Spring MVC concept and Pratical(CRUD operations)

Presented by Group 1

Table of Content

Part I: Spring Boot Introduction

Spring in a Nutshell

- Very popular framework for building Java applications
- Provides a large number of helper classes and annotations.

Spring Boot and Spring

- Spring Boot uses Spring behind the scenes
- Spring Boot simply makes it easier to use Spring

Cont'd...

Spring Boot Solution

- Make it easier to get started with Spring development
- Minimize the amount of manual configuration
- Perform auto-configuration based on props files and JAR class-path
- Help to resolve dependency conflicts (Maven or Gradle)
- Provide an embedded HTTP server so you can get started quickly
- Tomcat, Jetty, Undertow, ...

Quick Word on Maven

- When building your Java project, you may need additional JAR files
- For example: Spring, Hibernate, Commons Logging, JSON etc...
- One approach is to download the JAR files from each project web site
- Manually add the JAR files to your build path / classpath

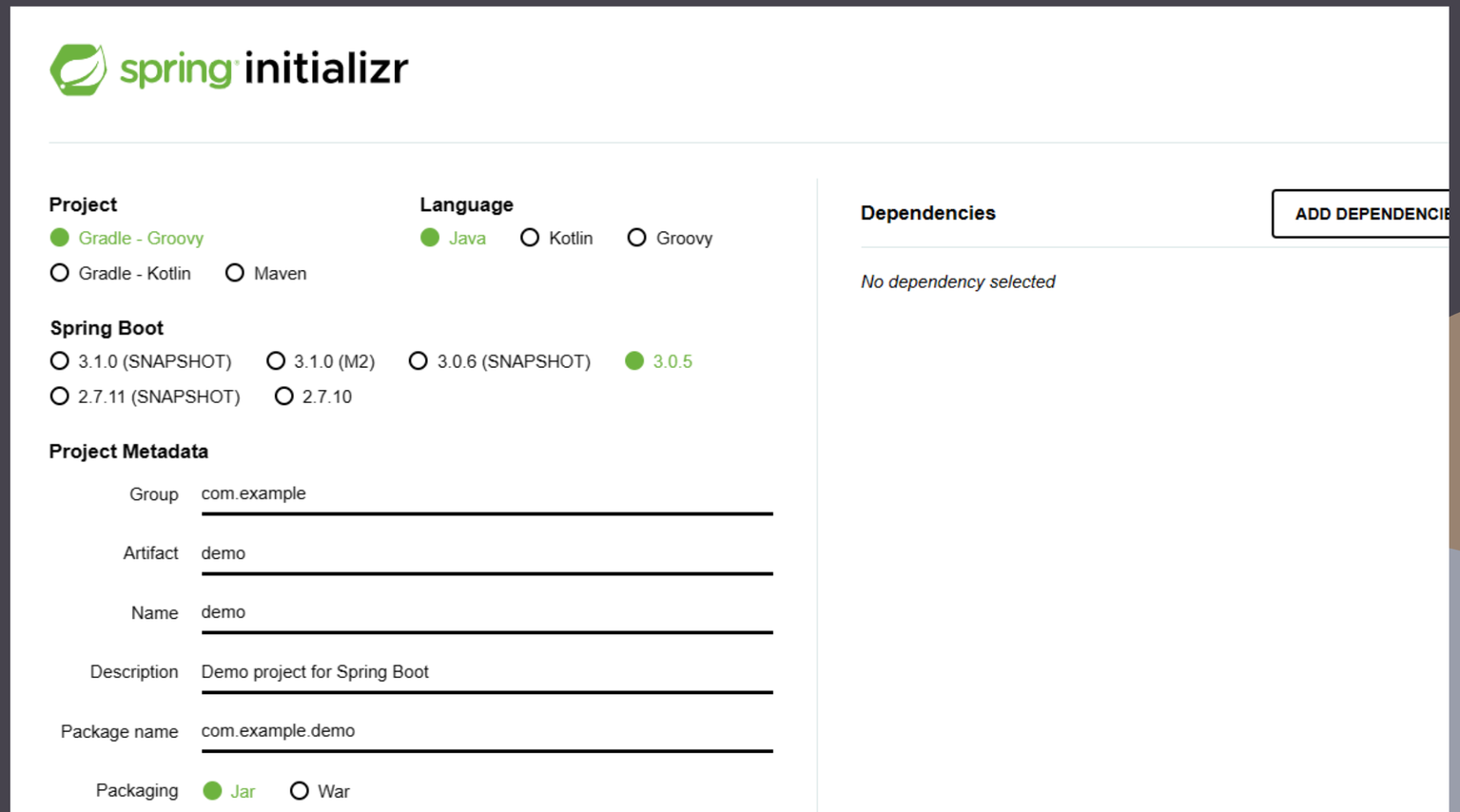
Cont'd...

Maven Solution

- Tell Maven the projects you are working with (dependencies) Spring, Hibernate etc ....
- Maven will go out and download the JAR files for those projects for you
- And Maven will make those JAR files available during compile/run
- Think of Maven as your friendly helper / personal shopper.

Spring Initializr

- Quickly create a starter Spring project
- Select your dependencies
- Creates a Maven/Gradle project
- Import the project into your IDE
- Eclipse, IntelliJ, NetBeans etc ...



PART II:

Spring MVC Concept

What is Spring MVC

Spring MVC is a popular web framework that is used to build web applications in Java. It is a part of the Spring Framework and follows the Model-View-Controller (MVC) design pattern.

The MVC design pattern separates an application into three components:

1. Model: Represents the data and the business logic of the application.
2. View: Displays the data to the user and handles user input.
3. Controller: Receives requests from the user, interacts with the model to process data, and

cont'd...

In Spring MVC, the framework provides components for each of these three parts of the application. The model is represented by Java objects or beans, which can be managed by Spring's inversion of control (IoC) container. The view is typically implemented using JavaServer Pages (JSP) or Thymeleaf, and the controller is implemented as a Java class that handles HTTP requests.

The core components of Spring MVC include the DispatcherServlet, which receives requests from the user and delegates them to the appropriate controller, and the HandlerMapping, which maps incoming requests to specific controllers.

Spring MVC also provides support for validation, data binding, and internationalization, making it a powerful and flexible web framework for building modern web applications.

PART III:

Spring MVC pratical(CRUD operations)

SPRING MVC CRUD PROJECT

Employee Management

- Create an employee instance.
- Read all employees in the database
- Update a employee instance.
- Delete a employees instance

Implemented using SpringBoot and ThymLeaf.

SpringMvc23139Application

EmployeeServiceImpl.java | application.properties | SpringMvc23139Application.java | pom.

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.10</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.mhwc</groupId>
<artifactId>springMVC23139</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springMVC23139</name>
<description>CRUD operation with Spring</description>
<properties>
    <java.version>1.8</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
```

roject › dependencies › dependency › artifactId

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

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
dependencies>
```

```java
package com.mhwc.springMVC23139.model;
import javax.persistence.*;
19 usages
@Entity
@Table(name="employee")
public class Employee {
    // define fields
    4 usages
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name="id")
    private int id;
    5 usages
    @Column(name="first_name")
    private String firstName;
    5 usages
    @Column(name="last_name")
    private String lastName;
    5 usages
    @Column(name="email")
    private String email;
    // define getter and setters
    //define constructors
    //define no args class
```

ID Generation Strategies

Name	Description
GenerationType.AUTO	Pick an appropriate strategy for the particular database
GenerationType.IDENTITY	Assign primary keys using database identity column
GenerationType.SEQUENCE	Assign primary keys using a database sequence
GenerationType.TABLE	Assign primary keys using an underlying database table to ensure uniqueness

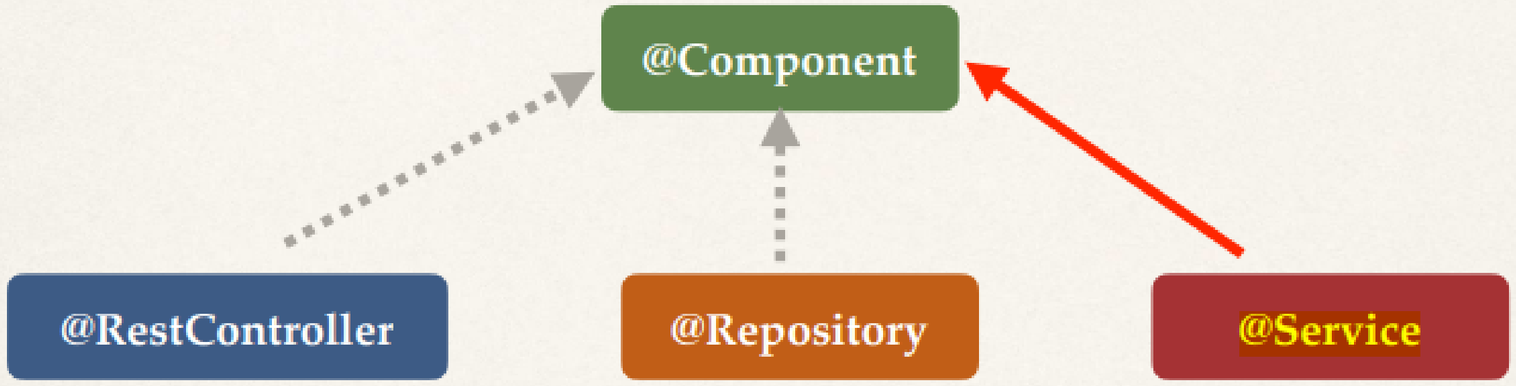
```java
package com.group1.employee.dao;

import com.group1.employee.entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Component;

import java.util.List;

3 usages
@Component
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
    1 usage
    List<Employee> findAllByOrderByLastNameAsc();

}
```

@Component annotation

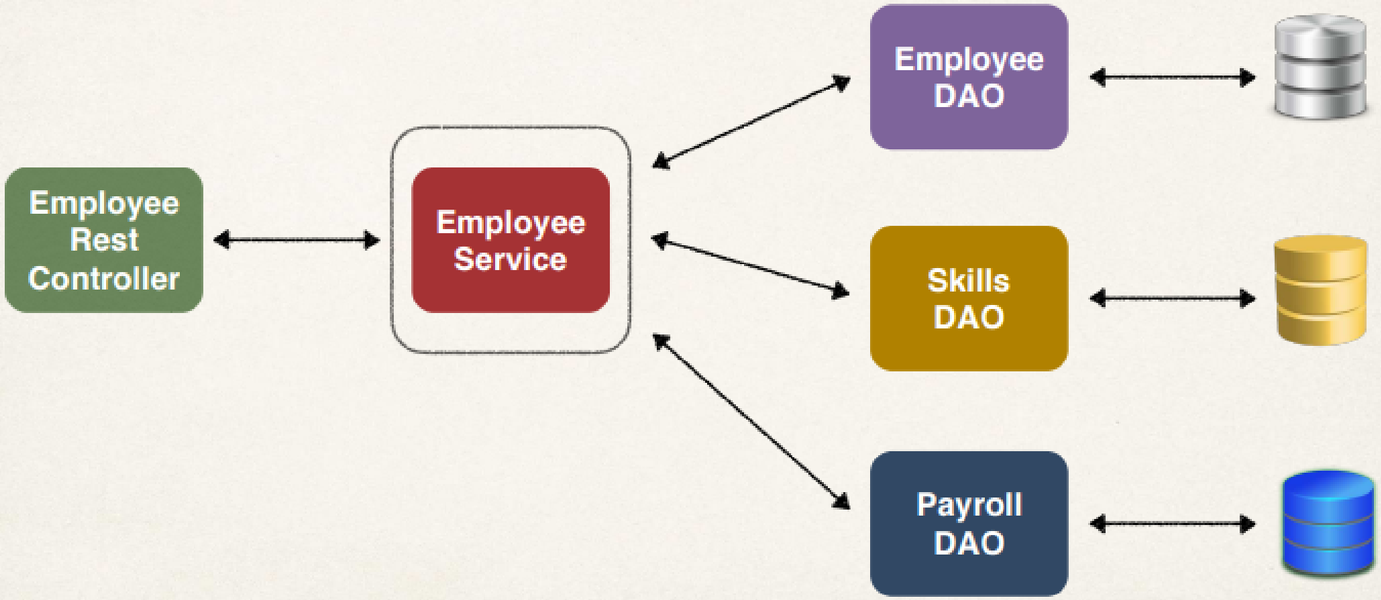
- @Component marks the class as a Spring Bean
  - A Spring Bean is just a regular Java class that is managed by Spring
- @Component also makes the bean available for dependency injection

```java
public interface EmployeeService {

    1 usage   1 implementation
    List<Employee> findAll();

    1 usage   1 implementation
    Employee findById(int theId);

    1 usage   1 implementation
    void save(Employee theEmployee);

    1 usage   1 implementation
    void deleteById(int theId);

}
```

```java
import com.mhwc.springMVC23139.dao.EmployeeRepository;
import com.mhwc.springMVC23139.model.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

@Service
public class EmployeeServiceImpl implements EmployeeService {

    5 usages
    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) { employeeRepository = theEmployeeRepository; }

    1 usage
    @Override
    public List<Employee> findAll() { return employeeRepository.findAllByOrderByLastNameAsc(); }

    1 usage
```



Integrate Multiple Data Sources



The major difference between these stereotypes is that they are used for different classifications. When we annotate a class for auto-detection, we should use the respective stereotype.

@Component

We can use @Component across the application to mark the beans as Spring's managed components. Spring will only pick up and register beans with *@Component*, and doesn't look for *@Service* and *@Repository* in general.

@Repository

***@Repository*'s job is to catch persistence-specific exceptions and re-throw them as one of Spring's unified unchecked exceptions**.

@Service

We mark beans with @Service to indicate that they're holding the business logic. Besides being used in the service layer, there isn't any other special use for this annotation.

```java
@Override
public Employee findById(int theId) {
    Optional<Employee> result = employeeRepository.findById(theId);

    Employee theEmployee = null;

    if (result.isPresent()) {
        theEmployee = result.get();
    }
    else {
        // we didn't find the employee
        throw new RuntimeException("Did not find employee id - " + theId);
    }

    return theEmployee;
}
1 usage
@Override
public void save(Employee theEmployee) { employeeRepository.save(theEmployee); }
1 usage
@Override
public void deleteById(int theId) {employeeRepository.deleteById(theId);
}
```

EmployeeController.java | EmployeeService.java | EmployeeServiceImpl.java | application.properties | SpringMvc23139Application.java

```java
import java.util.List;

@Controller
@RequestMapping("/employees")
public class EmployeeController {
    5 usages
    private EmployeeService employeeService;
    public EmployeeController(EmployeeService employeeService) { this.employeeService = employeeService
    //add mapping for list
    @GetMapping("/list")
    public String listEmployees(Model theModel){
        // get employees from DB
        List<Employee> theEmployees = employeeService.findAll();
        //add to the spring model
        theModel.addAttribute( attributeName: "employees",theEmployees);
        return "employees/list-employees";
    }
    @GetMapping("/showFormForAdd")
    public String showFormForAdd(Model theModel){
        //create model atrribute to bind form data
        Employee theEmployee = new Employee();
        theModel.addAttribute( attributeName: "employee",theEmployee);
        return "employees/employee-form";

    }
```

EmployeeController.java | EmployeeService.java | EmployeeServiceImpl.java | application.properties | SpringMvc2313

```java
    }
    @PostMapping("/save")
    public String saveEmployee(@ModelAttribute("employee") Employee theEmployee ){
        //save the employee
        employeeService.save(theEmployee);
        //use a redirect to prevent duplicate submissions
        return "redirect:/employees/list";
    }
    @GetMapping("/showFormForUpdate")
    public String updateEmployee(@RequestParam("employeeId") int theId, Model theModel){
        //get the employee from service
        Employee theEmployee = employeeService.findById(theId);
        //set employee in the model to prepolulate the form
        theModel.addAttribute( attributeName: "employee",theEmployee);
        //send over to our form
        return "employees/employee-form";
    }
    @GetMapping("/delete")
    public String deleteEmployee(@RequestParam("employeeId") int theId){
        //delete the employee
        employeeService.deleteById(theId);
        //redirect to the /employee/list
        return "redirect:/employees/list";
    }
}
```

Version Control | Run | TODO | Problems | Terminal | Services | Build | Dependencies

The **@RequestMapping** annotation can be applied to class-level and/or method-level in a controller. The class-level annotation maps a specific request path or pattern onto a controller. You can then apply additional method-level annotations to make mappings more specific to handler methods.

- The *@GetMapping* annotation is a composed version of ***@RequestMapping*** annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.GET)`.

  The ***@GetMapping*** **annotated methods handle the *HTTP GET* requests** matched with the given URI expression.

- The *@PostMapping* is a specialized version of *@RequestMapping* annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.POST)`.

  The **@PostMapping** **annotated methods handle the *HTTP POST* requests** matched with the given URI expression.

EmployeeController.java | hello.html | employee-form.html | EmployeeService.java | EmployeeServiceImpl.java | application.properties

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

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi"
          crossorigin="anonymous">

    <title>Save Employee</title>
</head>
<body>
    <div class="container">
        <h3>Employee Directory</h3>
        <hr>
        <p class="h4 mb-4">Save Employee</p>
        <form action="#" th:action="@{/employees/save}"
                        th:object="${employee}" method="POST">
            <!-- Add hidden field-->
          <input type="hidden" th:field="*{id}">
```

html > body > div.container > h3

EmployeeController.java | hello.html | employee-form.html | EmployeeService.java | EmployeeServic

```html
        <h3>Employee Directory</h3>
        <hr>
        <p class="h4 mb-4">Save Employee</p>
        <form action="#" th:action="@{/employees/save}"
                        th:object="${employee}" method="POST">
            <!-- Add hidden field-->
          <input type="hidden" th:field="*{id}">

          <input type="text" th:field="*{firstName}"
                class="form-control mb-4 w-25" placeholder="First Name">

            <input type="text" th:field="*{lastName}"
                   class="form-control mb-4 w-25" placeholder="Last Name">
            <input type="text" th:field="*{email}"
                   class="form-control mb-4 w-25" placeholder="Email">

            <button type="submit" class="btn btn-info col-2">Save</button>
        </form>
        <hr>
        <a th:href="@{/employees/list}">Back to Employees List</a>
    </div>


</body>
```

html > body > div.container > form

```html
<body>

<div class="container">

  <h3>Employee Directory</h3>
  <hr>
  <!-- Add a button -->
  <a th:href="@{/employees/showFormForAdd}"
     class="btn btn-primary btn-sm mb-3"
  >
    Add Employee
  </a>

  <table class="table table-bordered table-striped">
    <thead class="table-dark">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Email</th>
      <th>Action</th>
    </tr>
    </thead>
```

```html
<tbody>
<tr th:each="tempEmployee : ${employees}">

  <td th:text="${tempEmployee.firstName}" />
  <td th:text="${tempEmployee.lastName}" />
  <td th:text="${tempEmployee.email}" />

  <!--  Add update link-->
  <td>
    <a th:href="@{/employees/showFormForUpdate(employeeId=${tempEmployee.id})}"
     class="btn btn-info btn-sm">
      Update
    </a>
    <!-- Add delete button link-->
    <a th:href="@{/employees/delete(employeeId=${tempEmployee.id})}"
    class="btn btn-danger btn-sm"
    onclick="if (!(confirm('Are you sure you want to delete this employee?')))return false">
      Delete
    </a>
  </td>
</tr>
</tbody>
</table>
```

Employee Directory

Save Employee

First Name

Last Name

Email

Save

Back to Employees List

http://localhost:8080/employees/list

localhost:8080 says

Are you sure you want to delete this employee?

OK Cancel

Employee Directory

Add Employee

First Name	Last Name	Email	Action
Kirezi	Ornella	ornellakirezi8@gmail.com	Update Delete
Munyaneza	Roger	kenny@gmail.com	Update Delete
Munezero	Willy	munezerowilly5@gmail.com	Update Delete

Employee Directory

Add Employee

First Name	Last Name	Email	Action
Munyaneza	Roger	kenny@gmail.com	Update Delete
Munezero	Willy	munezerowilly5@gmail.com	Update Delete

Any Questions you can ask?

Thank you!!

GROUP 1 MEMBERS

- 23139 Munezero Hirwa Willy Christel
- 23043 Munyaneza Kenny Roger
- 23456 Rukundo Justin
- 23704 Kirezi ornella
- 23484 Niyomukiza Mordecai
- 23623 Umutoni Liliane
- 21403 Ahishakiye Jeanne
- 20853 Muhoza jules
- 23445 Giramata Arlène
- 22292 Shalom Dauphin NYARWAYA
- 23990 Hakizimana Bahati
- 22209 Benitha INGABIRE
- 22883 Izabayo Anne
- 22887 Niragire solange
- 23279 Iradukunda Léa
- 23163 Kamali Wilson