











**1. Create and Setup Spring Boot Project in Eclipse STS**

### 1. Right-click in the package explorer and select New -> Spring Starter Project

### 2. Specify project details

* **Generate: Maven Project**
* **Language: Java**
* **Java Version: 1.17 (Default)**
* Spring Boot: **3.1.2**
* Group: **com.ccteam**
* Artifact: **employee-crud-app**
* Name:
* Description: Demo project for Spring Boot
* Package Name : com.ccteam
* Packaging: jar (This is the default value)
* Dependencies: **Web, JPA, MySQL, Dev Tools ,Thyme leaf**

Once you click Finish, Maven would take some time to download all the dependencies and initialize the project.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create some packages in the project.

**[controller]**

**[service]**

**[repository]**

**[model]**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**2. Database Setup**

Create a database with the name "**employeedb**" in the MySQL database server.

Open the MySQL Workbench and Execute the following 2 commands.

Ex:

|  |
| --- |
| create database employeedb;  use employeedb; |

We’ll need to configure MySQL database **URL**, **username**, and **password**so that Spring can establish a connection with the database on startup.

Open *application.properties* and add following MySQL database configuration:

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url=jdbc:mysql://localhost:3306/employeedb?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false

spring.datasource.username=root

spring.datasource.password=mysql

# Hibernate

# The SQL dialect makes Hibernate generate better SQL for the chosen database

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto = update

logging.level.org.hibernate.SQL=DEBUG

logging.level.org.hibernate.type=TRACE

The *spring.jpa.hibernate.ddl-auto = update* property makes sure that the database tables and the domain models in your application are in sync. Whenever you change the domain model, hibernate will automatically update the mapped table in the database when you restart the application.

I have also specified the log levels for hibernate so that we can debug the SQL queries executed by hibernate

# 3. Create JPA Entity - Employee.java

|  |
| --- |
| package com.ccteam.employeecrudapp.model;;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.Table;  @Entity  @Table(name = "employees")  public class Employee {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private long id;  @Column(name = "first\_name")  private String firstName;  @Column(name = "last\_name")  private String lastName;  @Column(name = "email")  private String email;  public long getId() {  return id;  }  public void setId(long id) {  this.id = id;  }  public String getFirstName() {  return firstName;  }  public void setFirstName(String firstName) {  this.firstName = firstName;  }  public String getLastName() {  return lastName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  } |

We will implement the end-to-end "List Employee Feature" in our Employee Management System project.

# 1. Back-end changes

Create a *EmployeeRepository* interface under "com.ccteam.repository" package and add the following content to it:

|  |
| --- |
| package com.ccteam.employeecrudapp.repository;  import com.ccteam.employeecrudapp.model.Employee; import org.springframework.data.jpa.repository.JpaRepository;  public interface EmployeeRepository extends JpaRepository<Employee, Long> {  } |

This will provide CRUD database operations for the *Employee* entity.

## **[EmployeeService.java]**

Create an*EmployeeService* interface under "com.ccteam.service" and add the following content to it.

|  |
| --- |
| import java.util.List;  import com.ccteam.model.Employee;  public interface EmployeeService {  List<Employee> **getAllEmployees();**  } |

## **[EmployeeServiceImpl.java]**

Create an *EmployeeServiceImpl* class under "**com.ccteam.service**" and add the following content to it:

|  |
| --- |
| import java.util.List;  import java.util.Optional;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  @Service  public class EmployeeServiceImpl implements EmployeeService {  @Autowired  private EmployeeRepository employeeRepository;  @Override  public List < Employee > getAllEmployees() {  return employeeRepository.**findAll();**  }  } |

## **[EmployeeController.java]**

Create an *EmployeeServiceImpl* class under "com.ccteam.controller" and add the following content to it:

|  |
| --- |
| @Controller  public class EmployeeController {  @Autowired  private EmployeeService employeeService;  // display list of employees  @GetMapping("/")  public String viewHomePage(Model model) {  model.addAttribute("listEmployees", employeeService.getAllEmployees());  return "index";  }  } |

# 2. Front-end changes

## **[index.html]**

Create an *index.html* Thymeleaf template under the "resources/templates" folder and add the following content to it:

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="ISO-8859-1">  <title>Employee Management System</title>  <link rel="stylesheet"  href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"  integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"  crossorigin="anonymous">    </head>  <body>  <div class="container my-2">  <h1>Employees List</h1>  <table border="1" class = "table table-striped table-responsive-md">  <thead>  <tr>  <th>Employee First Name</th>  <th>Employee Last Name</th>  <th>Employee Email</th>  </tr>  </thead>  <tbody>  <tr th:each="employee : ${listEmployees}">  <td th:text="${employee.firstName}"></td>  <td th:text="${employee.lastName}"></td>  <td th:text="${employee.email}"></td>  </tr>  </tbody>  </table>  </div>  </body>  </html> |

# 3. Run Spring application and demo

After that insert some records in table and Re Run the Project.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

We will implement end-to-end to "Add Employee Feature" in our **Employee Management System** project.

First, we will complete the back-end changes.

**[EmployeeService.java]**

Add "saveEmployee()" method to the *EmployeeService* interface.

The complete code:

public interface EmployeeService {

List < Employee > getAllEmployees();

void saveEmployee(Employee employee);

}

## EmployeeServiceImpl.java

Let's implement "saveEmployee()" method *EmployeeServiceImpl* class. Here is the complete code

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List < Employee > getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public void saveEmployee(Employee employee) {

this.employeeRepository.save(employee);

}

}

## index.html changes

Add below button to index.html:

<a th:href = "@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

The complete code:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container my-2">

<h1>Employees List</h1>

<a th:href="@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

<table border="1" class="table table-striped table-responsive-md">

<thead>

<tr>

<th>Employee First Name</th>

<th>Employee Last Name</th>

<th>Employee Email</th>

<th> Actions </th>

</tr>

</thead>

<tbody>

<tr th:each="employee : ${listEmployees}">

<td th:text="${employee.firstName}"></td>

<td th:text="${employee.lastName}"></td>

<td th:text="${employee.email}"></td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

## EmployeeController changes

Add below method handler in *EmployeeController* class:

@GetMapping("/showNewEmployeeForm")

public String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = new Employee();

model.addAttribute("employee", employee);

return "new\_employee";

}

## Create new\_employee.html

Create new *new\_employee.html* file under "resources/templates" folder and add the following content to it:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container">

<h1>Employee Management System</h1>

<hr>

<h2>Save Employee</h2>

<form action="#" th:action="@{/saveEmployee}" th:object="${employee}" method="POST">

<input type="text" th:field="\*{firstName}" placeholder="Employee First Name" class="form-control mb-4 col-4">

<input type="text" th:field="\*{lastName}" placeholder="Employee Last Name" class="form-control mb-4 col-4">

<input type="text" th:field="\*{email}" placeholder="Employee Email" class="form-control mb-4 col-4">

<button type="submit" class="btn btn-info col-2"> Save Employee</button>

</form>

<hr>

<a th:href="@{/}"> Back to Employee List</a>

</div>

</body>

</html>

## EmployeeController changes

Add below method handler to *EmployeeController* class:

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

return "redirect:/";

}

The complete *EmployeeController* class code:

@Controller

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

model.addAttribute("listEmployees", employeeService.getAllEmployees());

return "index";

}

@GetMapping("/showNewEmployeeForm")

public String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = new Employee();

model.addAttribute("employee", employee);

return "new\_employee";

}

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

return "redirect:/";

}

}

## Run the Spring application and Demo

**Update Employee:**

# 1. Back-end changes

## EmployeeService.java interface change

Add below method in *EmployeeSerivce* interface:

Employee getEmployeeById(long id);

The complete code:

public interface EmployeeService {

List < Employee > getAllEmployees();

void saveEmployee(Employee employee);

Employee getEmployeeById(long id);

}

## EmployeeServiceImpl.java class change

Override and implement "getEmployeeById()" method in *EmployeeServiceImpl* class like:

@Override

public Employee getEmployeeById(long id) {

Optional < Employee > optional = employeeRepository.findById(id);

Employee employee = null;

if (optional.isPresent()) {

employee = optional.get();

} else {

throw new RuntimeException(" Employee not found for id :: " + id);

}

return employee;

}

The complete code:

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List < Employee > getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public void saveEmployee(Employee employee) {

this.employeeRepository.save(employee);

}

@Override

public Employee getEmployeeById(long id) {

Optional < Employee > optional = employeeRepository.findById(id);

Employee employee = null;

if (optional.isPresent()) {

employee = optional.get();

} else {

throw new RuntimeException(" Employee not found for id :: " + id);

}

return employee;

}

}

## EmployeeController changes

Add following method in *EmployeeController* class:

@GetMapping("/showFormForUpdate/{id}")

public String showFormForUpdate(@PathVariable ( value = "id") long id, Model model) {

// get employee from the service

Employee employee = employeeService.getEmployeeById(id);

// set employee as a model attribute to pre-populate the form

model.addAttribute("employee", employee);

return "update\_employee";

}

The complete code:

@Controller

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

model.addAttribute("listEmployees", employeeService.getAllEmployees());

return "index";

}

@GetMapping("/showNewEmployeeForm")

public String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = new Employee();

model.addAttribute("employee", employee);

return "new\_employee";

}

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

return "redirect:/";

}

@GetMapping("/showFormForUpdate/{id}")

public String showFormForUpdate(@PathVariable(value = "id") long id, Model model) {

// get employee from the service

Employee employee = employeeService.getEmployeeById(id);

// set employee as a model attribute to pre-populate the form

model.addAttribute("employee", employee);

return "update\_employee";

}

}

**2. Front-end changes**

Add below button in actions column:

<td> <a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">Update</a></td>

The complete code:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container my-2">

<h1>Employees List</h1>

<a th:href="@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

<table border="1" class="table table-striped table-responsive-md">

<thead>

<tr>

<th>Employee First Name</th>

<th>Employee Last Name</th>

<th>Employee Email</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<tr th:each="employee : ${listEmployees}">

<td th:text="${employee.firstName}"></td>

<td th:text="${employee.lastName}"></td>

<td th:text="${employee.email}"></td>

<td><a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">Update</a></td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

## Create update\_employee.html

Create *update\_employee.html* file under "resources/templates" folder and add the following content to it:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container">

<h1>Employee Management System</h1>

<hr>

<h2>Update Employee</h2>

<form action="#" th:action="@{/saveEmployee}" th:object="${employee}" method="POST">

<!-- Add hidden form field to handle update -->

<input type="hidden" th:field="\*{id}" />

<input type="text" th:field="\*{firstName}" class="form-control mb-4 col-4">

<input type="text" th:field="\*{lastName}" class="form-control mb-4 col-4">

<input type="text" th:field="\*{email}" class="form-control mb-4 col-4">

<button type="submit" class="btn btn-info col-2"> Update Employee</button>

</form>

<hr>

<a th:href="@{/}"> Back to Employee List</a>

</div>

</body>

</html>

# 3. Run Spring application and demo

Implement Delete Employee Feature

# 1. Back-end changes

## EmployeeService.java interface change

Add below method in *EmployeeService* interface:

void deleteEmployeeById(long id);

The complete code:

public interface EmployeeService {

List < Employee > getAllEmployees();

void saveEmployee(Employee employee);

Employee getEmployeeById(long id);

void deleteEmployeeById(long id);

}

## EmployeeServiceImpl class changes

Add below method in *EmployeeServiceImpl* class:

@Override

public void deleteEmployeeById(long id) {

this.employeeRepository.deleteById(id);

}

The complete code:

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List < Employee > getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public void saveEmployee(Employee employee) {

this.employeeRepository.save(employee);

}

@Override

public Employee getEmployeeById(long id) {

Optional < Employee > optional = employeeRepository.findById(id);

Employee employee = null;

if (optional.isPresent()) {

employee = optional.get();

} else {

throw new RuntimeException(" Employee not found for id :: " + id);

}

return employee;

}

@Override

public void deleteEmployeeById(long id) {

this.employeeRepository.deleteById(id);

}

}

## EmployeeController changes

Add below method to *EmployeeController* class:

@GetMapping("/deleteEmployee/{id}")

public String deleteEmployee(@PathVariable (value = "id") long id) {

// call delete employee method

this.employeeService.deleteEmployeeById(id);

return "redirect:/";

}

The complete code:

@Controller

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

model.addAttribute("listEmployees", employeeService.getAllEmployees());

return "index";

}

@GetMapping("/showNewEmployeeForm")

public String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = new Employee();

model.addAttribute("employee", employee);

return "new\_employee";

}

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

return "redirect:/";

}

@GetMapping("/showFormForUpdate/{id}")

public String showFormForUpdate(@PathVariable(value = "id") long id, Model model) {

// get employee from the service

Employee employee = employeeService.getEmployeeById(id);

// set employee as a model attribute to pre-populate the form

model.addAttribute("employee", employee);

return "update\_employee";

}

@GetMapping("/deleteEmployee/{id}")

public String deleteEmployee(@PathVariable(value = "id") long id) {

// call delete employee method

this.employeeService.deleteEmployeeById(id);

return "redirect:/";

}

}

**2. Front-end changes**

Add "Delete" button in index.html file:

<a th:href="@{/deleteEmployee/{id}(id=${employee.id})}" class="btn btn-danger">

Delete

</a>

The complete code:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container my-2">

<h1>Employees List</h1>

<a th:href="@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

<table border="1" class="table table-striped table-responsive-md">

<thead>

<tr>

<th>Employee First Name</th>

<th>Employee Last Name</th>

<th>Employee Email</th>

<th> Actions </th>

</tr>

</thead>

<tbody>

<tr th:each="employee : ${listEmployees}">

<td th:text="${employee.firstName}"></td>

<td th:text="${employee.lastName}"></td>

<td th:text="${employee.email}"></td>

<td> <a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">Update</a>

<a th:href="@{/deleteEmployee/{id}(id=${employee.id})}" class="btn btn-danger">Delete</a>

</td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

# 3. Run Spring application and demo

+++++++++++++++++++++++++++++++++++++++++++++++++++++

We will implement **Pagination** for our employee list page using Spring Data JPA and Thymeleaf.

To use paging and sorting APIs provided by Spring Data JPA, your repository interface must extend the *PagingAndSortingRepository* interface which defines the following couple of methods (T refers to an entity class):

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# Understand Spring Data JPA’s Pagination APIs

To use paging and sorting APIs provided by Spring Data JPA, your repository interface must extend the *PagingAndSortingRepository* interface which defines the following couple of methods (T refers to an entity class):

*JpaRepository* interface extends *PagingAndSortingRepository* interface so if your repository interface is of type *JpaRepository*, you don’t have to make a change to it.

For example, use the following to get the first page from the database, with 5 items per page:

int pageNumber = 1;

int pageSize = 5;

Pageable pageable = PageRequest.of(pageNumber, pageSize);

Page<Product> page = repository.findAll(pageable);

Then you can get the actual content as follows:

List<Employee> listEmployees = page.getContent();

With a Page object you can know the total rows in the database and the total pages according to the given page size:

long totalItems = page.getTotalElements();

int totalPages = page.getTotalPages();

This information is useful for implementing pagination in the view with the Thymeleaf template.

# 1. Back-end changes

## EmployeeService.java interface changes

Add below method to this interface:

Page<Employee> findPaginated(int pageNo, int pageSize);

The complete code:

public interface EmployeeService {

List < Employee > getAllEmployees();

void saveEmployee(Employee employee);

Employee getEmployeeById(long id);

void deleteEmployeeById(long id);

Page < Employee > findPaginated(int pageNo, int pageSize);

}

## EmployeeServiceImpl.java class changes

Add below method to *EmployeeServiceImpl* class:

@Override

public Page<Employee> findPaginated(int pageNo, int pageSize) {

Pageable pageable = PageRequest.of(pageNo - 1, pageSize);

return this.employeeRepository.findAll(pageable);

}

## EmployeeController.java class changes

Add below handler method to *EmployeeController* class to perform pagination:

@GetMapping("/page/{pageNo}")

public String findPaginated(@PathVariable(value = "pageNo") int pageNo, Model model) {

int pageSize = 5;

Page < Employee > page = employeeService.findPaginated(pageNo, pageSize);

List < Employee > listEmployees = page.getContent();

model.addAttribute("currentPage", pageNo);

model.addAttribute("totalPages", page.getTotalPages());

model.addAttribute("totalItems", page.getTotalElements());

model.addAttribute("listEmployees", listEmployees);

return "index";

}

Also, we need to make a change to an existing method as below:

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

return findPaginated(1, model);

}

# 2. Front-end changes

Add below pagination code to *index.html*:

<div th:if="${totalPages > 1}">

<div class="row col-sm-10">

<div class="col-sm-2">

Total Rows: [[${totalItems}]]

</div>

<div class="col-sm-1">

<span th:each="i: ${#numbers.sequence(1, totalPages)}">

<a th:if="${currentPage != i}" th:href="@{'/page/' + ${i}}">[[${i}]]</a>

<span th:unless="${currentPage != i}">[[${i}]]</span> &nbsp; &nbsp;

</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${currentPage + 1}}">Next</a>

<span th:unless="${currentPage < totalPages}">Next</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${totalPages}}">Last</a>

<span th:unless="${currentPage < totalPages}">Last</span>

</div>

</div>

</div>

The complete code:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<div class="container my-2">

<h1>Employees List</h1>

<a th:href="@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

<table border="1" class="table table-striped table-responsive-md">

<thead>

<tr>

<th>Employee First Name</th>

<th>Employee Last Name</th>

<th>Employee Email</th>

<th> Actions </th>

</tr>

</thead>

<tbody>

<tr th:each="employee : ${listEmployees}">

<td th:text="${employee.firstName}"></td>

<td th:text="${employee.lastName}"></td>

<td th:text="${employee.email}"></td>

<td> <a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">Update</a>

<a th:href="@{/deleteEmployee/{id}(id=${employee.id})}" class="btn btn-danger">Delete</a>

</td>

</tr>

</tbody>

</table>

<div th:if="${totalPages > 1}">

<div class="row col-sm-10">

<div class="col-sm-2">

Total Rows: [[${totalItems}]]

</div>

<div class="col-sm-1">

<span th:each="i: ${#numbers.sequence(1, totalPages)}">

<a th:if="${currentPage != i}" th:href="@{'/page/' + ${i}}">[[${i}]]</a>

<span th:unless="${currentPage != i}">[[${i}]]</span> &nbsp; &nbsp;

</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${currentPage + 1}}">Next</a>

<span th:unless="${currentPage < totalPages}">Next</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${totalPages}}">Last</a>

<span th:unless="${currentPage < totalPages}">Last</span>

</div>

</div>

</div>

</div>

</body>

</html>

# 3. Run Spring boot application and demo

++++++++++++++++++++++++++++++++++++++++++++++++++++

we will implement **Sorting**for our employee list page using Spring Data JPA and Thymeleaf.

# Understand Spring Data JPA’s Sorting API

The users will be able to sort the employee's list by clicking on the column header of the table.

First, create a **Sort**object like this:

Sort sort = Sort.by(“fieldName”).ascending();

This will sort the result by **fieldName**in ascending order. **fieldName**must match a field name declared in the entity class.

We can also sort by more than one field, for example:

Sort sort = Sort.by("fieldName1").ascending().and(Sort.by("fieldName2").descending());

Then we pass the **Sort**object to create a **Pageable**as follows:

Pageable pageable = PageRequest.of(pageNum - 1, pageSize, sort);

# 1. Back-end changes

## 1. EmployeeService.java interface changes

Let's add two fields to the existing method:

Page<Employee> findPaginated(int pageNo, int pageSize, String sortField, String sortDirection);

The complete code:

public interface EmployeeService {

List < Employee > getAllEmployees();

void saveEmployee(Employee employee);

Employee getEmployeeById(long id);

void deleteEmployeeById(long id);

Page < Employee > findPaginated(int pageNo, int pageSize, String sortField, String sortDirection);

}

## 2. EmployeeServiceImpl.java class changes

The sorting logic implemented in the below method:

@Override

public Page<Employee> findPaginated(int pageNo, int pageSize, String sortField, String sortDirection) {

Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() :

Sort.by(sortField).descending();

Pageable pageable = PageRequest.of(pageNo - 1, pageSize, sort);

return this.employeeRepository.findAll(pageable);

}

The complete code:

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List < Employee > getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public void saveEmployee(Employee employee) {

this.employeeRepository.save(employee);

}

@Override

public Employee getEmployeeById(long id) {

Optional < Employee > optional = employeeRepository.findById(id);

Employee employee = null;

if (optional.isPresent()) {

employee = optional.get();

} else {

throw new RuntimeException(" Employee not found for id :: " + id);

}

return employee;

}

@Override

public void deleteEmployeeById(long id) {

this.employeeRepository.deleteById(id);

}

@Override

public Page < Employee > findPaginated(int pageNo, int pageSize, String sortField, String sortDirection) {

Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() :

Sort.by(sortField).descending();

Pageable pageable = PageRequest.of(pageNo - 1, pageSize, sort);

return this.employeeRepository.findAll(pageable);

}

}

## 3.EmployeeController.java changes

Let's change the existing method to provide a support for sorting:

@GetMapping("/page/{pageNo}")

public String findPaginated(@PathVariable(value = "pageNo") int pageNo,

@RequestParam("sortField") String sortField,

@RequestParam("sortDir") String sortDir,

Model model) {

int pageSize = 5;

Page < Employee > page = employeeService.findPaginated(pageNo, pageSize, sortField, sortDir);

List < Employee > listEmployees = page.getContent();

model.addAttribute("currentPage", pageNo);

model.addAttribute("totalPages", page.getTotalPages());

model.addAttribute("totalItems", page.getTotalElements());

model.addAttribute("sortField", sortField);

model.addAttribute("sortDir", sortDir);

model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" : "asc");

model.addAttribute("listEmployees", listEmployees);

return "index";

}

Also provide default sorting field and sorting direction for home page:

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

return findPaginated(1, "firstName", "asc", model);

}

The complete code:

@Controller

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// display list of employees

@GetMapping("/")

public String viewHomePage(Model model) {

return findPaginated(1, "firstName", "asc", model);

}

@GetMapping("/showNewEmployeeForm")

public String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = new Employee();

model.addAttribute("employee", employee);

return "new\_employee";

}

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

return "redirect:/";

}

@GetMapping("/showFormForUpdate/{id}")

public String showFormForUpdate(@PathVariable(value = "id") long id, Model model) {

// get employee from the service

Employee employee = employeeService.getEmployeeById(id);

// set employee as a model attribute to pre-populate the form

model.addAttribute("employee", employee);

return "update\_employee";

}

@GetMapping("/deleteEmployee/{id}")

public String deleteEmployee(@PathVariable(value = "id") long id) {

// call delete employee method

this.employeeService.deleteEmployeeById(id);

return "redirect:/";

}

@GetMapping("/page/{pageNo}")

public String findPaginated(@PathVariable(value = "pageNo") int pageNo,

@RequestParam("sortField") String sortField,

@RequestParam("sortDir") String sortDir,

Model model) {

int pageSize = 5;

Page < Employee > page = employeeService.findPaginated(pageNo, pageSize, sortField, sortDir);

List < Employee > listEmployees = page.getContent();

model.addAttribute("currentPage", pageNo);

model.addAttribute("totalPages", page.getTotalPages());

model.addAttribute("totalItems", page.getTotalElements());

model.addAttribute("sortField", sortField);

model.addAttribute("sortDir", sortDir);

model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" : "asc");

model.addAttribute("listEmployees", listEmployees);

return "index";

}

}

# 2. Front-end changes

## 1.index.html

We make the header columns of the table sortable by adding hyperlinks with the following code:

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=firstName&sortDir=' + ${reverseSortDir}}">

Employee First Name</a>

</th>

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=lastName&sortDir=' + ${reverseSortDir}}">

Employee Last Name</a>

</th>

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=email&sortDir=' + ${reverseSortDir}}">

Employee Email</a>

</th>

<th> Actions </th>

We also need to change pagination part to provide sorting support like:

<div th:if = "${totalPages > 1}">

<div class = "row col-sm-10">

<div class = "col-sm-2">

Total Rows: [[${totalItems}]]

</div>

<div class = "col-sm-1">

<span th:each="i: ${#numbers.sequence(1, totalPages)}">

<a th:if="${currentPage != i}" th:href="@{'/page/' + ${i}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">[[${i}]]</a>

<span th:unless="${currentPage != i}">[[${i}]]</span> &nbsp; &nbsp;

</span>

</div>

<div class = "col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${currentPage + 1}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">Next</a>

<span th:unless="${currentPage < totalPages}">Next</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${totalPages}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">Last</a>

<span th:unless="${currentPage < totalPages}">Last</span>

</div>

</div>

</div>

The complete code:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Employee Management System</title>

<link rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"

integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"

crossorigin="anonymous">

</head>

<body>

<div class="container my-2">

<h1>Employees List</h1>

<a th:href = "@{/showNewEmployeeForm}" class="btn btn-primary btn-sm mb-3"> Add Employee </a>

<table border="1" class = "table table-striped table-responsive-md">

<thead>

<tr>

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=firstName&sortDir=' + ${reverseSortDir}}">

Employee First Name</a>

</th>

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=lastName&sortDir=' + ${reverseSortDir}}">

Employee Last Name</a>

</th>

<th>

<a th:href="@{'/page/' + ${currentPage} + '?sortField=email&sortDir=' + ${reverseSortDir}}">

Employee Email</a>

</th>

<th> Actions </th>

</tr>

</thead>

<tbody>

<tr th:each="employee : ${listEmployees}">

<td th:text="${employee.firstName}"></td>

<td th:text="${employee.lastName}"></td>

<td th:text="${employee.email}"></td>

<td> <a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">Update</a>

<a th:href="@{/deleteEmployee/{id}(id=${employee.id})}" class="btn btn-danger">Delete</a>

</td>

</tr>

</tbody>

</table>

<div th:if = "${totalPages > 1}">

<div class = "row col-sm-10">

<div class = "col-sm-2">

Total Rows: [[${totalItems}]]

</div>

<div class = "col-sm-1">

<span th:each="i: ${#numbers.sequence(1, totalPages)}">

<a th:if="${currentPage != i}" th:href="@{'/page/' + ${i}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">[[${i}]]</a>

<span th:unless="${currentPage != i}">[[${i}]]</span> &nbsp; &nbsp;

</span>

</div>

<div class = "col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${currentPage + 1}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">Next</a>

<span th:unless="${currentPage < totalPages}">Next</span>

</div>

<div class="col-sm-1">

<a th:if="${currentPage < totalPages}" th:href="@{'/page/' + ${totalPages}+ '?sortField=' + ${sortField} + '&sortDir=' + ${sortDir}}">Last</a>

<span th:unless="${currentPage < totalPages}">Last</span>

</div>

</div>

</div>

</div>

</body>

</html>

# 3. Run Spring boot application and demo

https://www.javaguides.net/2020/05/spring-boot-thymeleaf-crud-database-real-time-project-part7.html