

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

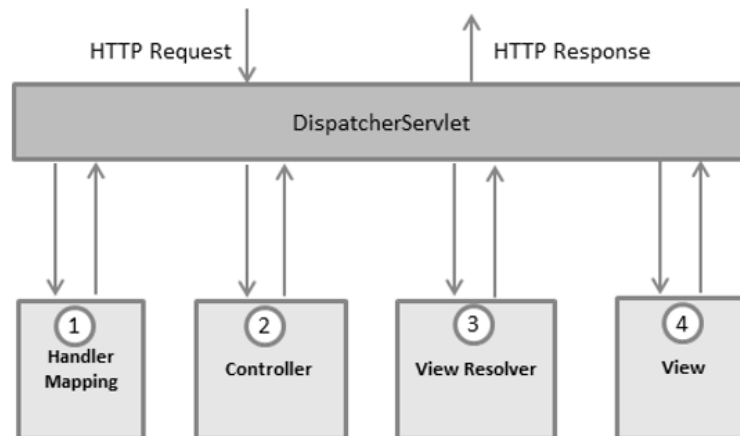
spring-boot-starter-web

-> 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.

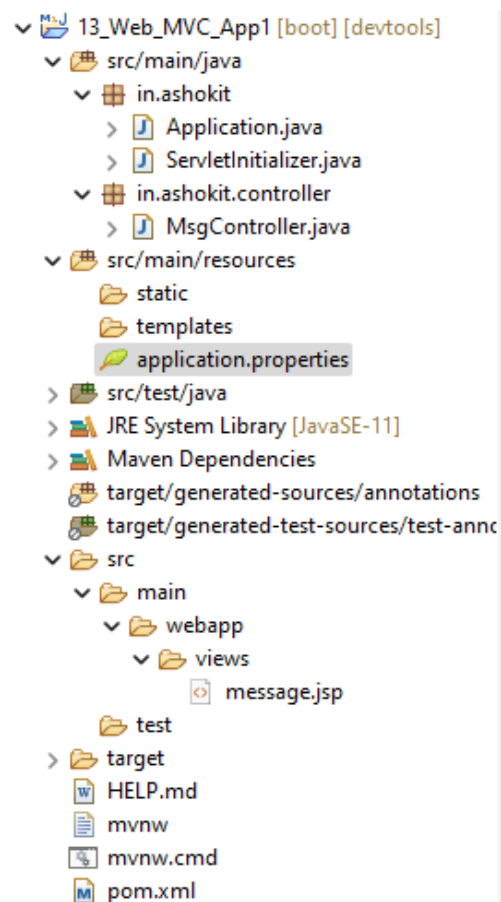
-> Embedded container port number is 8080. We can change that port number using application.properties file

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



```
message.jsp | MsgController.java | application.properties
1 package in.ashokit.controller;
2
3 import org.springframework.stereotype.Controller;
4
5
6
7 @Controller
8 public class MsgController {
9
10     @GetMapping("/welcome")
11     public ModelAndView getWelcomeMsg() {
12         ModelAndView mav = new ModelAndView();
13         mav.addObject("msg", "Hi, Welcome to Ashok IT..!!");
14         mav.setViewName("message");
15         return mav;
16     }
17
18     @GetMapping("/greet")
19     public ModelAndView getGreetMsg() {
20         ModelAndView mav = new ModelAndView();
21         mav.addObject("msg", "Good Evening..!!");
22         mav.setViewName("message");
23         return mav;
24     }
25 }
```

```
message.jsp |
1 ${msg}
```

```
application.properties |
1 spring.mvc.view.prefix=/views/
2 spring.mvc.view.suffix=.jsp
3
4 #server.servlet.context-path=/ashokit
```

02-WebApplication Requirement:

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

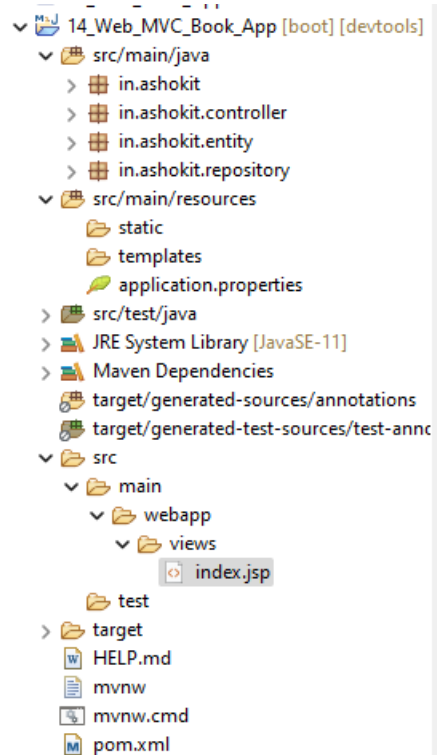
Book Details

Book Id :

Book Id : 101

Book Name : Spring

Book Price : 1000.0



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

```
Book.java X
1 package in.ashokit.entity;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 @Data
7 public class Book {
8
9     @Id
10    private Integer bookId;
11    private String bookName;
12    private Double bookPrice;
13 }
14
```

```
BookRepository.java X
1 package in.ashokit.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface BookRepository
6     extends JpaRepository<Book, Integer>{
7
8 }
9
```

```
BookController.java X
13
14 @Controller
15 public class BookController {
16
17     @Autowired
18     private BookRepository repo;
19
20     @GetMapping("/book")
21     public ModelAndView getBookById(@RequestParam("id") Integer id)
22     {
23         ModelAndView mav = new ModelAndView();
24         Optional<Book> findById = repo.findById(id);
25         if (findById.isPresent()) {
26             Book bookObj = findById.get();
27             //sending data to view
28             mav.addObject("book", bookObj);
29         }
30         // setting view page name
31         mav.setViewName("index");
32         return mav;
33     }
34 }
```

```
<> index.jsp X
C:\Users\ADMIN\Desktop> SBMS > workspace > 14_Web_MVC_Book_App
1  <html>
2  <head>
3  </head>
4  <body>
5      <h3>Book Details</h3>
6      <form action="book">
7          Book Id : <input type="text" name="id" />
8          <input type="submit" value="Search" />
9      </form>
10     <hr />
11     Book Id : ${book.bookId} <br/>
12     Book Name : ${book.bookName}<br/>
13     Book Price : ${book.bookPrice} <br/>
14 </body>
15 </html>
16
```

```
application.properties X
1 spring.datasource.username=root
2 spring.datasource.password=root
3 spring.datasource.url=jdbc:mysql://localhost:3306/sbms
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5
6 spring.jpa.hibernate.ddl-auto=update
7 spring.jpa.show-sql=true
8
9
10 spring.mvc.view.prefix=/views/
11 spring.mvc.view.suffix=.jsp
```

```
4 @Controller
5 public class BookController {
6
7     @Autowired
8     private BookRepository repo;
9
10    @GetMapping("/book")
11    public String getBookById(@RequestParam("id") Integer id, Model model) {
12        Optional<Book> findById = repo.findById(id);
13        if (findById.isPresent()) {
14            Book bookObj = findById.get();
15            model.addAttribute("book", bookObj);
16        }
17        return "index";
18    }
19 }
```

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

Student Enquiry Form

Name :

Email:

Gender: ☐ Male ☐ Fe-Male

Course :

Timings: ☐ Mrng ☐ Noon ☐ Evening

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>

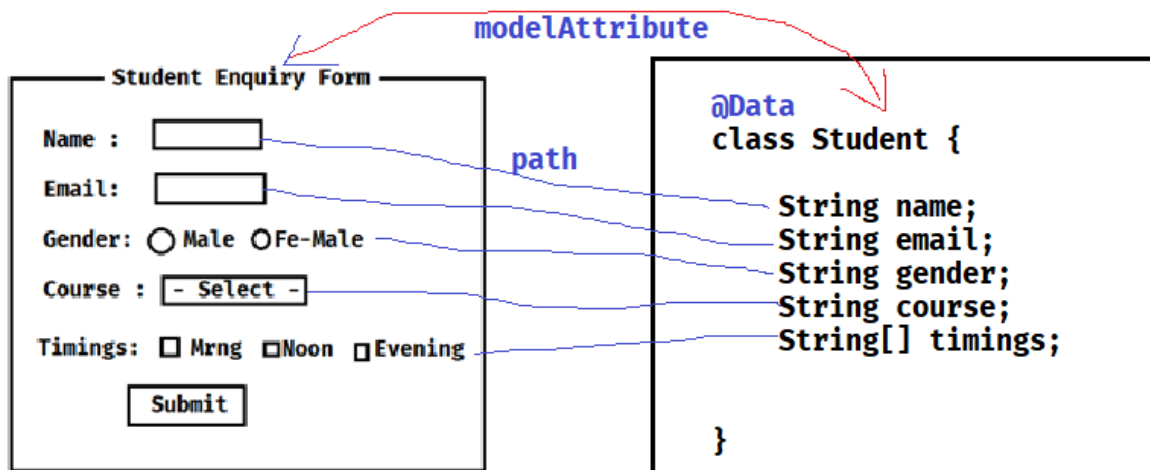
5) <form:select>

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

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

8) <form:hidden>

9) <form:textarea>



```

5 @Data
6 public class Student {
7
8     private String name;
9     private String email;
10    private String gender;
11    private String course;
12    private String[] timings;
13
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
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

```

```

8 @Service
9 public class StudentService {
10
11     public List<String> getCourses() {
12         return Arrays.asList("Java", "Python", "AWS", "DevOps");
13     }
14
15     public List<String> getTimings() {
16         return Arrays.asList("Morning", "Afternoon", "Evening");
17     }
18 }
19

```

```

StudentController.java x
12 @Controller
13 public class StudentController {
14
15     @Autowired
16     private StudentService service;
17
18     @GetMapping("/")
19     public String loadIndexPage(Model model) {
20         init(model);
21         return "index";
22     }
23
24     private void init(Model model) {
25         model.addAttribute("student", new Student());
26         model.addAttribute("courses", service.getCourses());
27         model.addAttribute("prefTimings", service.getTimings());
28     }
29
30     @PostMapping("/save")
31     public String handleSubmitBtn(Student s, Model model) {
32         model.addAttribute("msg", "Data Saved...");
33         init(model);
34         return "index";
35     }
36 }

```

Activate Windows

```

<%@ 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"
                    />Female
                </td>
            </tr>
        </table>
    </form:form>

```

```

        </td>
      </tr>

      <tr>
        <td>Course</td>
        <td><form:select path="course">
          <form:option value="">-Select-

          <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

```

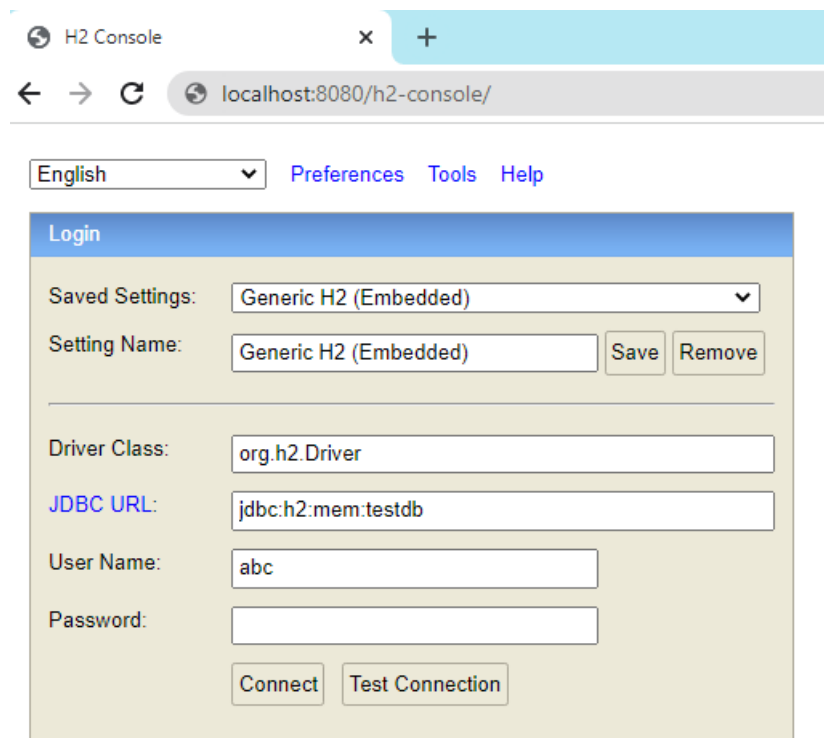
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>

```

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

```
application.properties ×
1 spring.datasource.url=jdbc:h2:mem:testdb
2 spring.datasource.username=abc
3 spring.datasource.password=abc
4 spring.datasource.driver-class-name=org.h2.Driver
```

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



Task

Store Product

Name :

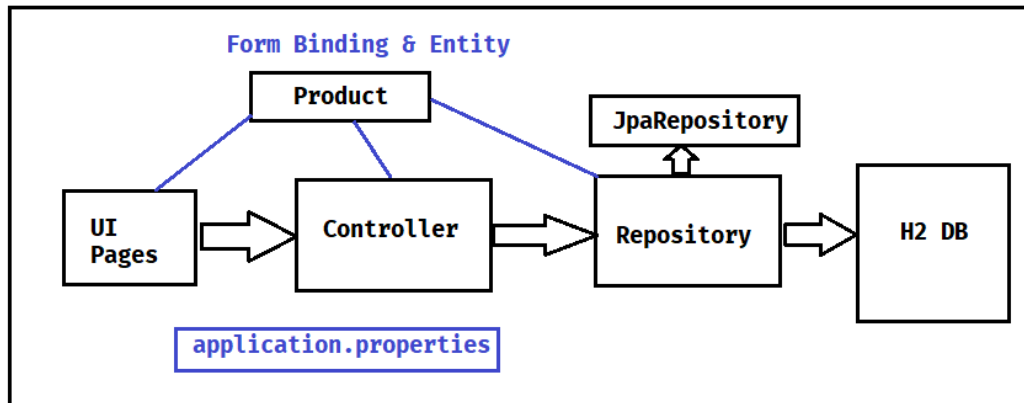
Price :

Quantity :

[View Records](#)

+ Add New Product

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

```
application.properties X
1 spring.datasource.url=jdbc:h2:mem:sbms
2 spring.datasource.username=ashokit
3 spring.datasource.password=it@123
4 spring.datasource.driver-class-name=org.h2.Driver
5
6 spring.mvc.view.prefix=/pages/
7 spring.mvc.view.suffix=.jsp
```

```
9 @Data
10 @Entity
11 public class Product {
12
13     @Id
14     @GeneratedValue
15     private Integer pid;
16     private String name;
17     private Double price;
18     private Integer qty;
19
20 }
```

```
ProductRepository.java X
1 package in.ashokit.repo;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface ProductRepository extends JpaRepository<Product, Integer>{
8
9 }
```

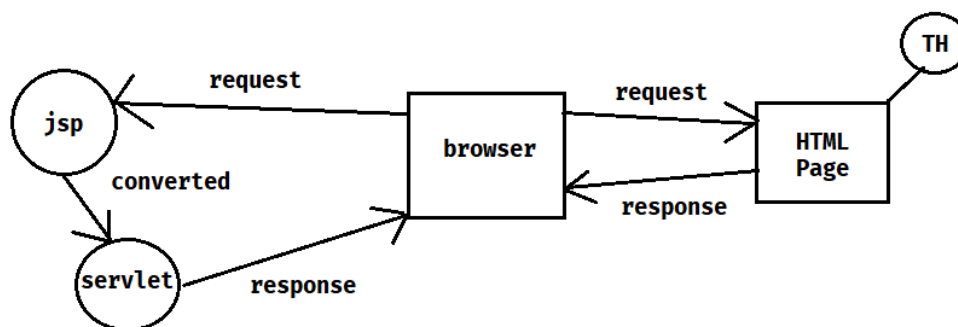
```
13 @Controller
14 public class ProductController {
15
16     @Autowired
17     private ProductRepository repo;
18
19     @GetMapping("/products")
20     public String loadProducts(Model model) {
21         model.addAttribute("products", repo.findAll());
22         return "data";
23     }
24
25     @GetMapping("/")
26     public String loadForm(Model model) {
27         model.addAttribute("p", new Product());
28         return "index";
29     }
30
31     @PostMapping("/product")
32     public String handleSave(@ModelAttribute("p") Product p, Model model) {
33         p = repo.save(p);
34         if (p.getPid() != null) {
35             model.addAttribute("msg", "Product Saved");
36         }
37         return "index";
38     }
39 }
40
```

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

```
~
7 @Controller
8 public class MsgController {
9
10     @GetMapping("/welcome")
11     public String getWelcomeMsg(Model model) {
12         model.addAttribute("msg", "Welcome to Thymeleaf Pages");
13         return "index";
14     }
15
16     @GetMapping("/greet")
17     public String getGreetMsg(Model model) {
18         model.addAttribute("msg", "Good Evening...!!");
19         return "index";
20     }
21
22 }
```

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

```
index.html X
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 </head>
7 <body>
8
9     <p th:text="${msg}"></p>
10
11 </body>
12 </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 All Products](#)

View Products Info

[+ Add New Product](#)

Product Id	Product Name	Product Price	Product Quantity	Action	
2	Mouse	500.0	25	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
3	Keyboard	1500.0	10	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>

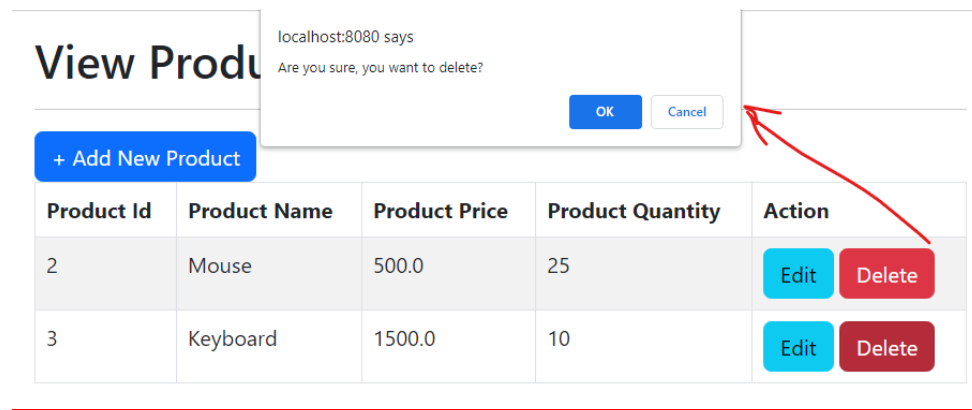
Product Form

Name:

Price:

Quantity:

[View All Products](#)



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

```

13 @Entity
14 @Data
15 public class Product {
16
17     @Id
18     @GeneratedValue
19     private Integer pid;
20
21     @NotBlank(message = "Name is mandatory")
22     @Size(min = 3, max = 15, message = "Name should have only 3 to 15 characters")
23     private String name;
24
25     @NotNull(message = "Price is mandatory")
26     @Positive(message = "Price should be postive number")
27     private Double price;
28
29     @NotNull(message = "Quantity is mandatory")
30     @Positive(message = "Qty should be postive number")
31     private Integer qty;
32
33 }

```

Activate Windows

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

```

55 @PostMapping("/product")
56 public String saveProduct(@Validated @ModelAttribute("product") Product p,
57                           BindingResult result, Model model) {
58
59     if (result.hasErrors()) {
60         return "index";
61     }
62
63     p = repo.save(p);
64     if (p.getPid() != null) {
65         model.addAttribute("msg", "Product Saved");
66     }
67     return "index";
68 }

```

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>

```

How to change embedded container from tomcat to jetty?

1) Exclude tomcat starter from web starter in pom.xml file using <exclusions>

2) Add jetty starter dependency in pom.xml file

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

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

Exception Handling in Web MVC App

=> Unexpected and unwanted situation is called as Exception

=> When exception occurs program will terminate abnormally

=> In Spring Boot we can handle exceptions in below 2 ways

1) Local Exception Handling (Controller Specific)

2) Global Exception Handling (Application specific)

```
7 @Controller
8 public class DemoController {
9
10     @GetMapping("/wish")
11     public String getWishMsg(Model model) {
12         String msgTxt = "Good Evening..!!";
13         String s = null;
14         s.length();
15         model.addAttribute("msg", msgTxt);
16         return "index";
17     }
18 }
19
```

```
8 @ControllerAdvice
9 public class AppExceptionHandler {}
10
11     private Logger logger = LoggerFactory.getLogger(AppExceptionHandler.class);
12
13     @ExceptionHandler(value = Exception.class)
14     public String handleAE(Exception ae) {
15         String msg = ae.getMessage();
16         logger.error(msg);
17         return "errorPage";
18     }
19 }
20
```

```
errorPage.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 </head>
7 <body>
8
9     <h2>Some Problem Occurred, Please try after sometime..</h2>
10
11 </body>
12 </html>
```

Spring Web MVC Summary

- 1) What is Spring Web MVC?
- 2) Web Application vs. Distributed Application
- 3) What are the advantages with Spring Web MVC?
- 4) Spring Web MVC Architecture
- 5) What is DispatcherServlet?
- 6) What is Handler Mapper?
- 7) What are Controller / Request Handler?
- 8) What is ModelAndView?
- 9) What is View Resolver?
- 10) What is View?
- 11) HTTP Methods (@GetMapping & @PostMapping)

- 12) What is @RequestParam ?
- 13) Spring MVC Form Tag Library
- 14) H2 Database
- 15) What is Thymeleaf?
- 16) How to perform Form Validations?
- 17) CRUD App with Boot + Web MVC + Data JPA + Thymeleaf + BS
- 18) What is @ModelAttribute?
- 19) How to change Embedded Container from Tomcat to Jetty?
- 20) How to change Embedded Container Port Number?
- 21) How to configure Context Path in Spring Boot?
- 22) What is @ResponseBody annotation?
- 23) What is @Controller & @RestController?
- 24) What is URL ambiguity in Spring Web MVC?
- 25) How to handle Exceptions in Spring Web MVC?
- 26) What is Local Exception Handling?
- 27) What is Global Exception Handling?
- 28) What is @ControllerAdvice and @ExceptionHandler ?