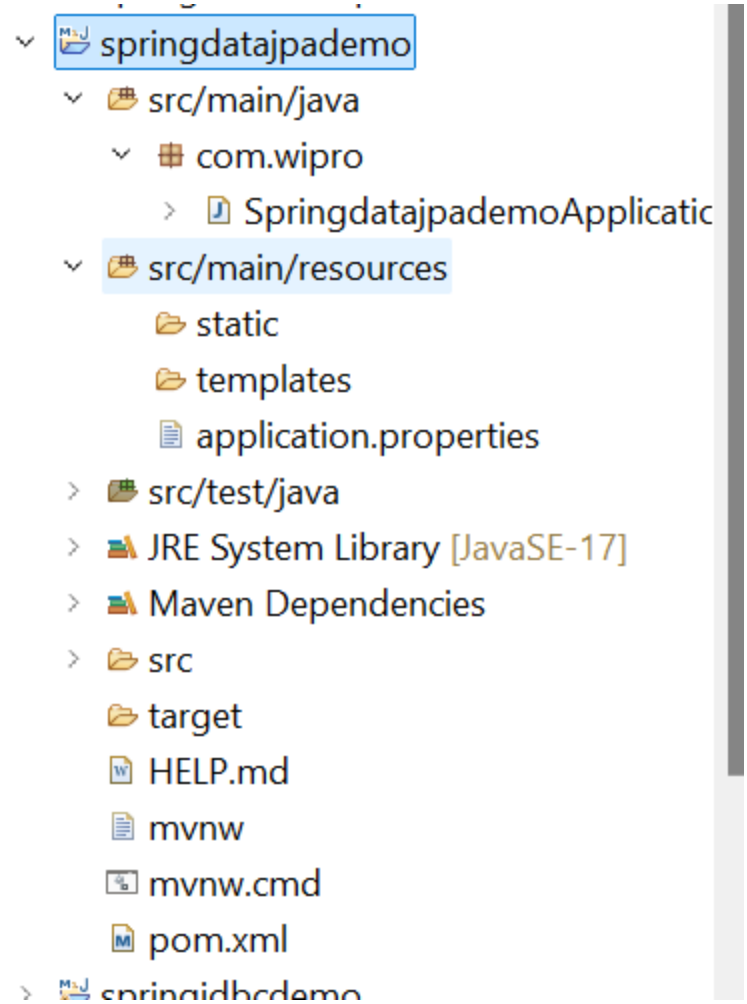
Why Are We Moving to Spring Data JPA?

1. **Faster Development** → No need to write SQL queries manually.
2. **Less Boilerplate Code** → Just define methods, and queries are auto-generated.
3. **Scalability** → Supports pagination, caching, and transactions.
4. **Database Independence** → Works with MySQL, PostgreSQL, Oracle, etc.
5. **Better Maintainability** → Code is clean, structured, and easy to maintain.
6. **Seamless Integration** → Works well with Spring Boot, Hibernate, and other Spring modules.

Spring Data JPA makes it easy to interact with databases in a **simple and automated way**. Instead of writing SQL queries, you just **define Java classes and interfaces**, and Spring handles the rest!

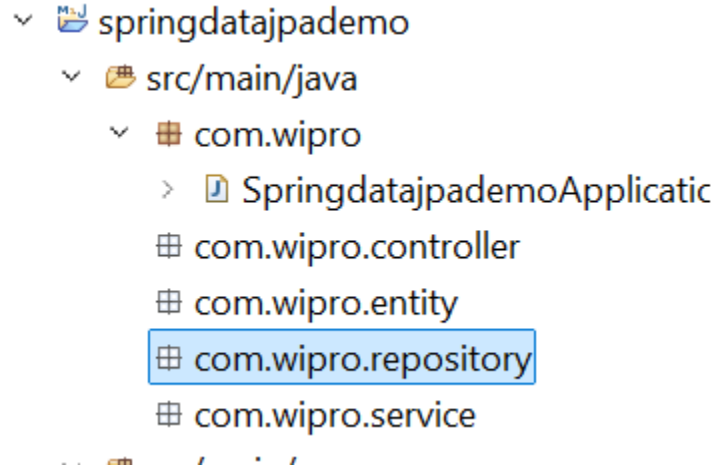


Now click on the generate a zip file will be downloaded…extract it ad import into the eclipse



This is the default files are generated

Now create the 4 packages named as **controller,entity,service and repository**



1.Whenever we make a request then it goes to the controller layer(Handling the http user request)
It is the top most layer(Presentation layer)

1. Presentation Layer (Controller Layer)

- This is the topmost layer responsible for handling user input.
- It processes HTTP ,, ests and sends responses.
- It interacts with the Service Layer to perform business logic.

2.From the controller layer it will go to the service layer

2. Business Layer (Service Layer)

- This layer contains business logic and rules.
- It processes data before saving or retrieving it.
- It communicates with the **Repository Layer** for database operations.

3.From the service layer it will go to the Repository layer

3. Data Access Layer (Repository Layer)

- It interacts with the database using JPA (or JDBC).
- This layer performs CRUD (Create, Read, Update, Delete) operations.

4.Repository will talk to the data base

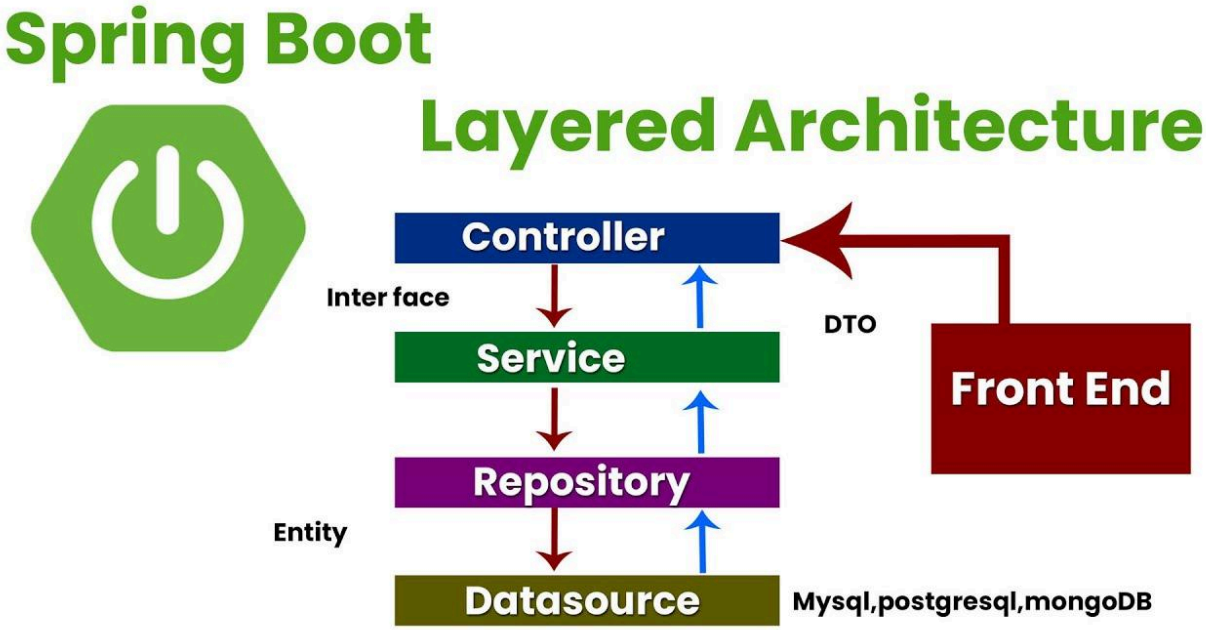
4. Persistence Layer (Database Layer)

- This is where actual data is stored.
- It includes relational databases like MySQL, PostgreSQL, or NoSQL databases like MongoDB.

How These Layers Interact?

1. **Client makes a request** → Hits the **Controller Layer**.
2. **Controller calls the Service Layer** → Business logic is executed.
3. **Service Layer calls Repository Layer** → Data is fetched or saved in the database.
4. **Repository interacts with the Database Layer** → Retrieves or updates data.
5. **Data flows back up** → From Repository → Service → Controller → Response is sent to the client.

This layered architecture ensures: ✅ **Separation of Concerns** – Each layer has a clear responsibility.
✅ **Reusability** – The service layer can be used by multiple controllers.
✅ **Maintainability** – Changes in one layer do not affect others.



Now we r going to create the employee class in the entity package

*Employee.java

```java
package com.wipro.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity //every entity class is annotated with entity
public class Employee
{
    //every entity class must contain the primary key
    @Id
    //generate the pk
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String department;
    public Employee()
    {
    }
    //generate the 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;
    }
    //generate the constructors without id bcoz id is pk..it is autogen
    public Employee(String name, String department) {
        super();
        this.name = name;
        this.department = department;
    }
```

Writable | Smart

Now we r going to create the controller class named as EmployeeController

```java
package com.wipro.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController //controller clss must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class



}

```

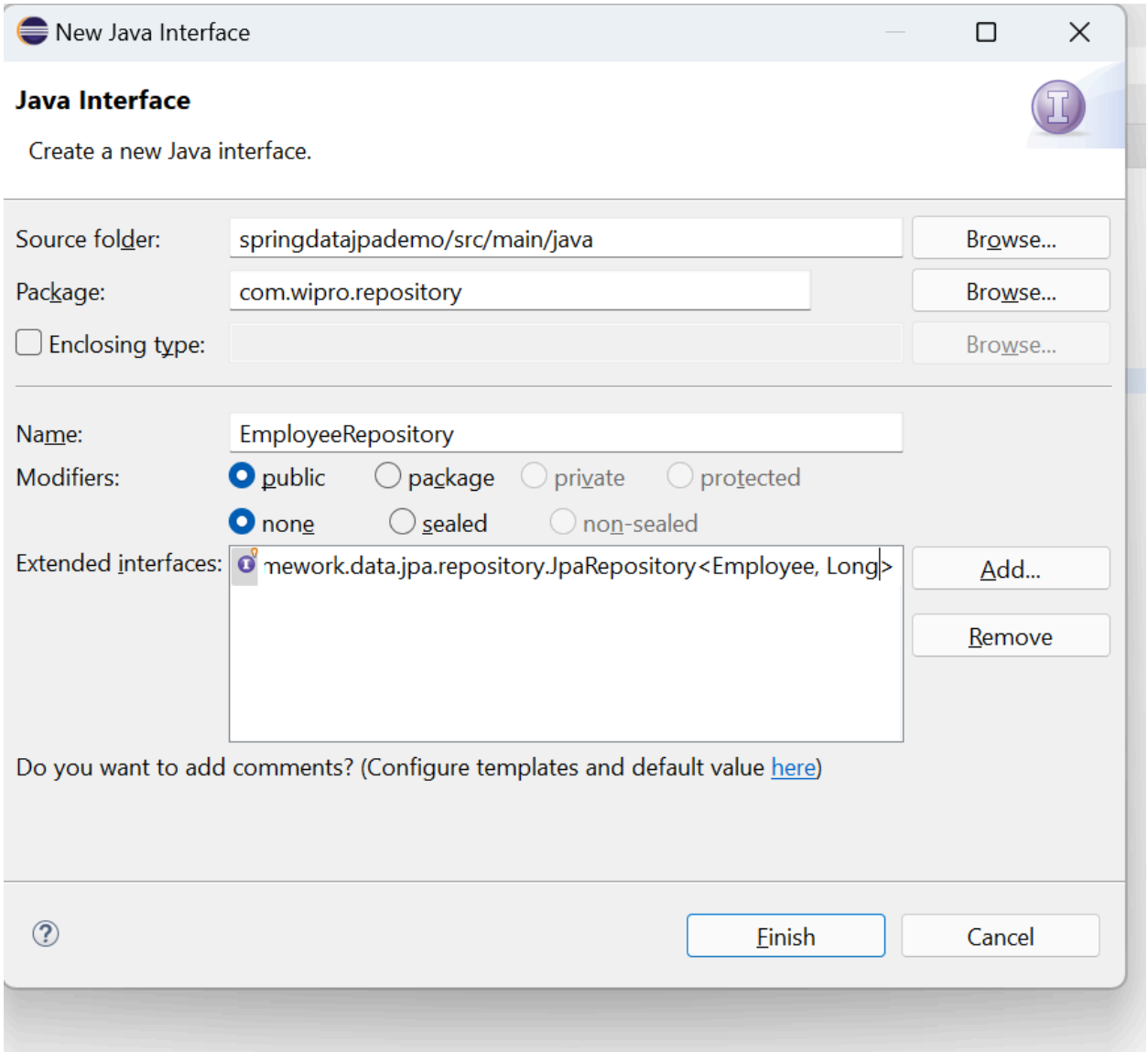
Now we need to create the service class bcoz it is connected to service layer

```java
package com.wipro.service;

import org.springframework.stereotype.Service;

@Service
public class EmployeeService {

}

```

Again this service class is depending on the repository so we need to create the repository method in the above class and create the repository interface in repo package

```java
package com.wipro.service;

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

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;
}

```

Now we need to create the repository interface



```java
package com.wipro.repository;

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> {

}

```

Now my requirement is,i want to insert the data
Before that we need to configure the details in the application.properties

EmployeeServ... EmployeeRepo... application....

```properties
spring.application.name=springdatajpademo
server.port=9090
#database configuration
spring.datasource.url=jdbc:mysql://localhost:3306/wipro
spring.datasource.username="root"
spring.datasource.username=#Mahadev7"

#Hibernate properties
#used to generate those queries
spring.jpa.database-platform=org.hiberante.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
#table exist it will drop or created
spring.jpa.show-sql=true
spring.jpa.format-sql=true
```

Now go to the controller class and create the employee

```java
package com.wipro.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

@RestController //controller clss must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class

    //create the employeee

    @PostMapping // used to create
    public Employee createEmployee(@RequestBody Employee employee)
    {
        return service.addEmployee(employee);   //we didn't mentioned the add emp method in service class ..so we have to
    }

}
```

```java
package com.wipro.service;

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

import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;

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

Now run the SpringDatajpaDemoapplication

Now go and update the data in postman

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
3
4 "name":"prem nath",
5 "department":"medical coder"
6
7 }
```

As soon as we removed the id and we modified name into prem nath and dept into medical coder

It is autogenerating the id into 3

Body Cookies Headers (5) Test Results

{} JSON Preview Visualize

```
1 {
2     "id": 3,
3     "name": "prem nath",
4     "department": "medical coder"
5 }
```

Now check in the mysql

```
1 •  SELECT * FROM wipro.employee;
```

Result Grid Filter Rows: Edit:

id	department	name
1	medical coder	prem nath
2	medical coder	prem nath
3	medical coder	prem nath
NULL	NULL	NULL

Now i am providing the overall codes

Employee.java × | EmployeeController.java | EmployeeService.java | EmployeeR

```java
package com.wipro.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity //every entity class is annotated with entity
public class Employee
{
    //every entity class must contain the primary key
    @Id
    //generate the pk
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String department;
    public Employee()
    {
    }
    //generate the 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;
    }
    //generate the constructors without id bcoz id is pk..it is autogen
    public Employee(String name, String department) {
        super();
        this.name = name;
        this.department = department;
    }
```

Writable | Smart I

Employee.java | EmployeeController.java | EmployeeService.java | EmployeeRepository.java | application.properties | Sprin

```java
package com.wipro.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

@RestController //controller clss must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class

    //create the employeee

    @PostMapping // used to create
    public Employee createEmployee(@RequestBody Employee employee)
    {
        return service.addEmployee(employee);   //we didn't mentioned the add emp method in service class ..so we have to
    }

}
```

```java
package com.wipro.service;

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

import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;

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

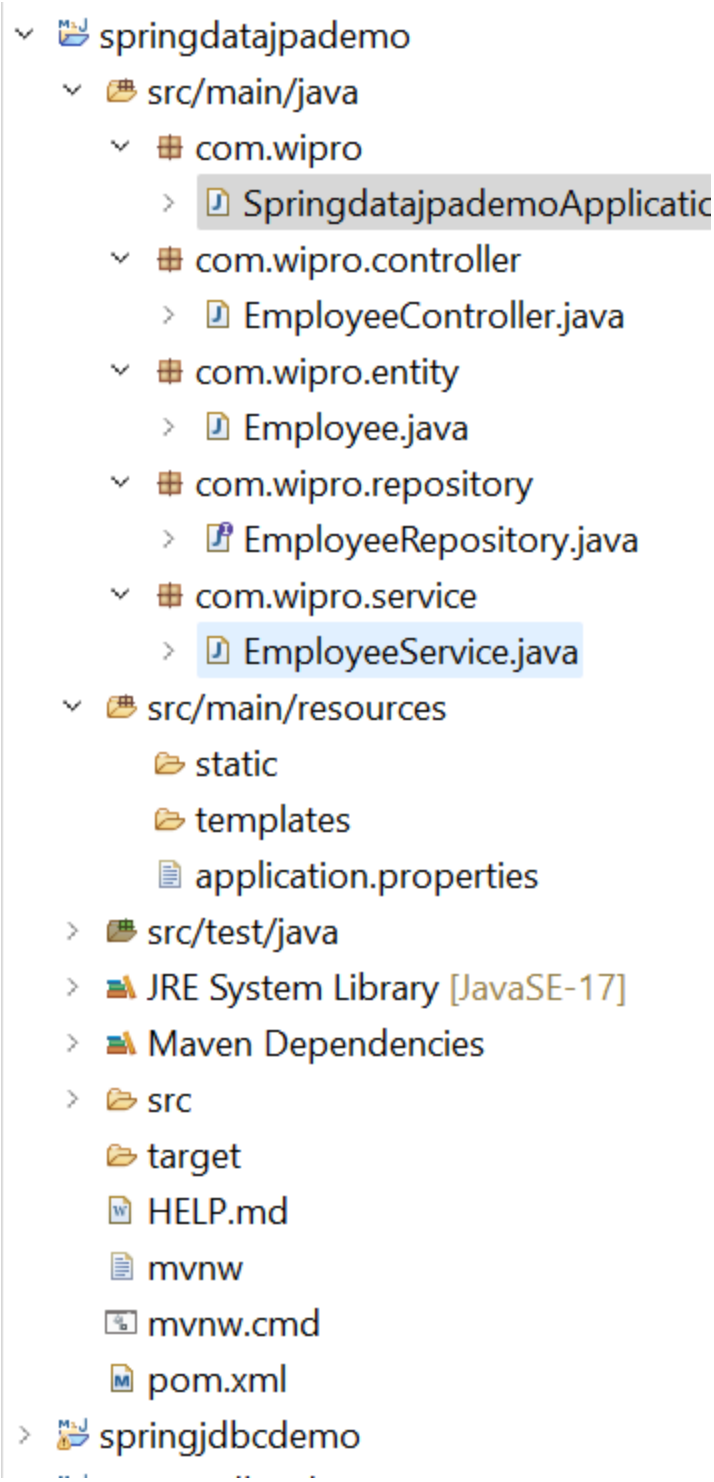
Employee.java | EmployeeController.java | EmployeeService.java | EmployeeRepository.java

```java
package com.wipro.repository;

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> {

}

```

Employee.java | EmployeeController.java | EmployeeService.java | EmployeeRepository.java | application.properties

```properties
spring.application.name=springdatajpademo
server.port=9090
#database configuration
spring.datasource.url=jdbc:mysql://localhost:3306/wipro
spring.datasource.username=root
spring.datasource.password=#Mahadev7

#Hibernate properties
#used to generate those queries
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=create
 #table exist it will drop or created
spring.jpa.show-sql=true
spring.jpa.format-sql=true


```

```java
package com.wipro;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringdatajpademoApplication {

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

}

```



Inserted the data in postman

POST localhost:9090/employee

Params Authorization Headers (8) Body Scripts Settin

none form-data x-www-form-urlencoded raw bir

```
1 {
2
3
4 "name":"prem nath",
5 "department":"medical coder"
6
7 }
```

Body Cookies Headers (5) Test Results

JSON Preview Visualize

```
1 {
2     "id": 3,
3     "name": "prem nath",
4     "department": "medical coder"
5 }
```

To get the all employee details we need to mention one method in EmployeeController

```
29
30      //get all employees
31
32      public List<Employee> getAllEmployees()
33      {
34          return service.getEmployees();
35      }
36
37  }
38
```

Now define the service method in the service class to avoid errors

```
20
21      public List<Employee> getEmployees() {
22          return repository.findAll();
23      }
24  }
25
```

Now check this by running the application

My data contains pavan and prem..now i will retrieve them based on the getAll

GET localhost:9090/employee Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Body Cookies Headers (5) Test Results 200 OK 11 ms 265 B

{} JSON Preview Visualize


```
[
    {
        "id": 1,
        "name": "pavan",
        "department": "software"
    },
    {
        "id": 2,
        "name": "prem",
        "department": "medical coder"
    }
]
```

```java
package com.wipro.service;

import java.util.List;

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

import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;

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

    public List<Employee> getEmployees() {
        return repository.findAll();
    }
}
```

```java
package com.wipro.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

@RestController //controller clss must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class

    //create the employeee

    @PostMapping // used to create
    public Employee createEmployee(@RequestBody Employee employee)
    {
        return service.addEmployee(employee);   //we didn't mentioned the add emp method in service class ..so we have to
    }

    //get all employees
    @GetMapping
    public List<Employee> getAllEmployees()
    {
        return service.getEmployees();
    }

}
```

Remaining codes is asusual

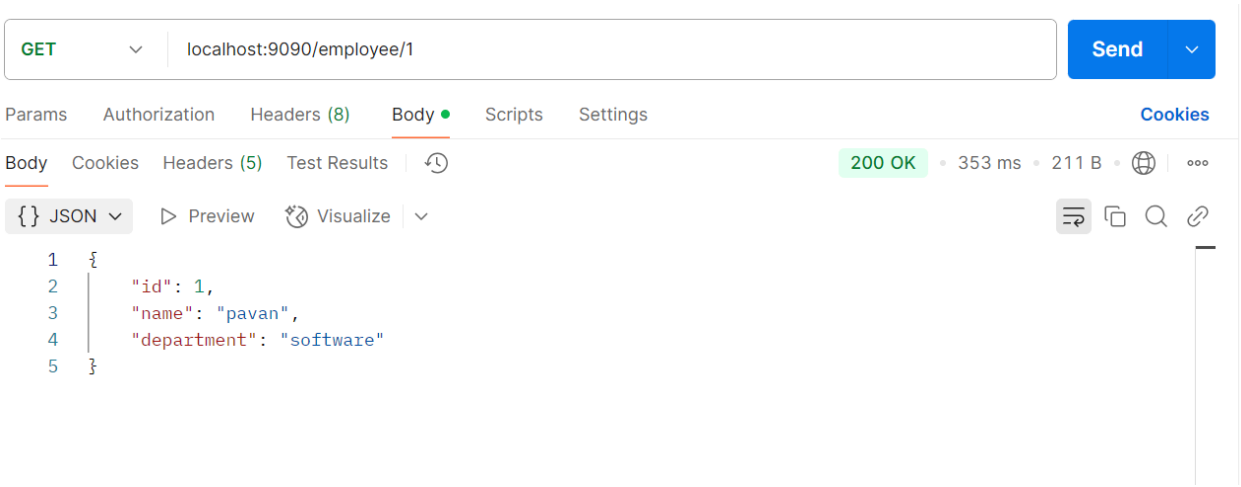
Now we r going to retrieve the data based on the id

```java

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


```

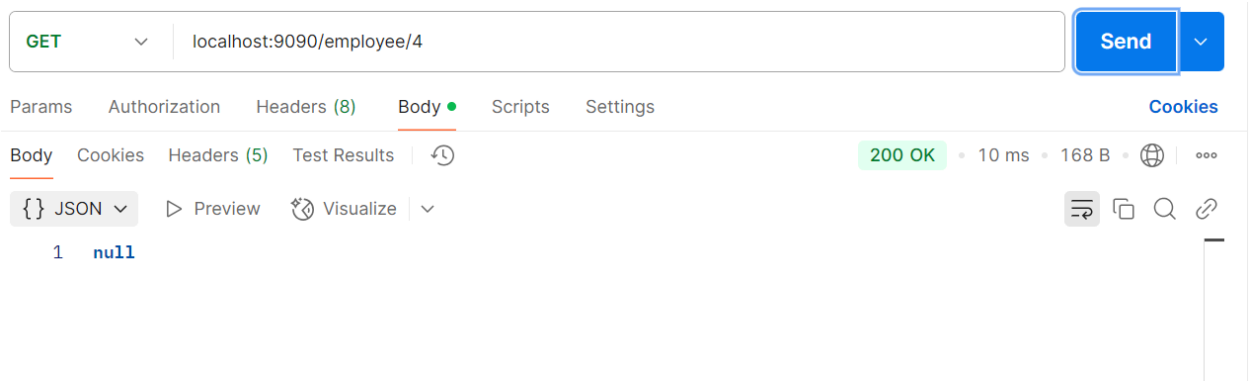
in the employee controller….mention the method in the service class also

```java
    public Optional<Employee> getEmployeeById(Long id) {
        return repository.findById(id);

    }
}
```

Now run the application



For suppose if we mention the unknown id then it returns null



```java
package com.wipro.controller;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
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.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

@RestController //controller class must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class

    //create the employeee

    @PostMapping // used to create
    public Employee createEmployee(@RequestBody Employee employee)
    {
        return service.addEmployee(employee);   //we didn't mentioned the add emp method in service class ..so we have to
    }

    //get all employees
    @GetMapping
    public List<Employee> getAllEmployees()
    {
        return service.getEmployees();
    }


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


}
```

```java
package com.wipro.service;

import java.util.List;
import java.util.Optional;

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

import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;

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

    public List<Employee> getEmployees() {
        return repository.findAll();
    }

    public Optional<Employee> getEmployeeById(Long id) {
        return repository.findById(id);

    }
}
```

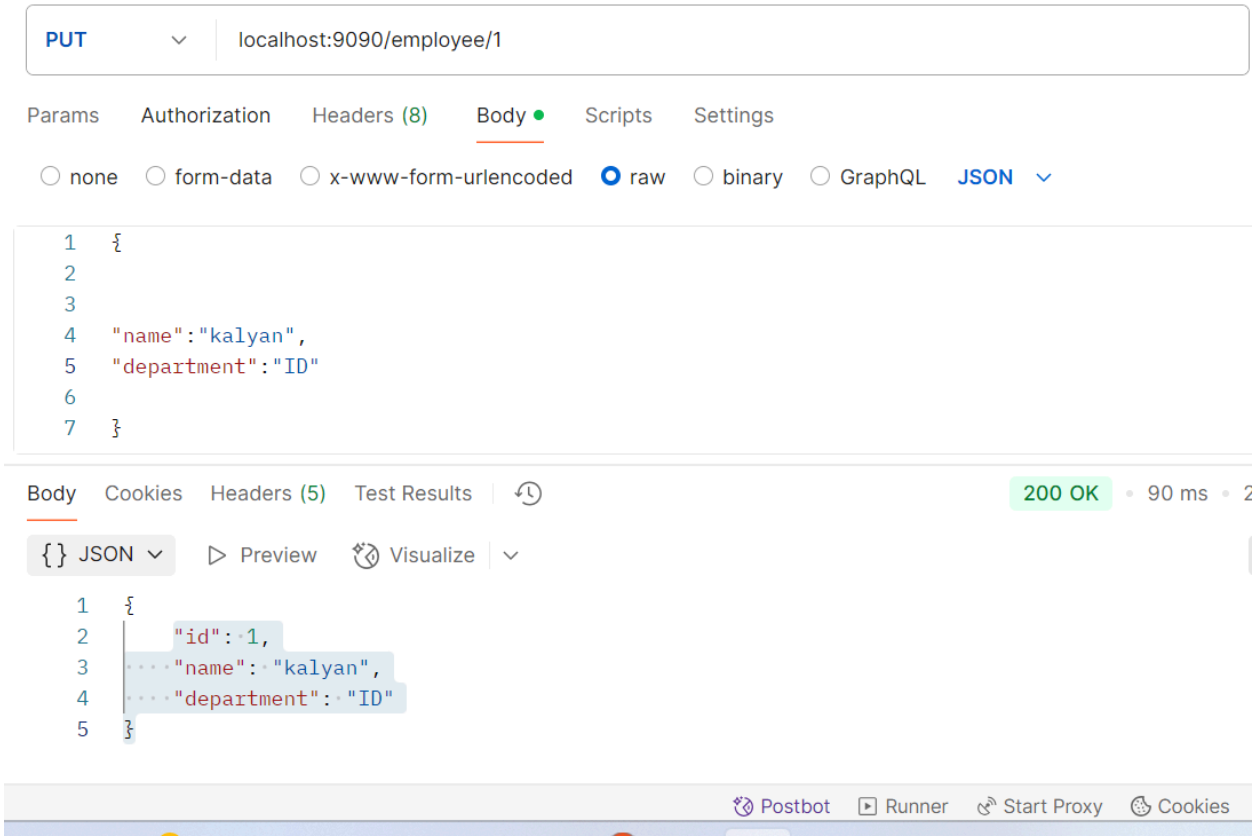
Now we have to **update the** employee

```java
//update employeee

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

As this method is not declared in the service class..we need to mention it

```java
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);
}
```

Updating with the postman



```
1 •  SELECT * FROM wipro.employee;
```

Result Grid | Filter Rows: | Edit:

id	department	name
1	ID	kalyan
2	medical coder	prem
NULL	NULL	NULL

```java
package com.wipro.controller;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
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.entity.Employee;
import com.wipro.service.EmployeeService;

@RestController //controller clss must be annotaated with RC
@RequestMapping("/employee") //this is the base url
public class EmployeeController {

    //our controller is connected to service layer so we need to autowire
    @Autowired
    public EmployeeService service; //now we need to create the service class

    //create the employeee

    @PostMapping // used to create
    public Employee createEmployee(@RequestBody Employee employee)
    {
        return service.addEmployee(employee);   //we didn't mentioned the add emp method in service class ..so we have to
    }

    //get all employees
    @GetMapping
    public List<Employee> getAllEmployees()
    {
        return service.getEmployees();
    }


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


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

    //update employeee

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



}
```

Writable | Smart Insert | 1 : 30 : 29

Above code is employe controller

Employee.java | *EmployeeController.java | EmployeeService.java | EmployeeRepository.j

```java
package com.wipro.service;

import java.util.List;
import java.util.Optional;

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

import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@Service
public class EmployeeService
{
    @Autowired
    private EmployeeRepository repository;

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

    public List<Employee> getEmployees() {
        return repository.findAll();
    }

    public Optional<Employee> getEmployeeById(Long id) {
        return repository.findById(id);

    }

    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);
    }

}
```

employee services

Now perform the delete operation

```
55
56      //delete by id
57      // Delete Employee by ID
58      @DeleteMapping("/{id}")
59      public void deleteEmployeeById(@PathVariable Long id) {
60          service.deleteEmployeeById(id);
61      }
62
63
64
```

```
public void deleteEmployeeById(Long id) {
    repository.deleteById(id);
}
```

DELETE localhost:9090/employee/1 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

```
1  {
2
3
4  "name":"kalyan",
5  "department":"ID"
6
7  }
```

Body Cookies Headers (4) Test Results 200 OK 68 ms 123 B

Raw Preview Visualize

```
1
```

Instead of deleting the data directly we can crosscheck also with the help of the **existByID()**

```java
    //delete  by id
    @DeleteMapping("/{id}")
    public String deleteEmployeeById(@PathVariable Long id) {
        return service.deleteEmployeeById(id);
    }




}
```

```java

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


}
```

Writable | Smar

DELETE localhost:9090/employee/2 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

```
{


"name":"kalyan",
```

Body Cookies Headers (5) Test Results 200 OK 415 ms 202 B

Raw Preview Visualize

```
employee with2 is deleted successfully
```

DELETE localhost:9090/employee/20 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

```
1 {
2
3
4 "name":"kalyan",
```

Body Cookies Headers (5) Test Results 200 OK 10 ms 182 B

Raw Preview Visualize

```
1 employee not found
```

Req: Fetch the employee based on the name and department (non primary key columns)

```java
package com.wipro.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@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
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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

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

    @Autowired
    private EmployeeService service;

    // Create Employee
    @PostMapping
    public Employee createEmployee(@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, @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);
    }
}
```

Writable | Smart

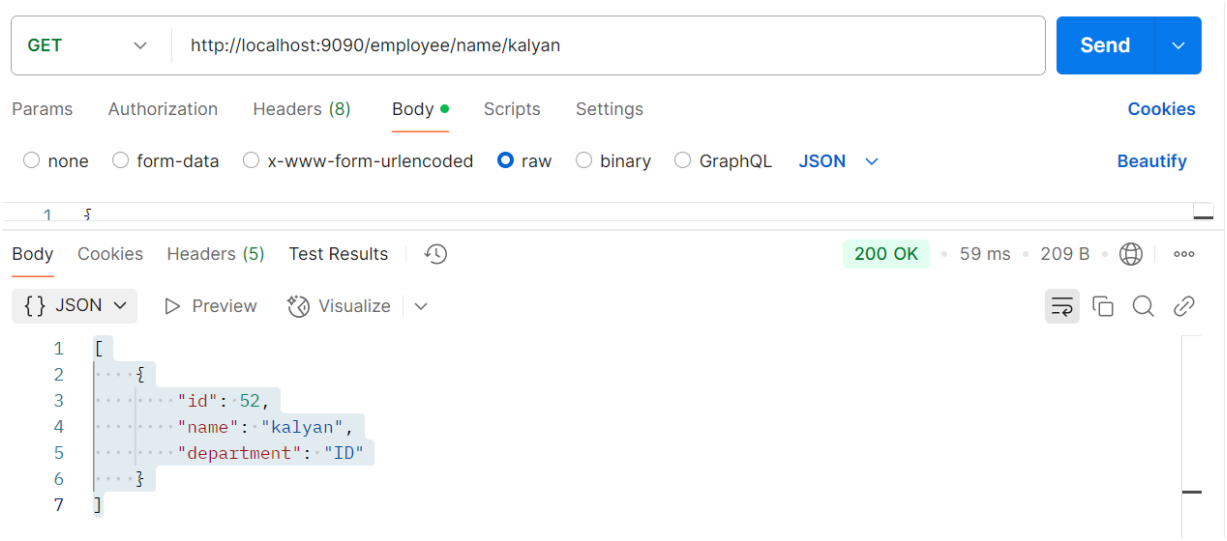
```java
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;

@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);
    }
}
```

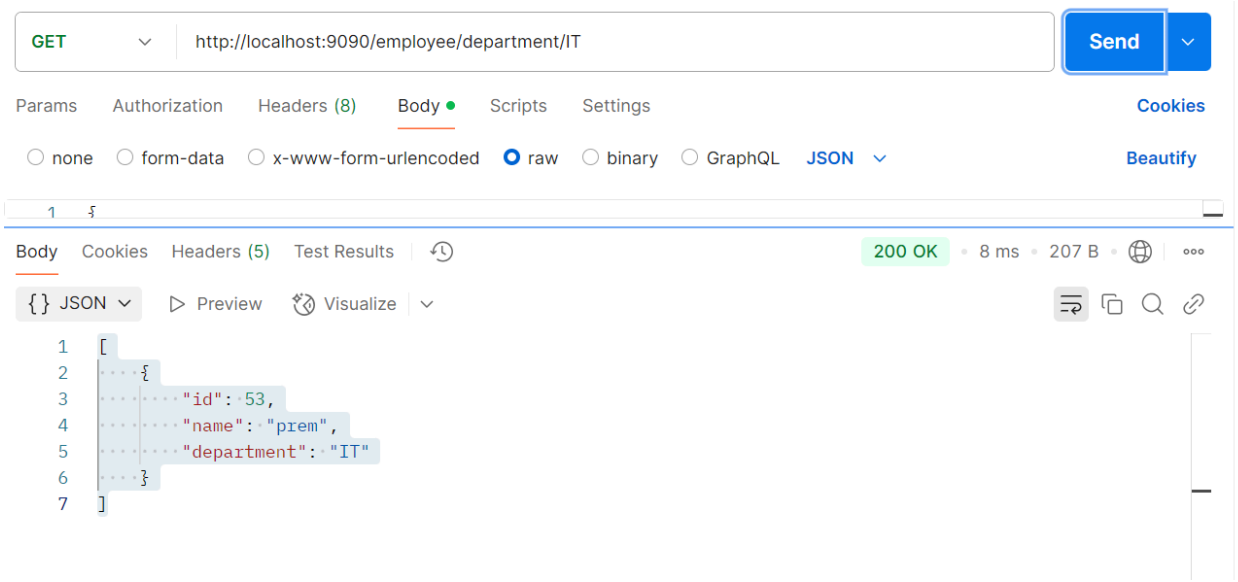
Writable

```java
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);
}

```

```properties
# Application Name
spring.application.name=springdatajpademo

# Server Port
server.port=9090

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/wipro
spring.datasource.username=root
spring.datasource.password=#Mahadev7

# Hibernate Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.format-sql=true

```

```java
package com.wipro;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

```





Pagination

What is Pagination?

Pagination is the **process of dividing large datasets into smaller chunks** (pages) and retrieving only a limited number of records per request. This improves performance by:

- Reducing the load on the database.

- Enhancing response times for API calls.
- Allowing users to navigate through data in pages.

Spring Data JPA's Pagination Support

Spring provides a built-in interface called `Pageable` to implement pagination. The `Pageable` interface helps to:

- Define the **page number** and **size**.
- Retrieve a subset of data from the database.

Spring also provides a `Page` interface, which contains:

- The actual list of records.
- Total number of pages.
- Total number of elements.
- Metadata about pagination.

Sorting

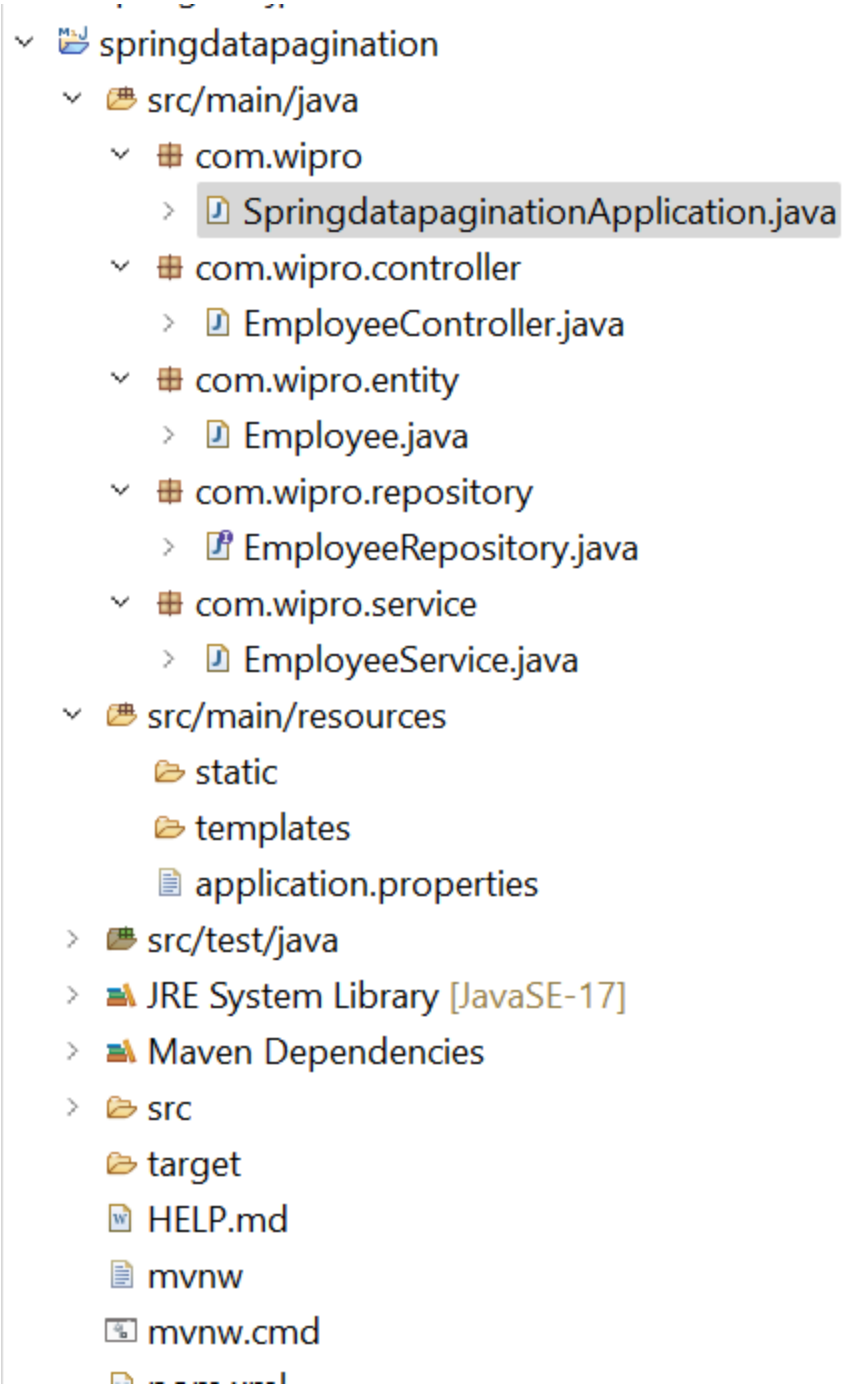
What is Sorting?

Sorting is the process of ordering data based on one or more fields (e.g., sorting employees by **name** or **salary**).
Spring Data JPA provides an interface called `Sort`, which allows sorting in:

- **Ascending order** (`Sort.by("fieldName").ascending()`)
- **Descending order** (`Sort.by("fieldName").descending()`)

Sorting is often combined with pagination to fetch sorted data in smaller chunks.

Download the same from spring.start.io then import that zip file


springdatapagination
src/main/java
com.wipro
SpringdatapaginationApplication.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

application.pro... × Employee.java EmployeeReposito... EmployeeService.... Em

```
spring.application.name=springdatapagination
server.port=9090

# MySQL Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/wipro
spring.datasource.username=root
spring.datasource.password=#Mahadev7

# Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

```
package com.wipro.entity;
import jakarta.persistence.*;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String department;

    public Employee() {}

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

    public Long getId() { return 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; }
}
```

application.pro... | Employee.java | EmployeeReposito... | EmployeeService.... | EmployeeCon

```java
package com.wipro.repository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
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> {
    Page<Employee> findAll(Pageable pageable);  // Pagination support
}
```

application.pro... | Employee.java | EmployeeReposito... | EmployeeService.... | EmployeeControll... | Springdatapagin...

```java
package com.wipro.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;
import com.wipro.entity.Employee;
import com.wipro.repository.EmployeeRepository;
import java.util.List;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    // Pagination
    public Page<Employee> getEmployeesWithPagination(int page, int size) {
        Pageable pageable = PageRequest.of(page, size);
        return repository.findAll(pageable);
    }

    // Sorting
    public List<Employee> getEmployeesWithSorting(String field) {
        return repository.findAll(Sort.by(Sort.Direction.ASC, field));  // Sorting in ascending order
    }

    // Pagination + Sorting
    public Page<Employee> getEmployeesWithPaginationAndSorting(int page, int size, String field) {
        Pageable pageable = PageRequest.of(page, size, Sort.by(Sort.Direction.ASC, field));
        return repository.findAll(pageable);
    }
}
```

application.pro... | Employee.java | EmployeeReposito... | EmployeeService.... | EmployeeControll... | Springdatapagin...

```java
package com.wipro.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;
import com.wipro.entity.Employee;
import com.wipro.service.EmployeeService;
import java.util.List;

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

    @Autowired
    private EmployeeService service;

    // Pagination API
    @GetMapping("/pagination/{page}/{size}")
    public Page<Employee> getEmployeesWithPagination(@PathVariable int page, @PathVariable int size) {
        return service.getEmployeesWithPagination(page, size);
    }

    // Sorting API
    @GetMapping("/sort/{field}")
    public List<Employee> getEmployeesWithSorting(@PathVariable String field) {
        return service.getEmployeesWithSorting(field);
    }

    // Pagination & Sorting Combined
    @GetMapping("/pagination/{page}/{size}/sort/{field}")
    public Page<Employee> getEmployeesWithPaginationAndSorting(@PathVariable int page,
                                                               @PathVariable int size,
                                                               @PathVariable String field) {
        return service.getEmployeesWithPaginationAndSorting(page, size, field);
    }
}
```

Writable | Smart Insert | 1 : 30 : 29

```java

package com.wipro;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringdatapaginationApplication {

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

}

```

Get the employees with the pagination

GET http://localhost:9090/employee/pagination/0/2 Send


```
{
    "content": [
        {
            "id": 52,
            "name": "kalyan",
            "department": "ID"
        },
        {
            "id": 53,
```

```
        "name": "prem",
        "department": "IT"
    }
],
"pageable": {
    "pageNumber": 0,
    "pageSize": 2,
    "sort": {
        "empty": true,
        "sorted": false,
        "unsorted": true
    },
    "offset": 0,
    "paged": true,
    "unpaged": false
},
"last": true,
"totalPages": 1,
"totalElements": 2,
"size": 2,
"number": 0,
"sort": {
    "empty": true,
    "sorted": false,
```

```
        "unsorted": true
    },
    "first": true,
    "numberOfElements": 2,
    "empty": false
}
```


Sort by ascending order