Spring Web MVC

-> It is one module in Spring Framework to develop web applications.

-> Web MVC module simplified web application development process.

1) Form Binding (form <---> java obj)
2) Flexibility in Form Binding (type conversion)
3) Multiple Presentation Technologies (JSP & Thymeleaf)
4) Form Tag Library (ready-made tags support)

Note: To develop web application using spring-boot we need to add below starter in pom.xml

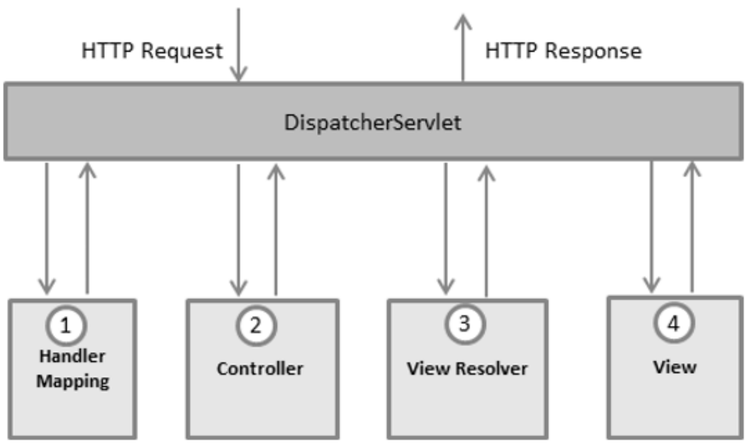
### spring-boot-starter-web ###

-> The above starter provides support for below things

1) MVC based web applications
2) RESTFul Services
3) Embedded Container (Tomcat)

Spring Web MVC Architecture

1) DispatcherServlet
2) Handler Mapper
3) Controller
4) ModelAndView
5) ViewResolver
6) View



=> DispatcherServlet: Framework Servlet / Front Controller.

### Responsible to perform Pre-Processing and Post-Processing of request

=> Handler Mapper : Responsible to identify Request Handler class (controller)

=> Controller: Java class which is responsible to handle request & response

### Controller will return ModelAndView object to DispatcherServlet.

Model: Represents data in key-value format

View: Logical File Name

Note: Controllers are loosely coupled with Presentation technology.

=> ViewResolver: To identify presentation file location and technology

=> View: It is responsible to render Model data in view file.

Building First Web App with Spring Boot

1) Create Spring Starter Project with below dependencies

a) spring-boot-starter-web

b) spring-boot-devtools

c) tomcat-embed-jasper (mvnrepository.com)

2) Create controller class with required methods & map controller methods to URL pattern

3) Create View File with presentation logic

4) Configure View Resolver in application.properties file

5) Run the application and test it.

Observations

-> devtools dependency is used to restart our server when we make code changes.

-> To represent java class as controller we are using @Controller annotation

-> Controller methods we need to map with HTTP methods using unique URL pattern

GET --> @GetMapping

POST --> @PostMapping

-> Apache Tomcat is coming as default embedded container.

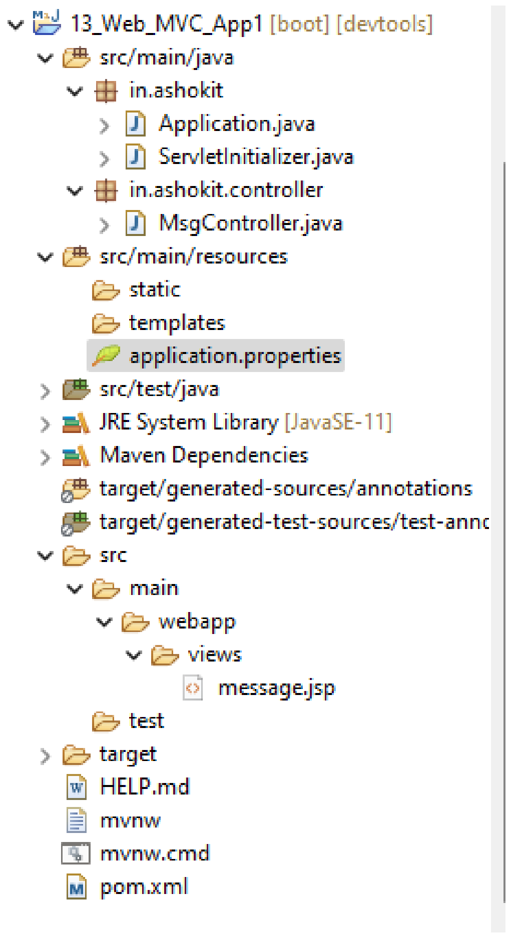
-> Embedded container port number is 8080. We can change that port number using application.properties file

server.port = 9090

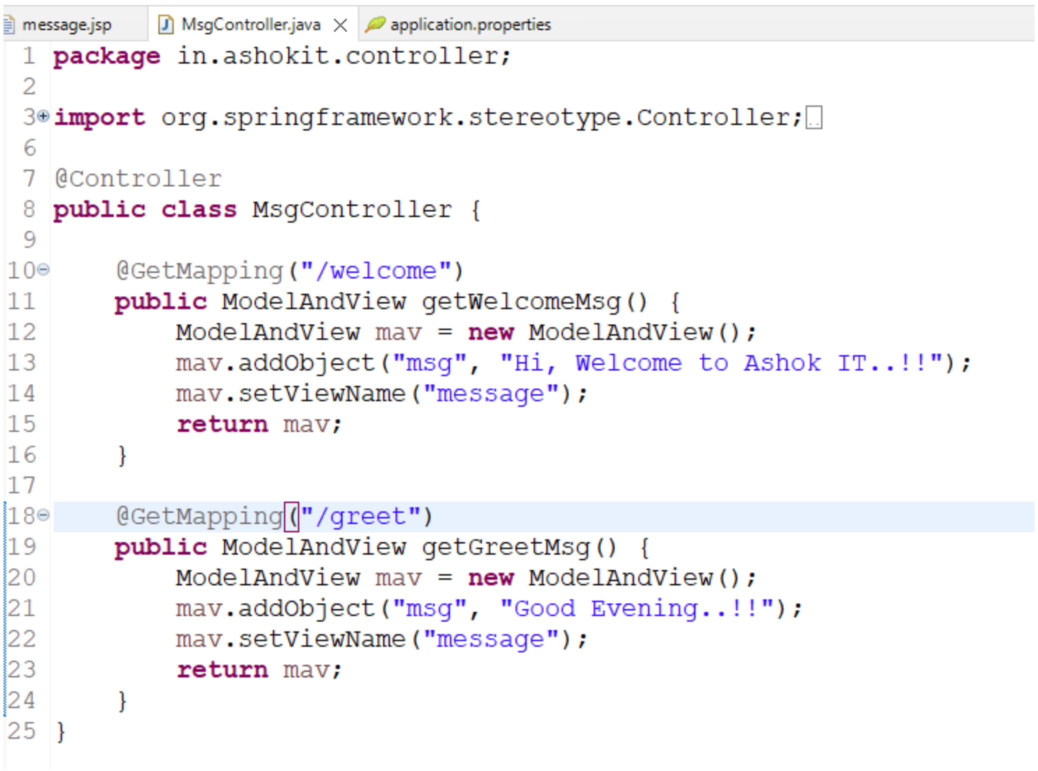
-> Spring Boot web apps will not have context path. We can add context-path using application.properties file.

server.servlet.context-path=/ashokit

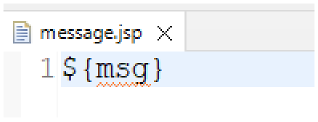
Application Code



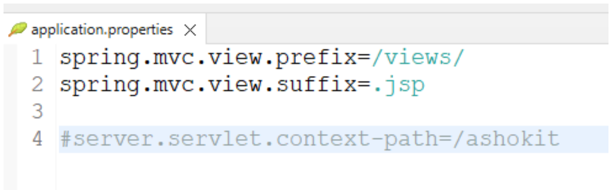
```
13_Web_MVC_App1 [boot] [devtools]
  src/main/java
    in.ashokit
      Application.java
      ServletInitializer.java
    in.ashokit.controller
      MsgController.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annc
  src
    main
      webapp
        views
          message.jsp
    test
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```


message.jsp MsgController.java application.properties

```java
package in.ashokit.controller;

import org.springframework.stereotype.Controller;

@Controller
public class MsgController {

    @GetMapping("/welcome")
    public ModelAndView getWelcomeMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Hi, Welcome to Ashok IT..!!");
        mav.setViewName("message");
        return mav;
    }

    @GetMapping("/greet")
    public ModelAndView getGreetMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Good Evening..!!");
        mav.setViewName("message");
        return mav;
    }
}
```


message.jsp

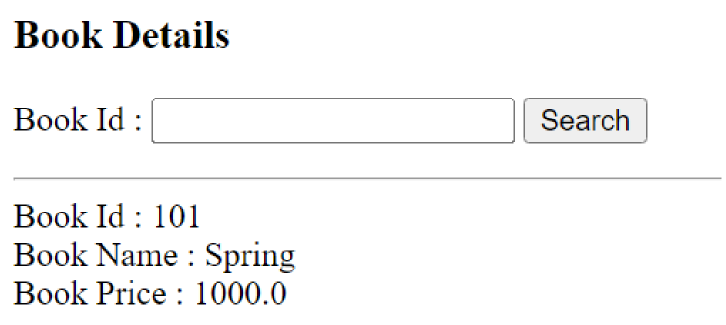
```
${msg}
```


application.properties

```
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp

#server.servlet.context-path=/ashokit
```

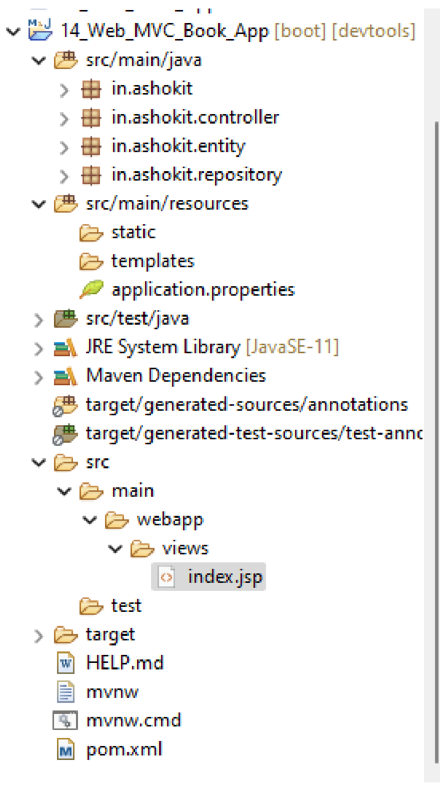
02-WebApplication Requirement:

Retrieve book record based on given id and display in web page like below


Book Details

Book Id : Search

Book Id : 101
Book Name : Spring
Book Price : 1000.0



```
14_Web_MVC_Book_App [boot] [devtools]
  src/main/java
    in.ashokit
    in.ashokit.controller
    in.ashokit.entity
    in.ashokit.repository
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annc
  src
    main
      webapp
        views
          index.jsp
    test
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

1) Create Spring Starter Project with below dependencies

a) web-starter

b) data-jpa

c) mysql-connector-j

d) lombok

e) devtools

f) tomcat-embed-jasper

2) Configure View Resolver & Data Source properties in application.properties file

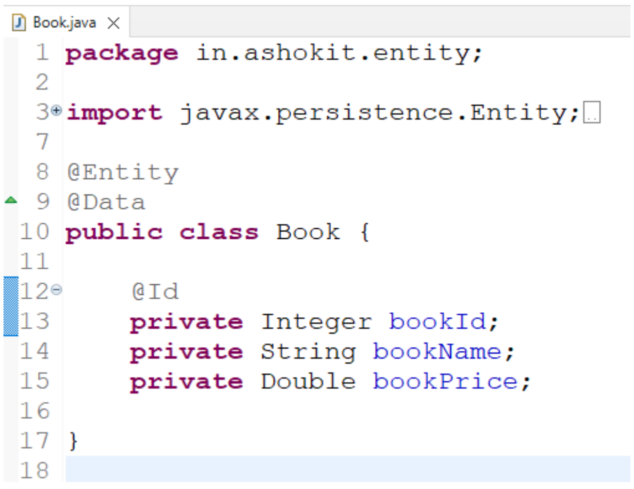
3) Create Entity class (table mapping)

4) Create Jpa Repository interface

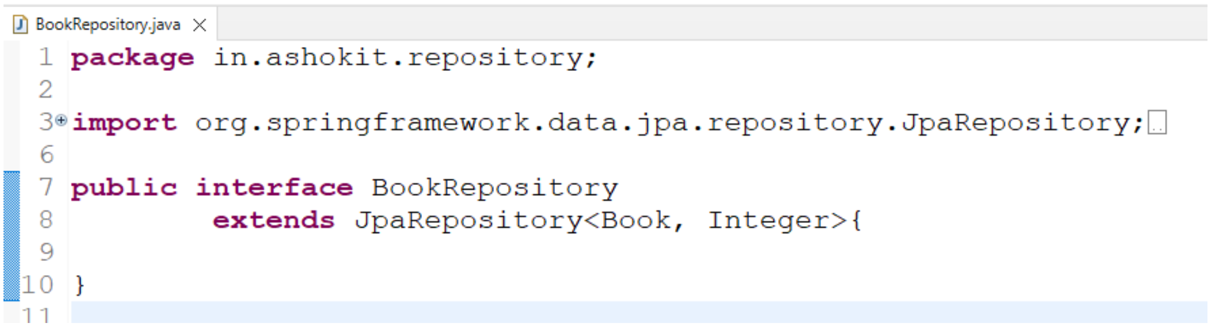
5) Create Controller class with methods to handle request & response

6) Create View Page

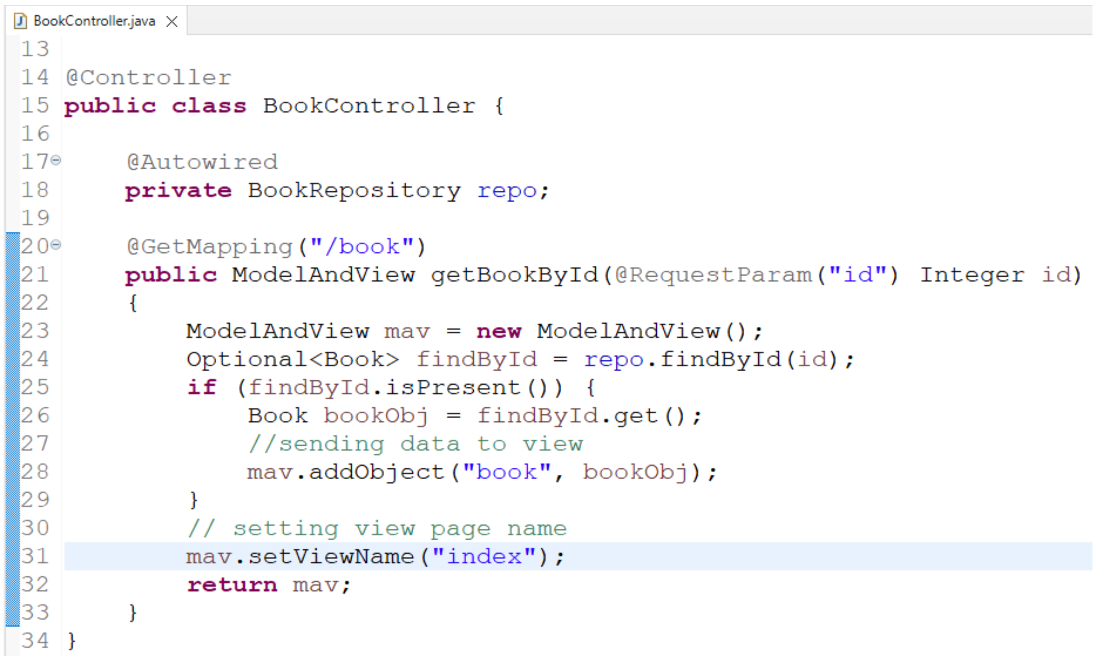
7) Run the application and test it



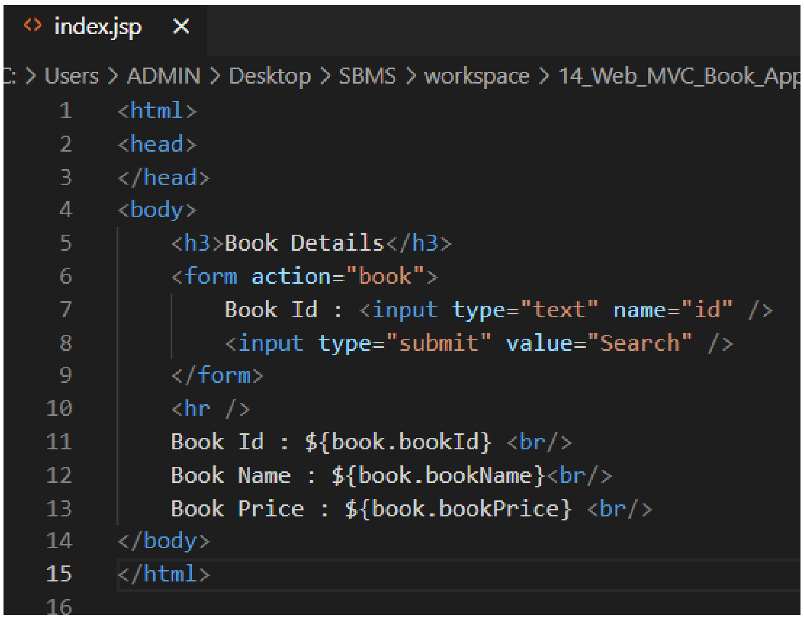
```java
package in.ashokit.entity;

import javax.persistence.Entity;

@Entity
@Data
public class Book {

    @Id
    private Integer bookId;
    private String bookName;
    private Double bookPrice;

}

```



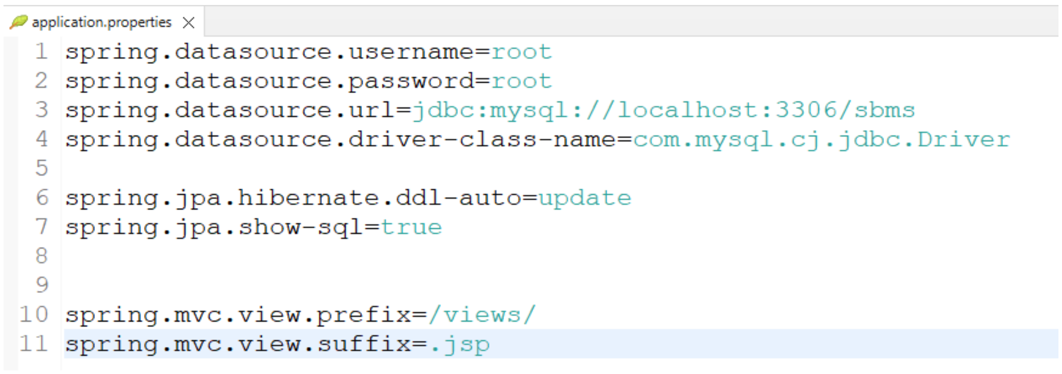
```java
package in.ashokit.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository
        extends JpaRepository<Book, Integer>{

}

```



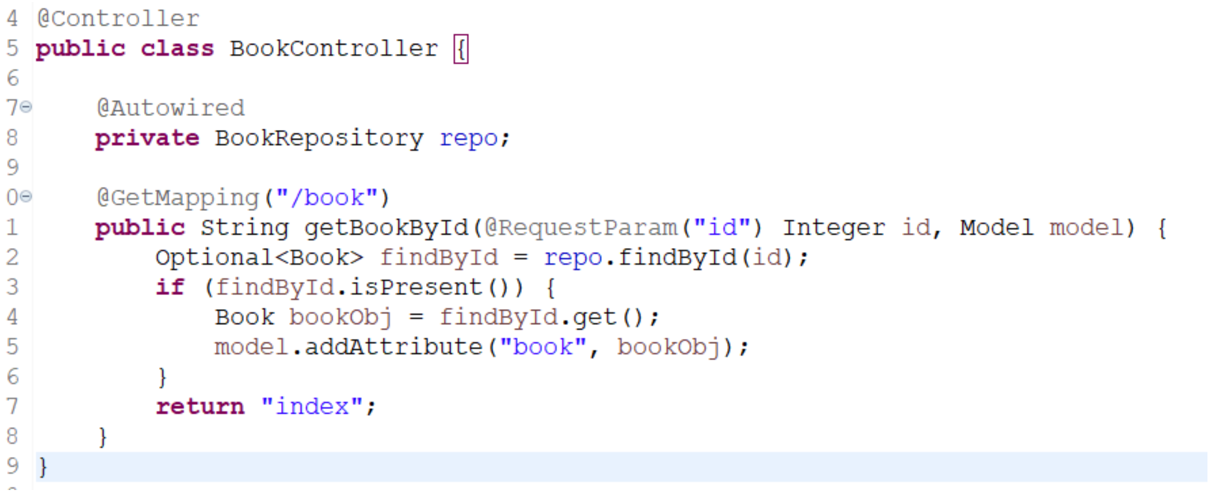
```java

@Controller
public class BookController {

    @Autowired
    private BookRepository repo;

    @GetMapping("/book")
    public ModelAndView getBookById(@RequestParam("id") Integer id)
    {
        ModelAndView mav = new ModelAndView();
        Optional<Book> findById = repo.findById(id);
        if (findById.isPresent()) {
            Book bookObj = findById.get();
            //sending data to view
            mav.addObject("book", bookObj);
        }
        // setting view page name
        mav.setViewName("index");
        return mav;
    }
}
```


index.jsp
C: > Users > ADMIN > Desktop > SBMS > workspace > 14_Web_MVC_Book_App

```
<html>
<head>
</head>
<body>
    <h3>Book Details</h3>
    <form action="book">
        Book Id : <input type="text" name="id" />
        <input type="submit" value="Search" />
    </form>
    <hr />
    Book Id : ${book.bookId} <br/>
    Book Name : ${book.bookName}<br/>
    Book Price : ${book.bookPrice} <br/>
</body>
</html>
```

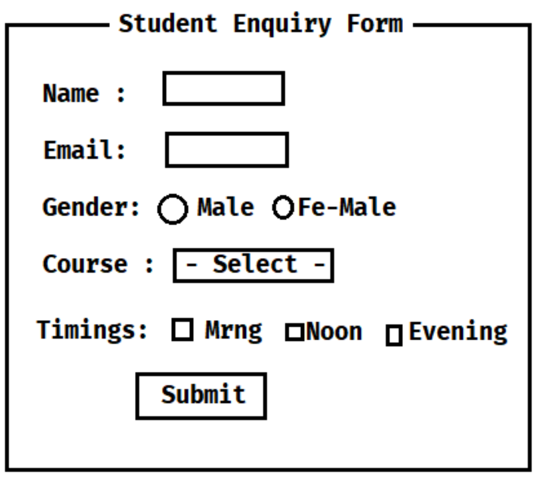


```
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.url=jdbc:mysql://localhost:3306/sbms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true


spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
```



```
@Controller
public class BookController {

    @Autowired
    private BookRepository repo;

    @GetMapping("/book")
    public String getBookById(@RequestParam("id") Integer id, Model model) {
        Optional<Book> findById = repo.findById(id);
        if (findById.isPresent()) {
            Book bookObj = findById.get();
            model.addAttribute("book", bookObj);
        }
        return "index";
    }
}
```

03 – Web Application Requirement : Develop Student Enquiry Form like below



1) Course name drop down values should come from database table

2) Timings checkboxes options should come from database table

Note: When we click on submit button record should inserted into database table (STUDENT_ENQUIRIES) and display success message on the same page.

Spring MVC Form Tag Library

-> Predefined tags provided to simplify Forms development

-> To use Spring MVC form tag library in jsp we have to add below taglib url

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

-> By using prefix we can access tags like below

1) <form:form>

2) <form:input>

3) <form:password>

4) <form:radioButton> & <form:radiobuttons>

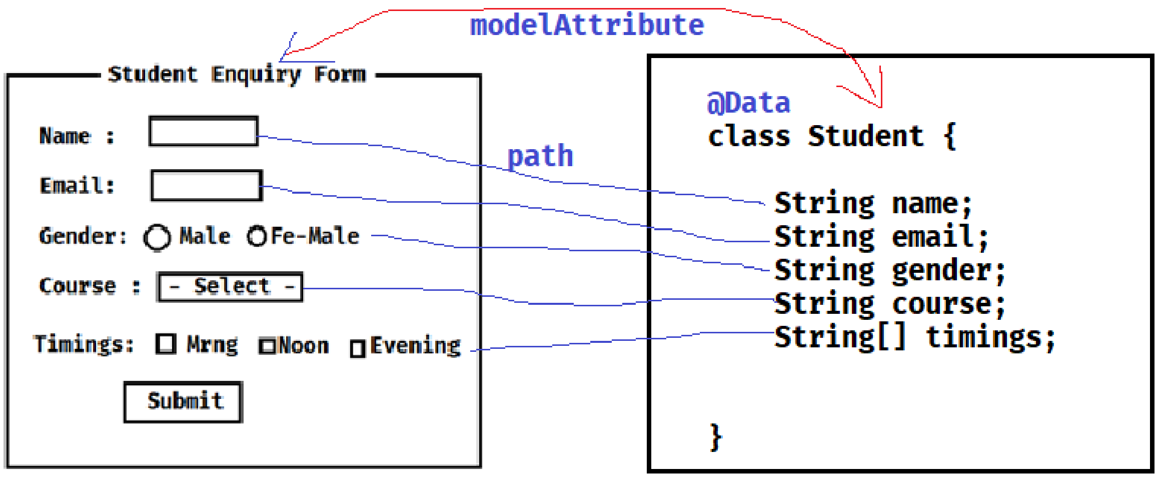
5) <form:select>

6) <form:option> & <form:options>

7) <form:checkbox> & <form:checkboxes>

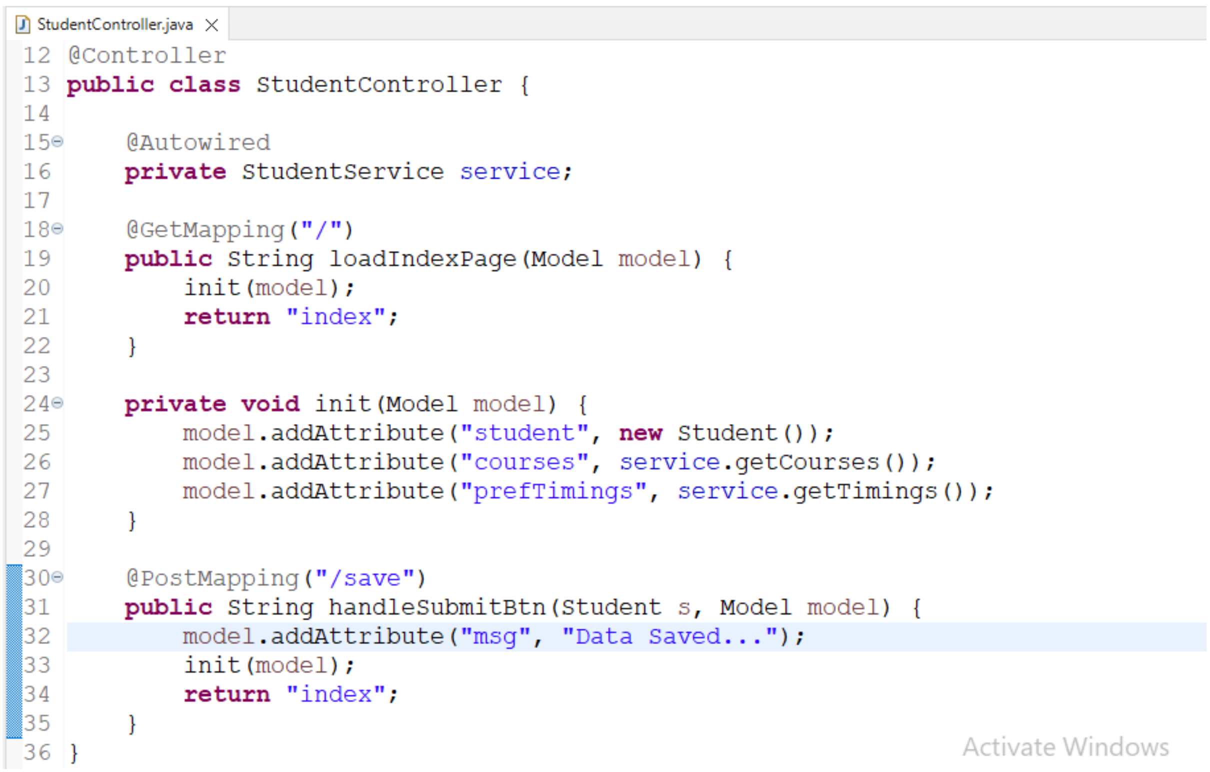
8) <form:hidden>

9) <form:textarea>



```java
@Data
public class Student {

    private String name;
    private String email;
    private String gender;
    private String course;
    private String[] timings;

}
```

```java
@Service
public class StudentService {

    public List<String> getCourses() {
        return Arrays.asList("Java", "Python", "AWS", "DevOps");
    }

    public List<String> getTimings() {
        return Arrays.asList("Morning", "Afternoon", "Evening");
    }
}
```



```java
@Controller
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping("/")
    public String loadIndexPage(Model model) {
        init(model);
        return "index";
    }

    private void init(Model model) {
        model.addAttribute("student", new Student());
        model.addAttribute("courses", service.getCourses());
        model.addAttribute("prefTimings", service.getTimings());
    }

    @PostMapping("/save")
    public String handleSubmitBtn(Student s, Model model) {
        model.addAttribute("msg", "Data Saved...");
        init(model);
        return "index";
    }
}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
</head>
<body>
	<h2>Student Enquiry Form</h2>
	<p>
		<font color='green'>${msg }</font>
	</p>

	<form:form action="save" modelAttribute="student" method="POST">
		<table>
			<tr>
				<td>Name:</td>
				<td><form:input path="name" /></td>
			</tr>

			<tr>
				<td>Email:</td>
				<td><form:input path="email" /></td>
			</tr>

			<tr>
				<td>Gender:</td>
				<td><form:radiobutton path="gender" value="M"
/>Male
					<form:radiobutton path="gender" value="F"
/>Fe-Male
```

```
                  </td>
                  </tr>

                  <tr>
                        <td>Course</td>
                        <td><form:select path="course">
                                    <form:option value="">-Select-
</form:option>
                                    <form:options items="${courses}" />
                              </form:select></td>
                  </tr>
                  <tr>
                        <td>Timings</td>
                        <td><form:checkboxes items="${prefTimings}"
path="timings" /></td>
                  </tr>
                  <tr>
                        <td></td>
                        <td><input type="submit" value="Save" /></td>
                  </tr>
            </table>
      </form:form>
</body>
</html>
```

Embedded Database

-> Embedded Database is called as In-Memory Database / Temporary database

Ex: H2 DB

-> When application starts H2 DB will start. When application stopped database will be stopped.

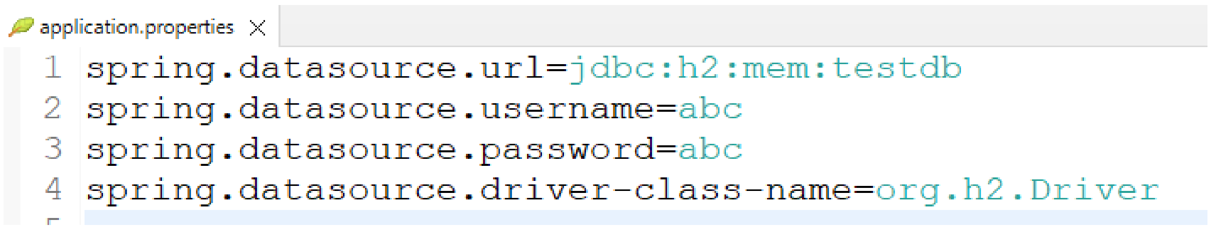
Note: Embedded Databases are used for Proof Of Concept (POC) development.

How to use Embedded DB in Spring Boot

-) Add H2 dependency in pom.xml file

```
  <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
  </dependency>
```

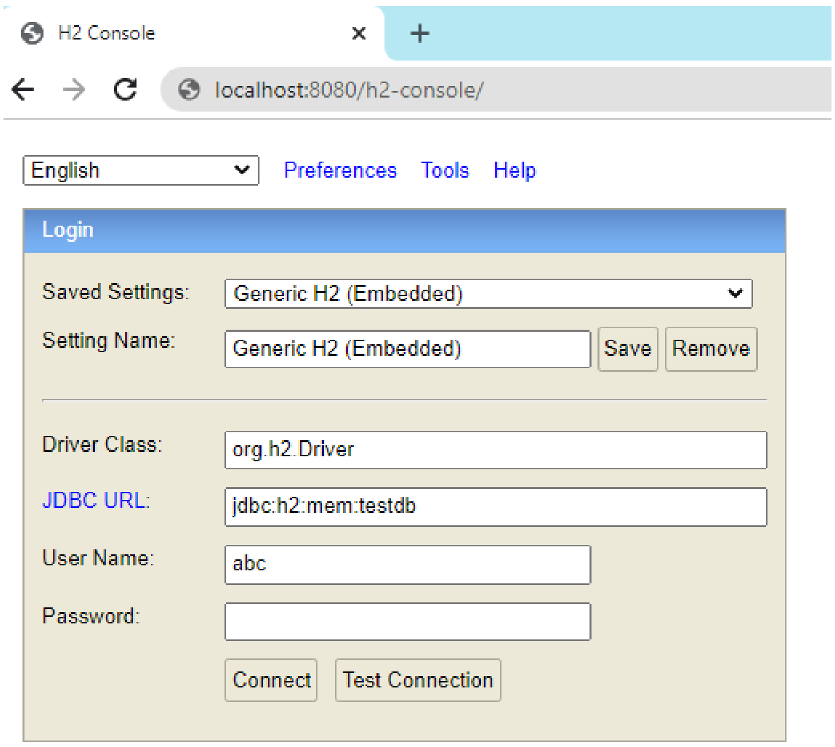
2) Configure H2 DB data source properties in application.properties file



application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=abc
spring.datasource.password=abc
spring.datasource.driver-class-name=org.h2.Driver
```

3) Once application started we can access H2 DB console using below URL



H2 Console

localhost:8080/h2-console/

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

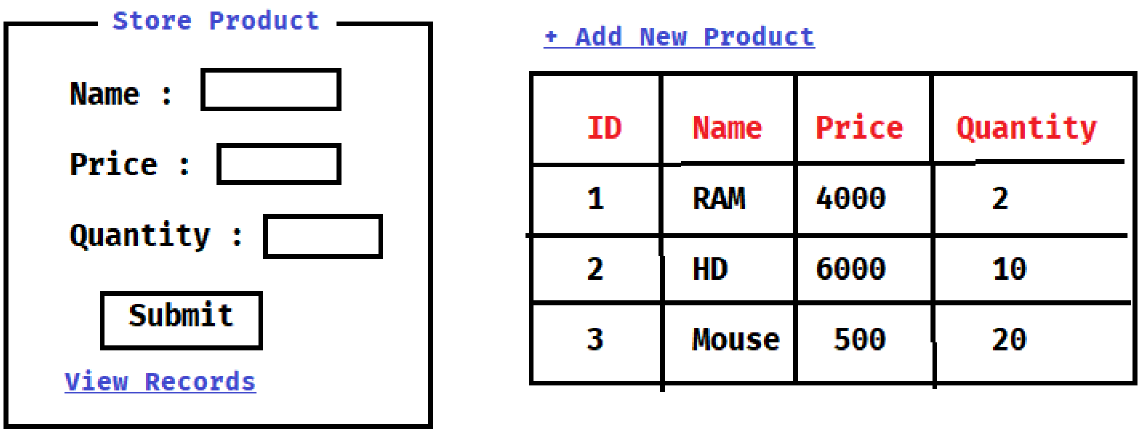
Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

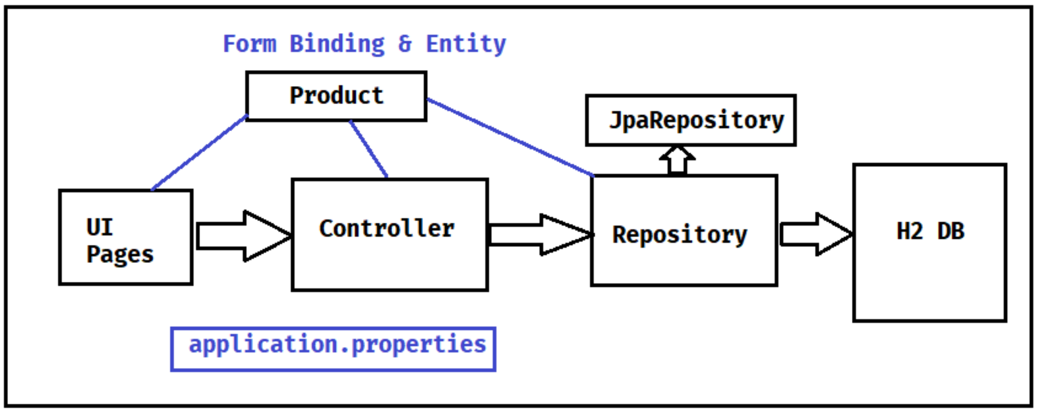
User Name: abc

Password:

Task



ID	Name	Price	Quantity
1	RAM	4000	2
2	HD	6000	10
3	Mouse	500	20



1) Create Spring Starter Project with below dependencies

    a) web-starter
    b) jpa-starter
    c) h2 db
    d) lombok
    e) tomcat-embed-jasper
    f) jstl

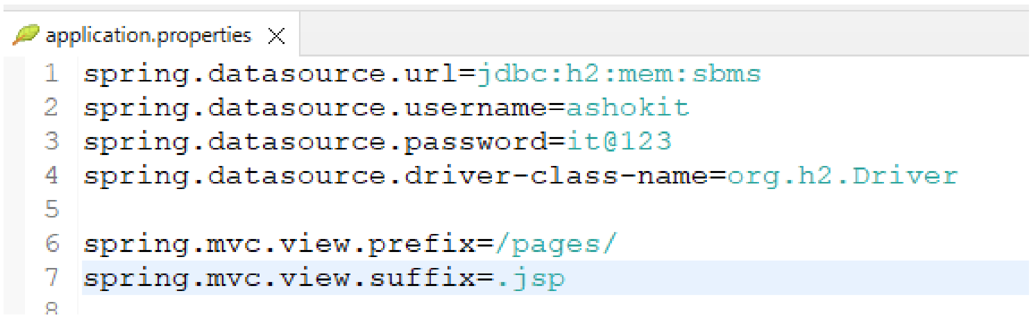
2) Configure Data Source Properties and View Resolver Properties in application.properties file

3) Create Entity Class & Repository interface

4) Create Controller class with Required Methods

5) Create View Pages

6) Run the application



```
spring.datasource.url=jdbc:h2:mem:sbms
spring.datasource.username=ashokit
spring.datasource.password=it@123
spring.datasource.driver-class-name=org.h2.Driver

spring.mvc.view.prefix=/pages/
spring.mvc.view.suffix=.jsp
```

```java
@Data
@Entity
public class Product {

    @Id
    @GeneratedValue
    private Integer pid;
    private String name;
    private Double price;
    private Integer qty;

}
```

ProductRepository.java

```java
package in.ashokit.repo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Integer>{

}
```

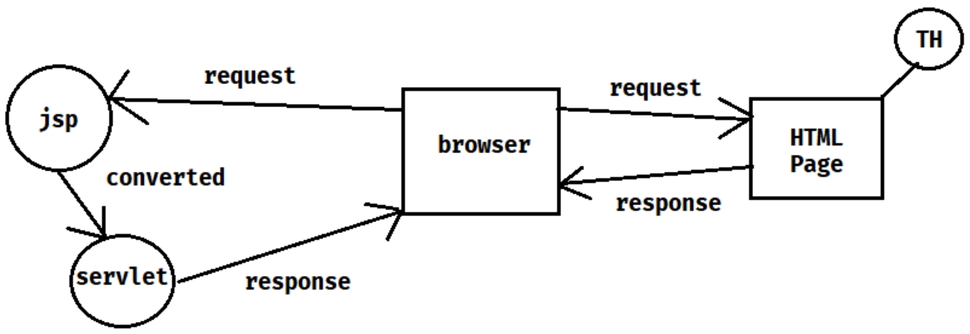
```java
@Controller
public class ProductController {

    @Autowired
    private ProductRepository repo;

    @GetMapping("/products")
    public String loadProducts(Model model) {
        model.addAttribute("products", repo.findAll());
        return "data";
    }

    @GetMapping("/")
    public String loadForm(Model model) {
        model.addAttribute("p", new Product());
        return "index";
    }

    @PostMapping("/product")
    public String handleSave(@ModelAttribute("p") Product p, Model model) {
        p = repo.save(p);
        if (p.getPid() != null) {
            model.addAttribute("msg", "Product Saved");
        }
        return "index";
    }
}
```

Thymeleaf

- ➩ It is a template engine
- ➩ In spring web mvc we can use Thymeleaf as presentation technology
- ➩ We can use Thymeleaf as replacement for JSP

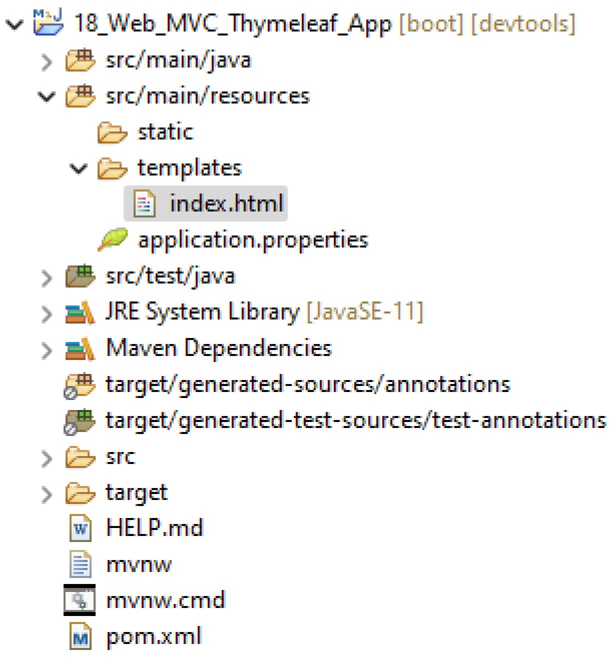
Note: Every JSP page should be converted to Servlet to send response to browser hence performance wise JSP is slow.

- ➩ **To overcome JSP problems we are using Thymeleaf for presentation layer development.**

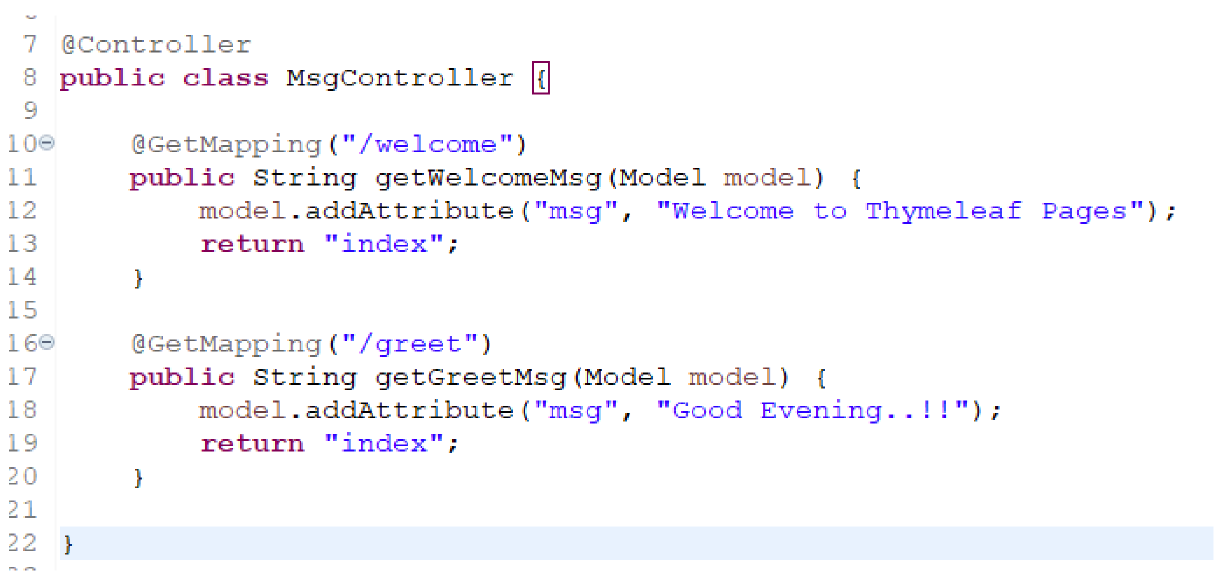


- ➩ **To work with Thymeleaf we need to add below starter in pom.xml file**

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

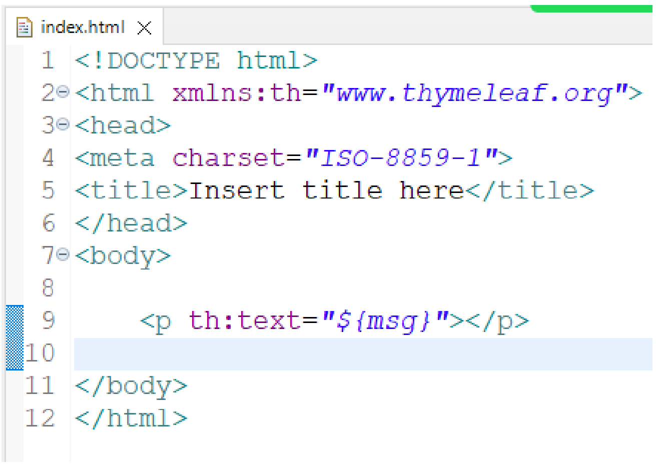


```
18_Web_MVC_Thymeleaf_App [boot] [devtools]
  src/main/java
  src/main/resources
    static
    templates
      index.html
    application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```



```java
@Controller
public class MsgController {

    @GetMapping("/welcome")
    public String getWelcomeMsg(Model model) {
        model.addAttribute("msg", "Welcome to Thymeleaf Pages");
        return "index";
    }

    @GetMapping("/greet")
    public String getGreetMsg(Model model) {
        model.addAttribute("msg", "Good Evening..!!");
        return "index";
    }

}
```

➔ **Create below HTML page under src/main/resources/templates folder.**

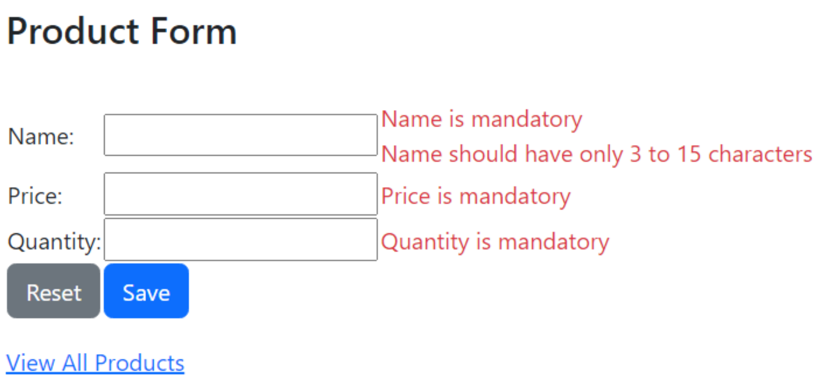


index.html

```html
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <p th:text="${msg}"></p>

</body>
</html>
```

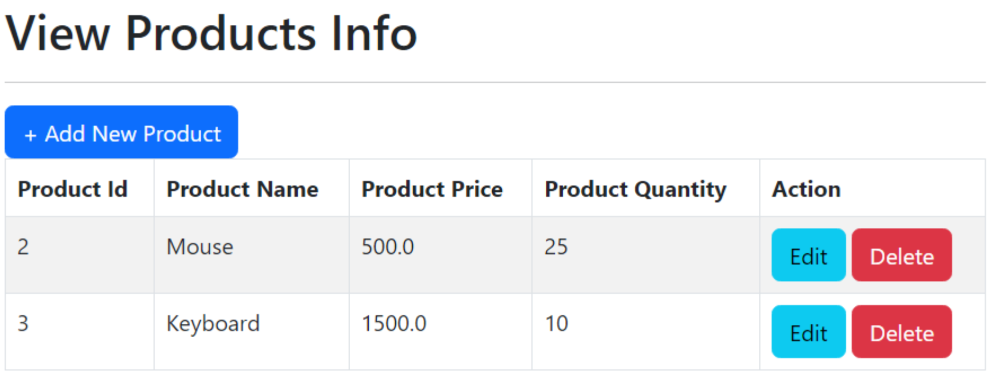
Assignment: Develop Product application using Thymeleaf

1) **Save Product**
2) **View Products**
3) **Edit and Update Product**
4) **Delete Product**



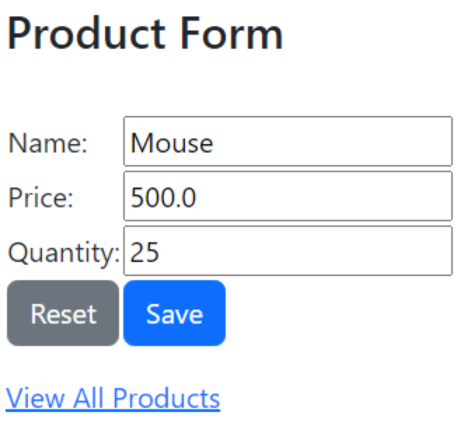
Product Form

Name:		Name is mandatory Name should have only 3 to 15 characters
Price:		Price is mandatory
Quantity:		Quantity is mandatory



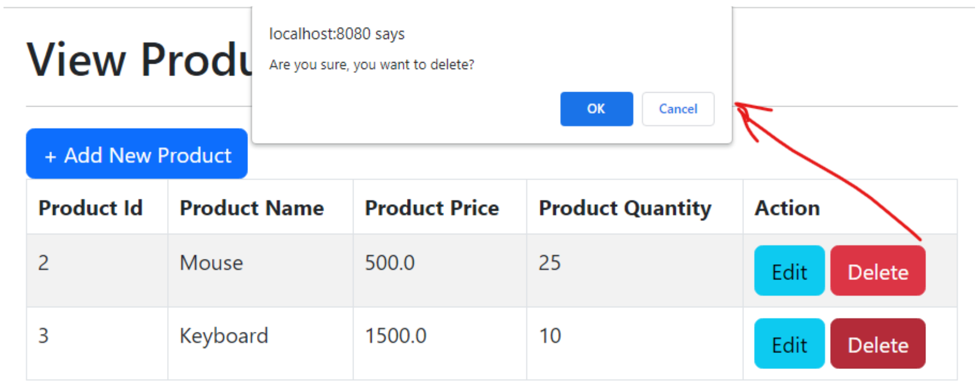
View Products Info

Product Id	Product Name	Product Price	Product Quantity	Action
2	Mouse	500.0	25	Edit Delete
3	Keyboard	1500.0	10	Edit Delete



Product Form

Name:	Mouse
Price:	500.0
Quantity:	25



localhost:8080 says

Are you sure, you want to delete?

Product Id	Product Name	Product Price	Product Quantity	Action
2	Mouse	500.0	25	Edit Delete
3	Keyboard	1500.0	10	Edit Delete

1) Create Spring Starter Project with below dependencies

- **a) web-starter**
- **b) thymeleaf-starter**
- **c) data-jpa-starter**
- **d) h2 driver**
- **e) lombok**
- **f) validation-starter**
- **g) devtools**

2) Configure Datasource properties in application.properties file

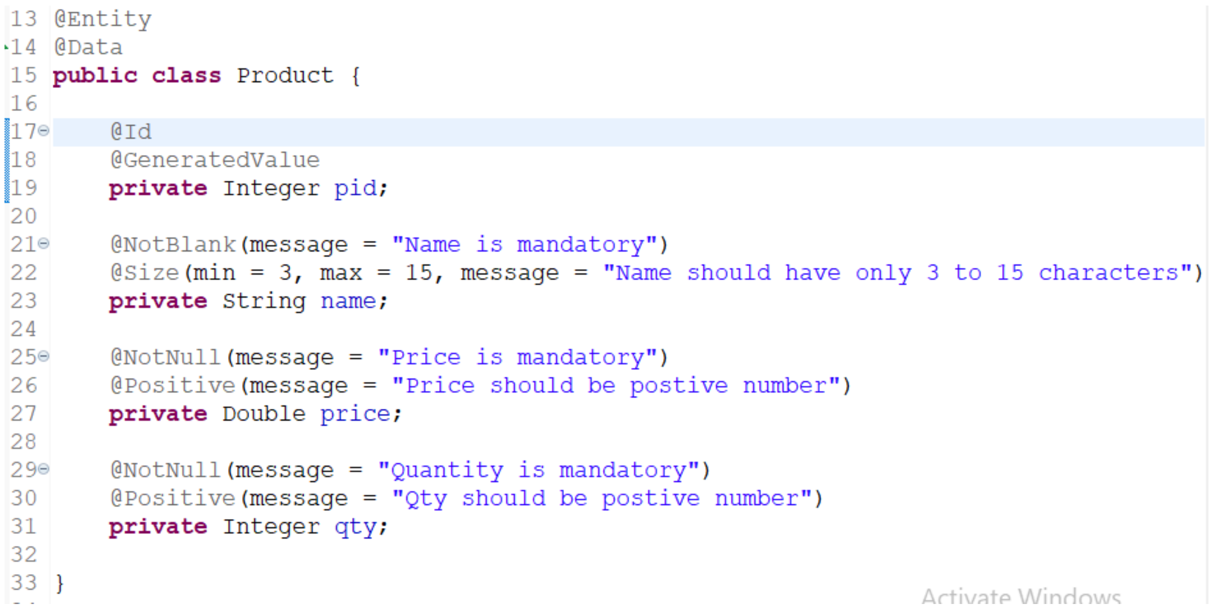
3) Create Entity class & Repository interface

4) Create Controller class with required methods

5) Create View Files using Thymeleaf and Bootstrap

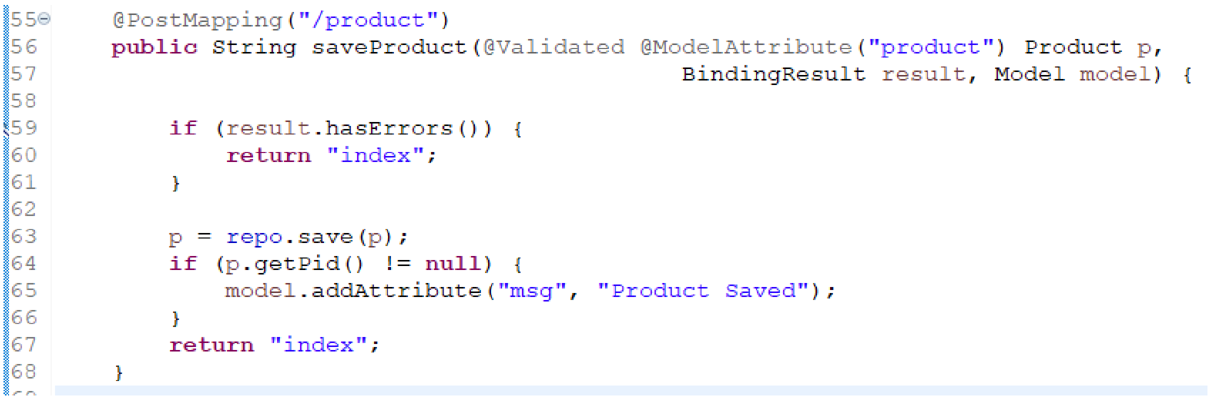
Implementing Form Validations

1) Write Validation rules using annotations in binding class like below



```
@Entity
@Data
public class Product {

    @Id
    @GeneratedValue
    private Integer pid;

    @NotBlank(message = "Name is mandatory")
    @Size(min = 3, max = 15, message = "Name should have only 3 to 15 characters")
    private String name;

    @NotNull(message = "Price is mandatory")
    @Positive(message = "Price should be postive number")
    private Double price;

    @NotNull(message = "Quantity is mandatory")
    @Positive(message = "Qty should be postive number")
    private Integer qty;

}
```

2) Make changes to controller method to valid form data. If form validations are failed then return same page



```
 @PostMapping("/product")
 public String saveProduct(@Validated @ModelAttribute("product") Product p,
                                              BindingResult result, Model model) {

     if (result.hasErrors()) {
         return "index";
     }

     p = repo.save(p);
     if (p.getPid() != null) {
         model.addAttribute("msg", "Product Saved");
     }
     return "index";
 }
```

3) Print Validation message in the form for every field like below

```
<tr>
    <td>Price:</td>
    <td><input type="number" th:field="*{price}"></td>
    <td th:if="${#fields.hasErrors('price')}" th:errors="*{price}"
        class="text-danger"></td>
</tr>
```