

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

① 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.

2 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.

3 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';
```

4 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**.

5 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!

6 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!



Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Maven

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.5.0 (SNAPSHOT) ☐ 3.5.0 (M2) ☐ 3.4.4 (SNAPSHOT) ☒ 3.4.3 ☐ 3.3.10 (SNAPSHOT) ☐ 3.3.9

Project Metadata

Group

com.wipro

Artifact

springdatajpademo

Name

springdatajpademo

Description

Demo project for Spring Boot Data Jpa Demo

Package name

com.wipro

Packaging

☒ Jar ☐ War

Java

☐ 23 ☐ 21 ☒ 17

Dependencies

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Data JPA SQL

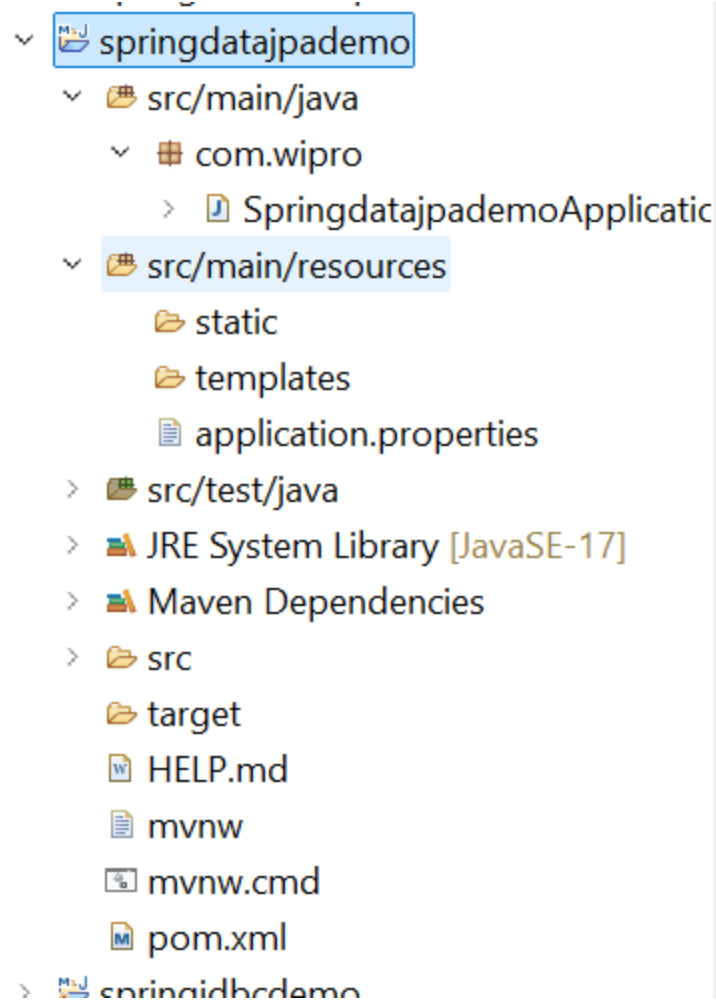
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL

MySQL JDBC driver.

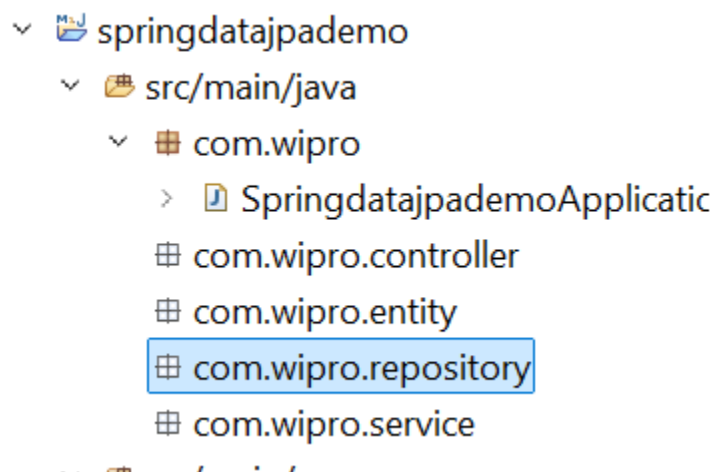
ADD DEPENDENCIES... CTRL + B

Now click on the generate a zip file will be downloaded...extract it and 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 requests 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.

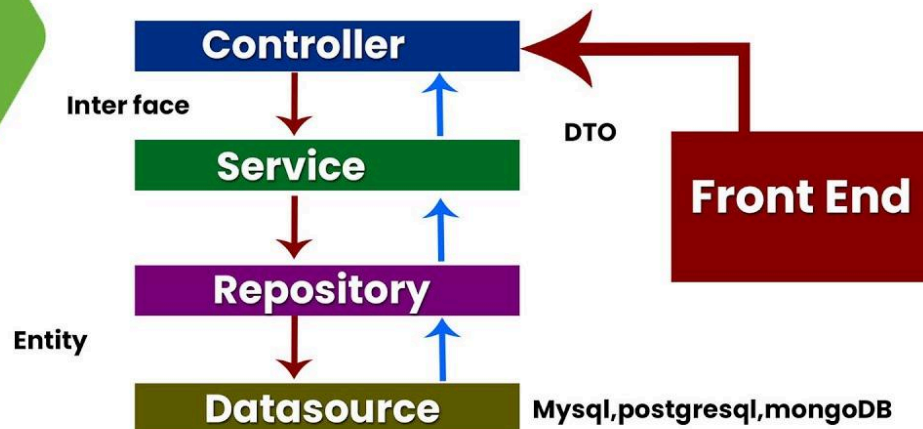
☒ **Reusability** – The service layer can be used by multiple controllers.

☒ **Maintainability** – Changes in one layer do not affect others.

Spring Boot



Layered Architecture



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

```
*Employee.java ×
1 package com.wipro.entity;
2 import jakarta.persistence.Entity;
3 import jakarta.persistence.GeneratedValue;
4 import jakarta.persistence.GenerationType;
5 import jakarta.persistence.Id;
6 @Entity //every entity class is annotated with entity
7 public class Employee
8 {
9     //every entity class must contain the primary key
10    @Id
11    //generate the pk
12    @GeneratedValue(strategy = GenerationType.AUTO)
13    private Long id;
14    private String name;
15    private String department;
16    public Employee()
17    {
18    }
19    //generate the getters and setters
20    public Long getId() {
21        return id;
22    }
23    public void setId(Long id) {
24        this.id = id;
25    }
26
27    public String getName() {
28        return name;
29    }
30    public void setName(String name) {
31        this.name = name;
32    }
33    public String getDepartment() {
34        return department;
35    }
36    public void setDepartment(String department) {
37        this.department = department;
38    }
39    //generate the constructors without id bcoz id is pk..it is autogen
40    public Employee(String name, String department) {
41        super();
42        this.name = name;
43        this.department = department;
44    }
45 }
```

Writable Smart

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


```

1 package com.wipro.controller;|
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController //controller class must be annotaated with RC
8 @RequestMapping("/employee") //this is the base url
9 public class EmployeeController {
10
11     //our controller is connected to service layer so we need to autowire
12     @Autowired
13     public EmployeeService service; //now we need to create the service class
14
15
16
17 }
18

```

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

```

1 package com.wipro.service;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class EmployeeService {
7
8 }
9

```

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

```

1 package com.wipro.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 @Service
7 public class EmployeeService
8 {
9     @Autowired
10    private EmployeeRepository repository;
11 }
12

```

Now we need to create the repository interface

New Java Interface

Java Interface

Create a new Java interface.

Source folder:

springdatajpademo/src/main/java

Browse...

Package:

com.wipro.repository

Browse...

☐ Enclosing type:

Browse...

Name:

EmployeeRepository

Modifiers:

☒ public
☐ package
☐ private
☐ protected

☒ none
☐ sealed
☐ non-sealed

Extended interfaces:

Add...

Remove

Do you want to add comments? (Configure templates and default value [here](#))

?

Finish

Cancel

```

1 package com.wipro.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.wipro.entity.Employee;
7
8 @Repository
9 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
10
11 }
12

```

Now my requirement is, i want to insert the data

Before that we need to configure the details in the application.properties

```

1 spring.application.name=springdatajpademo
2 server.port=9090
3 #database configuration
4 spring.datasource.url=jdbc:mysql://localhost:3306/wipro
5 spring.datasource.username="root"
6 spring.datasource.password=#Mahadev7"
7
8 #Hibernate properties
9 #used to generate those queries
10 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
11 spring.jpa.hibernate.ddl-auto=update
12 #table exist it will drop or created
13 spring.jpa.show-sql=true
14 spring.jpa.format-sql=true
15
16
17

```

Now go to the controller class and create the employee

```

1 package com.wipro.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.PostMapping;
5 import org.springframework.web.bind.annotation.RequestBody;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import com.wipro.entity.Employee;
10 import com.wipro.service.EmployeeService;
11
12 @RestController //controller class must be annotated with RC
13 @RequestMapping("/employee") //this is the base url
14 public class EmployeeController {
15
16     //our controller is connected to service layer so we need to autowire
17     @Autowired
18     private EmployeeService service; //now we need to create the service class
19
20     //create the employee
21
22     @PostMapping // used to create
23     public Employee createEmployee(@RequestBody Employee employee)
24     {
25         return service.addEmployee(employee); //we didn't mentioned the add emp method in service class ..so we have to
26     }
27
28 }
29

```

```

1 package com.wipro.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 import com.wipro.entity.Employee;
7 import com.wipro.repository.EmployeeRepository;
8
9 @Service
10 public class EmployeeService
11 {
12     @Autowired
13     private EmployeeRepository repository;
14
15     public Employee addEmployee(Employee employee) {
16         return repository.save(employee);
17     }
18 }

```

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

```
1 package com.wipro.entity;
2 import jakarta.persistence.Entity;
3 import jakarta.persistence.GeneratedValue;
4 import jakarta.persistence.GenerationType;
5 import jakarta.persistence.Id;
6 @Entity //every entity class is annotated with entity
7 public class Employee
8 {
9     //every entity class must contain the primary key
10    @Id
11    //generate the pk
12    @GeneratedValue(strategy = GenerationType.AUTO)
13    private Long id;
14    private String name;
15    private String department;
16    public Employee()
17    {
18    }
19    //generate the getters and setters
20    public Long getId() {
21        return id;
22    }
23    public void setId(Long id) {
24        this.id = id;
25    }
26
27    public String getName() {
28        return name;
29    }
30    public void setName(String name) {
31        this.name = name;
32    }
33    public String getDepartment() {
34        return department;
35    }
36    public void setDepartment(String department) {
37        this.department = department;
38    }
39    //generate the constructors without id bcoz id is pk..it is autogen
40    public Employee(String name, String department) {
41        super();
42        this.name = name;
43        this.department = department;
44    }
45 }
```

Writabla

Smart I

Employee.java EmployeeController.java × EmployeeService.java EmployeeRepository.java application.properties Spring

```
1 package com.wipro.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.PostMapping;
5 import org.springframework.web.bind.annotation.RequestBody;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import com.wipro.entity.Employee;
10 import com.wipro.service.EmployeeService;
11
12 @RestController //controller class must be annotated with RC
13 @RequestMapping("/employee") //this is the base url
14 public class EmployeeController {
15
16     //our controller is connected to service layer so we need to autowire
17     @Autowired
18     public EmployeeService service; //now we need to create the service class
19
20     //create the employee
21
22     @PostMapping // used to create
23     public Employee createEmployee(@RequestBody Employee employee)
24     {
25         return service.addEmployee(employee); //we didn't mentioned the add emp method in service class ..so we have to
26     }
27 }
28
29
```

```
1 package com.wipro.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 import com.wipro.entity.Employee;
7 import com.wipro.repository.EmployeeRepository;
8
9 @Service
10 public class EmployeeService
11 {
12     @Autowired
13     private EmployeeRepository repository;
14
15     public Employee addEmployee(Employee employee) {
16         return repository.save(employee);
17     }
18 }
19
```



```

Employee.java EmployeeController.java × EmployeeService.java EmployeeRepository.java ×
1 package com.wipro.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.wipro.entity.Employee;
7
8 @Repository
9 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
10
11 }
12

```

```



























Employee.java EmployeeController.java EmployeeService.java EmployeeRepository.java application.properties ×
1 spring.application.name=springdatajpademo
2 server.port=9090
3 #database configuration
4 spring.datasource.url=jdbc:mysql://localhost:3306/wipro
5 spring.datasource.username=root
6 spring.datasource.password=#Mahadev7
7
8 #Hibernate properties
9 #used to generate those queries
10 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
11
12 spring.jpa.hibernate.ddl-auto=create
13 #table exist it will drop or created
14 spring.jpa.show-sql=true
15 spring.jpa.format-sql=true
16
17
18

```

```

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

```

- ▼  springdatajpademo
 - ▼  src/main/java
 - ▼  com.wipro
 - >  SpringdatajpademoApplicatic
 - ▼  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
- >  springjdbcdemo

Inserted the data in postman

POST

localhost:9090/employee

Params

Authorization

Headers (8)

Body ●

Scripts

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

```
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

```

39
40
41 public List<Employee> getEmployees() {
42     return repository.findAll();
43 }
44 }
45

```

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

```
1  [  
2    {  
3      "id": 1,  
4      "name": "pavan",  
5      "department": "software"  
6    },  
7    {  
8      "id": 2,  
9      "name": "prem",  
10     "department": "medical coder"  
11   }  
12 ]
```

```
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.entity.Employee;
9 import com.wipro.repository.EmployeeRepository;
10
11 @Service
12 public class EmployeeService
13 {
14     @Autowired
15     private EmployeeRepository repository;
16
17     public Employee addEmployee(Employee employee) {
18         return repository.save(employee);
19     }
20
21     public List<Employee> getEmployees() {
22         return repository.findAll();
23     }
24 }
25
```

```

1 package com.wipro.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PostMapping;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import com.wipro.entity.Employee;
13 import com.wipro.service.EmployeeService;
14
15 @RestController //controller class must be annotated with RC
16 @RequestMapping("/employee") //this is the base url
17 public class EmployeeController {
18
19     //our controller is connected to service layer so we need to autowire
20     @Autowired
21     public EmployeeService service; //now we need to create the service class
22
23     //create the employee
24
25     @PostMapping // used to create
26     public Employee createEmployee(@RequestBody Employee employee)
27     {
28         return service.addEmployee(employee); //we didn't mention the add emp method in service class ..so we have to
29     }
30
31     //get all employees
32     @GetMapping
33     public List<Employee> getAllEmployees()
34     {
35         return service.getEmployees();
36     }
37
38 }
39

```

Remaining codes is as usual

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

```

40
41 //get employee by id
42 @GetMapping("/{id}")
43 public Optional<Employee> getEmployeeById(@PathVariable Long id) {
44     return service.getEmployeeById(id);
45 }
46
47

```

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

```

25
26 public Optional<Employee> getEmployeeById(Long id) {
27     return repository.findById(id);
28 }
29 }
30 }
31

```

Now run the application

GET localhost:9090/employee/1 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Body Cookies Headers (5) Test Results 200 OK • 353 ms • 211 B •

{ } JSON Preview Visualize

```

1 {
2   "id": 1,
3   "name": "pavan",
4   "department": "software"
5 }

```

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

GET localhost:9090/employee/4 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Body Cookies Headers (5) Test Results 200 OK • 10 ms • 168 B •

{ } JSON Preview Visualize

```

1 null

```



```

Employee.java EmployeeController.java EmployeeService.java EmployeeRepository.java ap
1 package com.wipro.controller;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 import com.wipro.entity.Employee;
15 import com.wipro.service.EmployeeService;
16
17 @RestController //controller class must be annotated with RC
18 @RequestMapping("/employee") //this is the base url
19 public class EmployeeController {
20
21     //our controller is connected to service layer so we need to autowire
22     @Autowired
23     public EmployeeService service; //now we need to create the service class
24
25     //create the employee
26
27     @PostMapping // used to create
28     public Employee createEmployee(@RequestBody Employee employee)
29     {
30         return service.addEmployee(employee); //we didn't mentioned the add emp method in service class ..so we have to
31     }
32
33     //get all employees
34     @GetMapping
35     public List<Employee> getAllEmployees()
36     {
37         return service.getEmployees();
38     }
39
40
41     //get employee by id
42     @GetMapping("/{id}")
43     public Optional<Employee> getEmployeeById(@PathVariable Long id) {
44         return service.getEmployeeById(id);
45     }
46
47
48 }
49

```

```

Employee.java      EmployeeController.java      Employees
1  package com.wipro.service;
2
3  import java.util.List;
4  import java.util.Optional;
5
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Service;
8
9  import com.wipro.entity.Employee;
10 import com.wipro.repository.EmployeeRepository;
11
12 @Service
13 public class EmployeeService
14 {
15     @Autowired
16     private EmployeeRepository repository;
17
18     public Employee addEmployee(Employee employee) {
19         return repository.save(employee);
20     }
21
22     public List<Employee> getEmployees() {
23         return repository.findAll();
24     }
25
26     public Optional<Employee> getEmployeeById(Long id) {
27         return repository.findById(id);
28     }
29 }
30
31

```

Now we have to **update the employee**

```

//update employee

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

```

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

```

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

The screenshot shows the Postman interface for a PUT request. The URL is `localhost:9090/employee/1`. The request body is a JSON object: `{ "name": "kalyan", "department": "ID" }`. The response status is `200 OK` with a response time of `90 ms`. The response body is a JSON object: `{ "id": 1, "name": "kalyan", "department": "ID" }`.

Request:

- Method: PUT
- URL: localhost:9090/employee/1
- Body Type: raw (JSON)
- Body:

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




Response:

- Status: 200 OK
- Time: 90 ms
- Body Type: JSON
- Body:

```
{
  "id": 1,
  "name": "kalyan",
  "department": "ID"
}
```

Postman interface elements visible at the bottom include: Postbot, Runner, Start Proxy, and Cookies.

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

Result Grid |   Filter Rows: Edit: 

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

```

1 package com.wipro.controller;
2 import java.util.List;
3 import java.util.Optional;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8 import org.springframework.web.bind.annotation.PostMapping;
9 import org.springframework.web.bind.annotation.PutMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 import com.wipro.entity.Employee;
15 import com.wipro.service.EmployeeService;
16
17 @RestController //controller class must be annotated with RC
18 @RequestMapping("/employee") //this is the base url
19 public class EmployeeController {
20
21     //our controller is connected to service layer so we need to autowire
22     @Autowired
23     public EmployeeService service; //now we need to create the service class
24
25     //create the employee
26
27     @PostMapping // used to create
28     public Employee createEmployee(@RequestBody Employee employee)
29     {
30         return service.addEmployee(employee); //we didn't mentioned the add emp method in service class ..so we have to
31     }
32
33     //get all employees
34     @GetMapping
35     public List<Employee> getAllEmployees()
36     {
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
48     // Update Employee
49     @PutMapping("/{id}")
50     public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
51         return service.updateEmployee(id, updatedEmployee);
52     }
53
54
55
56
57 }
58
37     return service.getEmployees();
38 }
39
40
41 //get employee by id
42 @GetMapping("/{id}")
43 public Optional<Employee> getEmployeeById(@PathVariable Long id) {
44     return service.getEmployeeById(id);
45 }
46
47 //update employee
48
49 // Update Employee
50 @PutMapping("/{id}")
51 public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {
52     return service.updateEmployee(id, updatedEmployee);
53 }
54
55
56
57 }
58

```

Writable Smart Insert 1 : 30 : 29

Above code is employee controller

```

Employee.java EmployeeController.java EmployeeService.java x EmployeeRepository.java
1 package com.wipro.service;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import com.wipro.entity.Employee;
10 import com.wipro.repository.EmployeeRepository;
11
12 @Service
13 public class EmployeeService
14 {
15     @Autowired
16     private EmployeeRepository repository;
17
18     public Employee addEmployee(Employee employee) {
19         return repository.save(employee);
20     }
21
22     public List<Employee> getEmployees() {
23         return repository.findAll();
24     }
25
26     public Optional<Employee> getEmployeeById(Long id) {
27         return repository.findById(id);
28     }
29
30
31     public Employee updateEmployee(Long id, Employee updatedEmployee) {
32         return repository.findById(id).map(employee -> {
33             employee.setName(updatedEmployee.getName());
34             employee.setDepartment(updatedEmployee.getDepartment());
35             return repository.save(employee);
36         }).orElse(null);
37     }
38
39 }
40

```

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);
}

```

The screenshot shows a REST client interface with the following details:

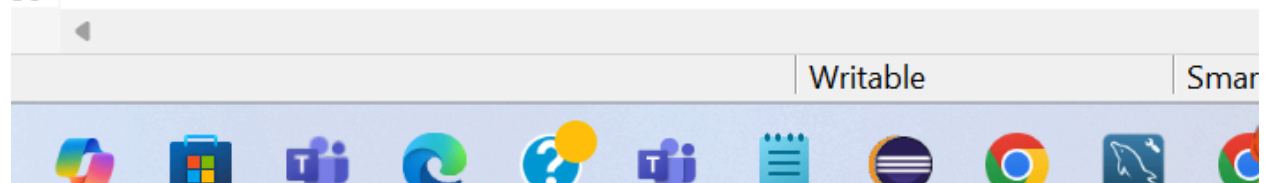
- Method:** DELETE
- URL:** localhost:9090/employee/1
- Body:** A JSON object: `{ "name": "kalyan", "department": "ID" }`
- Response:** 200 OK, 68 ms, 123 B
- Headers:** 4 headers are listed.
- Test Results:** A section for test results.
- Raw View:** The response body is shown in raw format as a single line.

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

```

55
56 //delete by id
57 @DeleteMapping("/{id}")
58 public String deleteEmployeeById(@PathVariable Long id) {
59     return service.deleteEmployeeById(id);
60 }
61
62
63
64
65 }
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



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

```

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

```

Body
Cookies
Headers (5)
Test Results

200 OK
415 ms
202 B

Raw
Preview
Visualize

1 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)

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

Employee.java EmployeeController.java EmployeeService.java Employee

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

Writable Smart

Employee.java EmployeeController.java EmployeeService.java Employee

```
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8 import com.wipro.entity.Employee;
9 import com.wipro.repository.EmployeeRepository;
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
```

Writable

Smart

```

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

```

```

1 # Application Name
2 spring.application.name=springdatajpademo
3
4 # Server Port
5 server.port=9090
6
7 # Database Configuration
8 spring.datasource.url=jdbc:mysql://localhost:3306/wipro
9 spring.datasource.username=root
10 spring.datasource.password=#Mahadev7
11
12 # Hibernate Properties
13 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
14 spring.jpa.hibernate.ddl-auto=update
15 spring.jpa.show-sql=true
16 spring.jpa.format-sql=true
17

```

```

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

```

GET http://localhost:9090/employee/name/kalyan

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

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

1 {

Body Cookies Headers (5) Test Results 200 OK • 59 ms • 209 B •

{ } JSON Preview Visualize

```
1 [
2   {
3     "id": 52,
4     "name": "kalyan",
5     "department": "IT"
6   }
7 ]
```

GET http://localhost:9090/employee/department/IT

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

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

1 {

Body Cookies Headers (5) Test Results 200 OK • 8 ms • 207 B •

{ } JSON Preview Visualize

```
1 [
2   {
3     "id": 53,
4     "name": "prem",
5     "department": "IT"
6   }
7 ]
```

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 EmployeeRepository... EmployeeService... Em
1 spring.application.name=springdatapagination
2 server.port=9090
3
4 # MySQL Configuration
5 spring.datasource.url=jdbc:mysql://localhost:3306/wipro
6 spring.datasource.username=root
7 spring.datasource.password=#Mahadev7
8
9 # Hibernate Configuration
10 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
11 spring.jpa.hibernate.ddl-auto=update
12 spring.jpa.show-sql=true
13
```

```
1 package com.wipro.entity;
2 import jakarta.persistence.*;
3
4 @Entity
5 public class Employee {
6     @Id
7     @GeneratedValue(strategy = GenerationType.IDENTITY)
8     private Long id;
9     private String name;
10    private String department;
11
12    public Employee() {}
13
14    public Employee(String name, String department) {
15        this.name = name;
16        this.department = department;
17    }
18
19    public Long getId() { return id; }
20    public String getName() { return name; }
21    public void setName(String name) { this.name = name; }
22    public String getDepartment() { return department; }
23    public void setDepartment(String department) { this.department = department; }
24 }
25
```

```

1 package com.wipro.repository;
2 import org.springframework.data.domain.Page;
3 import org.springframework.data.domain.Pageable;
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     Page<Employee> findAll(Pageable pageable); // Pagination support
11 }
12

```

```

1 package com.wipro.service;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.data.domain.*;
4 import org.springframework.stereotype.Service;
5 import com.wipro.entity.Employee;
6 import com.wipro.repository.EmployeeRepository;
7 import java.util.List;
8
9 @Service
10 public class EmployeeService {
11
12     @Autowired
13     private EmployeeRepository repository;
14
15     // Pagination
16     public Page<Employee> getEmployeesWithPagination(int page, int size) {
17         Pageable pageable = PageRequest.of(page, size);
18         return repository.findAll(pageable);
19     }
20
21     // Sorting
22     public List<Employee> getEmployeesWithSorting(String field) {
23         return repository.findAll(Sort.by(Sort.Direction.ASC, field)); // Sorting in ascending order
24     }
25
26     // Pagination + Sorting
27     public Page<Employee> getEmployeesWithPaginationAndSorting(int page, int size, String field) {
28         Pageable pageable = PageRequest.of(page, size, Sort.by(Sort.Direction.ASC, field));
29         return repository.findAll(pageable);
30     }
31 }
32

```

```
application.pro... Employee.java EmployeeRepository EmployeeService... EmployeeControll... x Springdatapagin...
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.*;
5 import com.wipro.entity.Employee;
6 import com.wipro.service.EmployeeService;
7 import java.util.List;
8
9 @RestController
10 @RequestMapping("/employee")
11 public class EmployeeController {
12
13     @Autowired
14     private EmployeeService service;
15
16     // Pagination API
17     @GetMapping("/pagination/{page}/{size}")
18     public Page<Employee> getEmployeesWithPagination(@PathVariable int page, @PathVariable int size) {
19         return service.getEmployeesWithPagination(page, size);
20     }
21
22     // Sorting API
23     @GetMapping("/sort/{field}")
24     public List<Employee> getEmployeesWithSorting(@PathVariable String field) {
25         return service.getEmployeesWithSorting(field);
26     }
27
28     // Pagination & Sorting Combined
29     @GetMapping("/pagination/{page}/{size}/sort/{field}")
30     public Page<Employee> getEmployeesWithPaginationAndSorting(@PathVariable int page,
31                                                                 @PathVariable int size,
32                                                                 @PathVariable String field) {
33         return service.getEmployeesWithPaginationAndSorting(page, size, field);
34     }
35 }
36
```

Writable Smart Insert 1 : 30 : 29

```
application.pro... Employee.java EmployeeRepository EmployeeService... EmployeeControl...
1 |
2 package com.wipro;
3
4 import org.springframework.boot.SpringApplication;
5
6
7 @SpringBootApplication
8 public class SpringdatapaginationApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(SpringdatapaginationApplication.class, args);
12     }
13
14 }
15
```

Get the employees with the pagination

GET ⌵ 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

GET

localhost:9090/employee/sort/name

Send

Params

Authorization

Headers (6)

Body

Scripts

Settings

Cookies

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body

Cookies

Headers (5)

Test Results

200 OK • 16 ms • 251 B •

JSON

Preview

Visualize

```
1  [
2    {
3      "id": 52,
4      "name": "kalyan",
5      "department": "ID"
6    },
7    {
8      "id": 53,
9      "name": "prem",
10     "department": "IT"
11   }
12 ]
```