Book Store management System using Spring Boot, Thymeleaf and MySQL DB

OBJECTIVE

To demonstrate the basic concepts of Spring Boot, Thymeleaf along with **E-commerce Website** using MySQL DB

ABSTRACT

Backend

- Spring Boot Application
  - Model
  - Repository
  - Controller
  - Configuration

Frontend

- Thymeleaf

Database

- MySQL

Author: Dr.Leena Silvoster M, AP-CSE

Contents

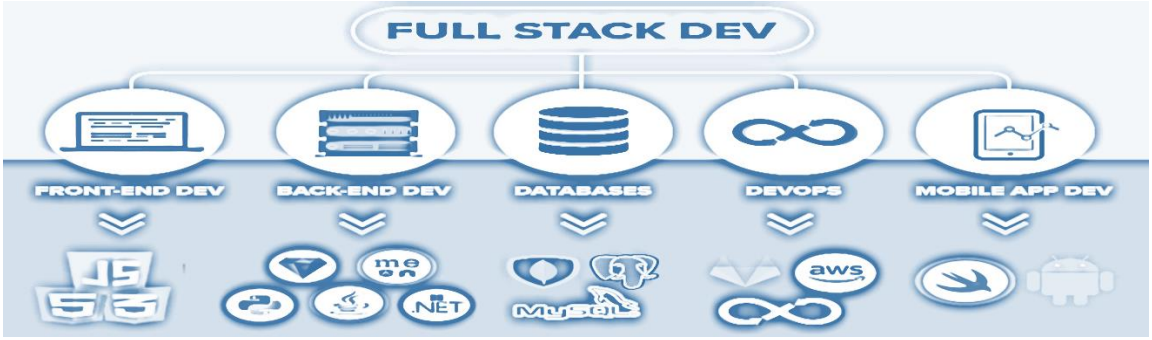
E-COMMERCE WEBSITE USING SPRING BOOT, THYMELEAF AND MYSQL DB

Introduction

Web Application Development

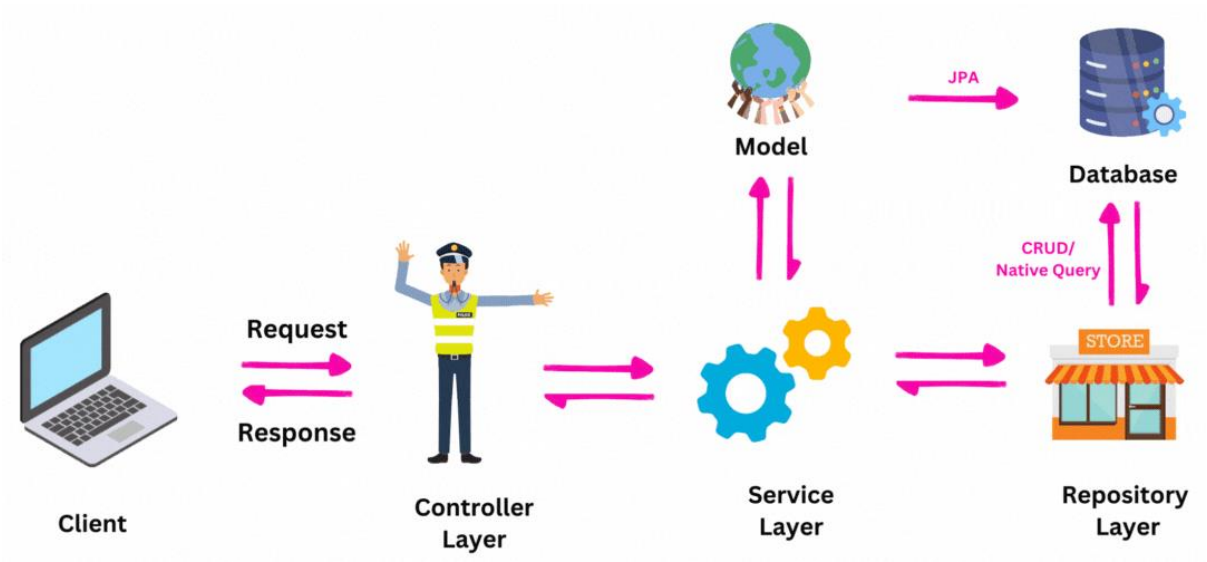
- Web application development is the creation of application programs that reside on remote/local servers and are delivered to the user's device over the Internet / Intranet.

- Web technologies like HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript, which are executed by the browser, are used to create web applications.
- A full-stack developer needs to be proficient in both front-end and back-end of a website



Spring Boot

- Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM).
- Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:
    1. Autoconfiguration
    2. An opinionated approach to configuration
    3. The ability to create standalone applications
- These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup. Spring Boot applications can also be optimized and run with the Open Liberty runtime."

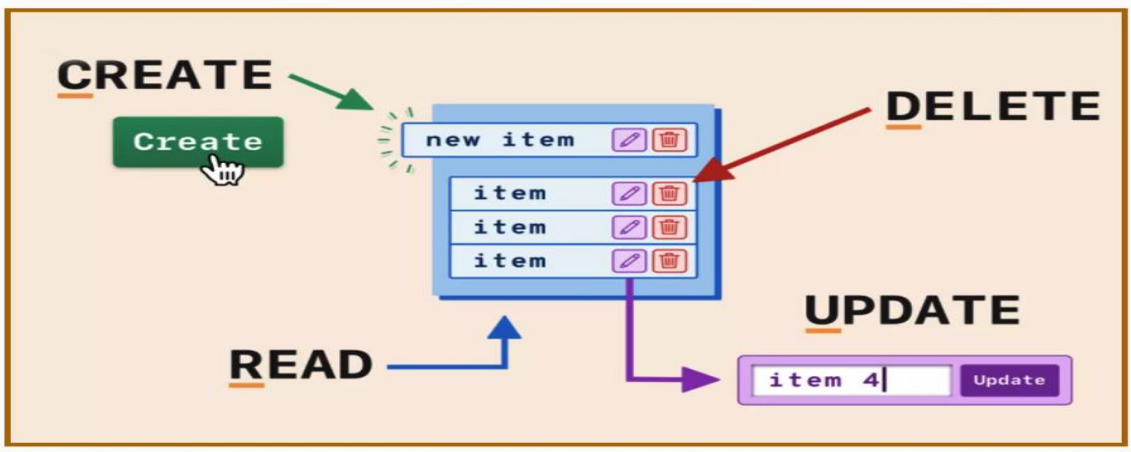


Thymeleaf

- The Thymeleaf is an open-source Java library that is licensed under the Apache License 2.0. It is a HTML5/XHTML/XML template engine. It is a server-side Java template engine for both web (servlet-based) and non-web (offline) environments. It is perfect for modern-day HTML5 JVM web development. It provides full integration with Spring Framework.
- It applies a set of transformations to template files in order to display data or text produced by the application. It is appropriate for serving XHTML/HTML5 in web applications.
- The goal of Thymeleaf is to provide a stylish and well-formed way of creating templates. It is based on XML tags and attributes. These XML tags define the execution of predefined logic on the DOM (Document Object Model) instead of explicitly writing that logic as code inside the template. It is a substitute for JSP.
- The architecture of Thymeleaf allows the fast processing of templates that depends on the caching of parsed files. It uses the least possible amount of I/O operations during execution.

CRUD

- CRUD is an acronym for **C**reate, **R**ead, **U**pdate, and **D**elete. CRUD operations are basic data manipulation for the database
  - Create (C): The create operation involves adding new data to the storage system. i.e., Inserting a new record or document into a table or collection.
  - Read (R): The read operation retrieves existing data from the storage system. It allows you to fetch and view the data.
  - Update (U): The update operation modifies existing data in the storage system. It allows you to make changes to specific records or documents
  - Delete (D): The delete operation removes data from the storage system. It involves the permanent deletion of records or documents.



Project Summary

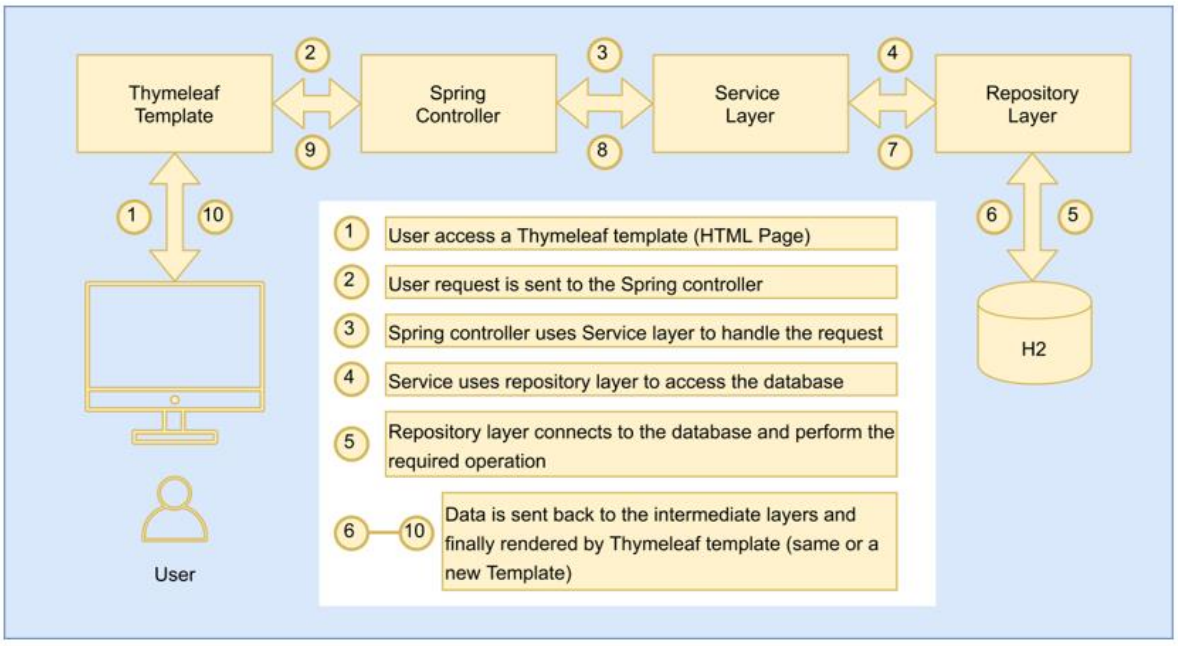
Problem Statements

- The traditional methods of managing a **E-commerce Website** involve manual processes that are time-consuming, error-prone, and lack efficiency. In order to overcome these challenges and enhance the overall management of the shopping store, there is a need for a comprehensive **E-commerce Website** that integrates modern technology to streamline operations, improve accuracy, and provide a better customer experience.

Requirements

- Software Requirements
  - Eclipse IDE
  - Spring Boot Framework
  - Hibernate Framework
  - Thymeleaf

Proposed Architecture



Methodology

- Requirement Gathering
- System Design
- Development
- Testing
- Deployment

Setting Up New Spring Boot Project, Dependency, Development and Deployment

- Create new Spring Boot project in Eclipse IDE
  - New→others →Spring Web Project
  - Add Dependency
    - JPA
    - H2
    - Thymeleaf
    - Spring Boot
  - In Project Folder, Right click → RunAs → Spring → Spring Boot App (i.e EcomcartApplication.java)

```
package com.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EcomcartApplication {

	public static void main(String[] args) {

		SpringApplication.run(EcomcartApplication.class, args);

	}

}
```

Working with Model

- Create the package as "model" and store Product.java
- In Product.java, create new book model with the following fields: id (auto increment), productName, price

#Product.java

```
package com.demo.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;
```

```
@Entity
public class Product {
	@Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
	@Column(name = "product_name")
    private String productName;
	@Column(name = "price")
    private double price;

    // Constructors, getters, and setters
    // Constructors
    public Product() {
    }

    public Product(Long id, String productName, double price) {
        this.id = id;
        this.productName = productName;
        this.price = price;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getProductName() {
```

```
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

Working with Repository

- Create the package as "repository" and store ProductRepository.java
- In ProductRepository.java, create the interface that extends from JpaRepository **<Shop, Long>** (i.e) Template that receives Shop Object and Id from Product.java

```
package com.demo.repository;

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

import com.demo.model.Product;

public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

Working with Service

- Create the package as "service" and store ShopService.java
- In ShopService.java, create the modules for CRUD operations using ProductRepository Interface

```
package com.demo.service;
```

```
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.demo.model.Product;
import com.demo.repository.ProductRepository; // Adjusted import

@Service
public class ShopService {
    @Autowired
    private ProductRepository productRepository;

    // Retrieve all products
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    // Retrieve a product by its ID
    public Product getProductById(Long id) {
        return productRepository.findById(id).orElse(null);
    }

    // Save a new or existing product
    public void saveProduct(Product product) {
        productRepository.save(product);
    }

    // Delete a product by its ID
    public void deleteProduct(Long id) {
        productRepository.deleteById(id);
    }
```

}

Working with Controller

- Create the package as “controller” and store Shopcontroller.java
- In Shopcontroller.java, create the REST API end points for getting the ShopService

```
package com.demo.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import com.demo.model.Product; // Adjusted import

import com.demo.service.ShopService;


@Controller

public class ShopController {

	@Autowired

	private ShopService shopService;


	@GetMapping("/")

	public String showIndex() {

		return "index";

	}


	@GetMapping("/shop")

	public String getAllProducts(Model model) {

		List<Product> products = shopService.getAllProducts(); // Changed variable name

		model.addAttribute("products", products); // Changed attribute name
```

```
        return "product-list"; // Changed view name
    }

    @GetMapping("/shop/{id}")
    public String getProductById(@PathVariable Long id, Model model) { // Changed method name
        Product product = shopService.getProductById(id); // Changed variable name
        model.addAttribute("product", product); // Changed attribute name
        return "product-detail"; // Changed view name
    }

    @GetMapping("/shop/new")
    public String showNewProductForm(Model model) { // Changed method name
        model.addAttribute("product", new Product()); // Changed attribute name and class
        return "new-product"; // Changed view name
    }

    @PostMapping("/shop/new")
    public String saveProduct(@ModelAttribute("product") Product product) { // Changed method name and parameter
        shopService.saveProduct(product); // Changed method call
        return "redirect:/shop"; // Changed redirect URL
    }

    @GetMapping("/shop/delete/{id}")
    public String deleteProduct(@PathVariable Long id) { // Changed method name
        shopService.deleteProduct(id); // Changed method call
        return "redirect:/shop"; // Changed redirect URL
    }

    @GetMapping("/shop/edit/{id}")
```

```
    public String showEditForm(@PathVariable Long id, Model model) { // Changed method name
        Product product = shopService.getProductById(id); // Changed variable name and method call
        model.addAttribute("product", product); // Changed attribute name
        return "edit-product"; // Changed view name
    }

    @PostMapping("/shop/edit/{id}")
    public String editProduct(@PathVariable Long id,
            @ModelAttribute("product") Product updatedProduct) { // Changed method name and parameter
        Product existingProduct = shopService.getProductById(id); // Changed variable name and method call
        if (existingProduct != null) {
            existingProduct.setProductName(updatedProduct.getProductName()); // Changed method call
            existingProduct.setPrice(updatedProduct.getPrice()); // Changed method call
            shopService.saveProduct(existingProduct); // Changed method call
        }
        return "redirect:/shop"; // Changed redirect URL
    }
}
```

Working with Configuration

- Create the package as “Config” and store ThymeleafConfig.java
- In ThymeleafConfig.java, setting up the thymeleaf resolver for templates (html files) and static (CSS, JS and media files) folder

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import
org.thymeleaf.spring6.templateresolver.SpringResourceTemplateResolver;
```

```
@Configuration
public class ThymeleafConfig implements WebMvcConfigurer {

    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new
SpringResourceTemplateResolver();
        templateResolver.setPrefix("classpath:/templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode("HTML");
        templateResolver.setCharacterEncoding("UTF-8");
        templateResolver.setCacheable(false);
        return templateResolver;
    }

        @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/**").addResourceLocations("classpath:
/static/");
    }
}
```

Configuring Backend DBs, Hihernate and Thymeleaf

- In application.properties, configure DB, hibernate, thymeleaf,

```
# Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/shopdb

spring.datasource.username=lee

spring.datasource.password=Leena@123

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

# Thymeleaf Configuration

spring.thymeleaf.cache=false

spring.thymeleaf.enabled=true

spring.thymeleaf.check-template-location=false

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html
```

- In templates folder, include the html files for requesting and rendering the results.

#index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Spring Boot CRUD App</title>
  <link rel="stylesheet" th:href="@{css/style.css}" />
  <style>
    body, html {
      height: 100%;
      margin: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      text-align: center;
      font-family: "Monotype Corsiva", cursive, sans-serif; /* Added fallback font */
    }
    .button {
      display: inline-block;
      background-color: #007bff;
      color: white;
      padding: 10px 20px;
      border-radius: 5px;
      text-decoration: none;
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <div>
    <img src="/static/images.png" alt="Logo">
    <h1>Welcome to the SHOPPING CART CRUD Application</h1>
```

```
        <a th:href="@{/shop}" class="button">Go to Product List</a> <!-- Converted to button -->
    </div>
</body>
</html>
```

#product-list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Product List with Shopping Cart</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <link rel="stylesheet" th:href="@{/css/style.css}" />
  <script th:src="@{/static/js/jquery.min.js}"></script> <!-- Assuming the correct path to jQuery -->
  <style>
    /* CSS styles for shopping cart */
    #shopping-cart {
      margin-top: 20px;
      border: 1px solid #ccc;
      padding: 20px;
    }
    #cart-items th, #cart-items td {
      text-align: center;
      vertical-align: middle;
    }
    #cart-items th:first-child, #cart-items td:first-child {
      text-align: left;
    }
    #cart-items td:last-child {
      white-space: nowrap;
    }
    #cart-items .quantity-input {
      width: 50px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2 class="text-center">Product List</h2>
```

```
<!-- Search Input -->
<div class="row justify-content-center">
  <div class="col-md-6">
    <input type="search" id="myInput" class="form-control mb-3" placeholder="Search products...">
  </div>
</div>
<!-- Product List Table -->
<table class="table table-bordered">
  <thead>
    <tr>
      <th>ID</th>
      <th>Product Name</th>
      <th>Price</th>
      <th>Action</th>
      <th>Add to Cart</th> <!-- New column for Add to Cart button -->
    </tr>
  </thead>
  <tbody>
    <!-- Loop through products -->
    <tr th:each="product: ${products}">
      <td th:text="${product.id}"></td>
      <td th:text="${product.productName}"></td>
      <td th:text="${product.price}"></td>
      <td>
        <a th:href="@{'/shop/edit/' + ${product.id}}" class="btn btn-primary btn-sm">Edit</a>
        <a th:href="@{'/shop/delete/' + ${product.id}}" class="btn btn-danger btn-sm">Delete</a>
      </td>
      <td>
        <button class="btn btn-success btn-add-to-cart" th:attr="data-id=${product.id}, data-
name=${product.productName}, data-price=${product.price}">
          Add to Cart
        </button>
      </td>
    </tr>
```

```
      </tbody>
    </table>
    <!-- Add New Product Button -->
    <div class="text-center">
      <a href="/shop/new" class="btn btn-success">Add New Product</a>
    </div>
  </div>

  <!-- Shopping Cart Component -->
  <div id="shopping-cart" class="container">
    <h2 class="text-center">Shopping Cart</h2>
    <table id="cart-items" class="table table-bordered">
      <thead>
        <tr>
          <th>Product Name</th>
          <th>Unit Price</th>
          <th>Quantity</th>
          <th>Overall Price</th>
        </tr>
      </thead>
      <tbody>
        <!-- Cart items will be dynamically added here -->
      </tbody>
      <tfoot>
        <tr>
          <th colspan="3">Total Price:</th>
          <td id="total-price">$0.00</td>
        </tr>
      </tfoot>
    </table>
  </div>

  <script>
    // JavaScript code for shopping cart functionality
```

```
$(document).ready(function() {
    // Function to add product to cart
    function addToCart(productId, productName, price) {
        // Check if the product already exists in the cart
        var existingRow = $('#cart-items tbody tr[data-id="' + productId + '"]');
        if (existingRow.length > 0) {
            // If the product already exists, increment its quantity
            var quantityInput = existingRow.find('.quantity-input');
            var newQuantity = parseInt(quantityInput.val()) + 1;
            quantityInput.val(newQuantity);
            updateRow(existingRow);
        } else {
            // If the product doesn't exist, add a new row to the cart
            var newRow = $('<tr data-id="' + productId + '">' +
                        '<td>' + productName + '</td>' +
                        '<td class="unit-price">$' + price + '</td>' +
                        '<td><input type="number" class="quantity-input" value="1" min="1" style="width:50px;"></td>' +
                        '<td class="overall-price">$' + price + '</td>' +
                    '</tr>');
            $('#cart-items tbody').append(newRow);
        }
        updateTotal();
    }

    // Function to update a row in the cart
    function updateRow(row) {
        var quantity = parseInt(row.find('.quantity-input').val());
        var unitPrice = parseFloat(row.find('.unit-price').text().substring(1)); // Remove '$' and parse as float
        var overallPrice = quantity * unitPrice;
        row.find('.overall-price').text('$' + overallPrice.toFixed(2));
    }

    // Function to update the total price of the cart
```

```
function updateTotal() {
    var totalPrice = 0;
    $('#cart-items tbody tr').each(function() {
        var overallPriceText = $(this).find('.overall-price').text().substring(1); // Remove '$'
        var overallPrice = parseFloat(overallPriceText);
        totalPrice += overallPrice;
    });
    $('#total-price').text('$' + totalPrice.toFixed(2));
}

// Add to Cart button click event listener
$('.btn-add-to-cart').click(function() {
    var productId = $(this).data('id');
    var productName = $(this).data('name');
    var price = $(this).data('price');
    addToCart(productId, productName, price);
});

// Quantity input change event listener
$('#cart-items').on('change', '.quantity-input', function() {
    updateRow($(this).closest('tr'));
    updateTotal();
});

// Function to filter product list based on search input
$('#myInput').on('keyup', function() {
    var searchText = $(this).val().toLowerCase();
    $('table.table-bordered tbody tr').each(function() { // Corrected selector
        var productName = $(this).find('td:nth-child(2)').text().toLowerCase(); // Second column
contains product name
        if (productName.includes(searchText)) {
            $(this).show();
        } else {
            $(this).hide();
```

```
            }
        });
    });
});
</script>
</body>
</html>
```

#new-product.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add New Product</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" th:href="@{/css/style.css}" />
    <style>
        /* Custom CSS for additional styling */
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .container {
            width: 50%; /* Adjust the width of the container */
            padding: 20px; /* Add padding for spacing */
        }
        .heading {
            text-align: center; /* Center align the heading */
            margin-bottom: 20px; /* Add margin bottom for spacing */
        }
        .form-group {
            margin-bottom: 20px; /* Add space between form elements */
```

```
    }
  </style>
</head>
<body>
  <div class="container">
    <h2 class="heading">Add New Product</h2> <!-- Move the heading to the top -->
    <form th:action="@{/shop/new}" th:object="${product}" method="post">
      <div class="form-group">
        <label for="productName">Product Name:</label>
        <input type="text" id="productName" name="productName" th:field="*{productName}"
class="form-control" required />
      </div>
      <div class="form-group">
        <label for="price">Price:</label>
        <input type="text" id="price" name="price" th:field="*{price}" class="form-control" required />
      </div>
      <button type="submit" class="btn btn-primary">Save</button>
      <a href="/shop" class="btn btn-secondary">Cancel</a>
    </form>
  </div>
</body>
</html>
```

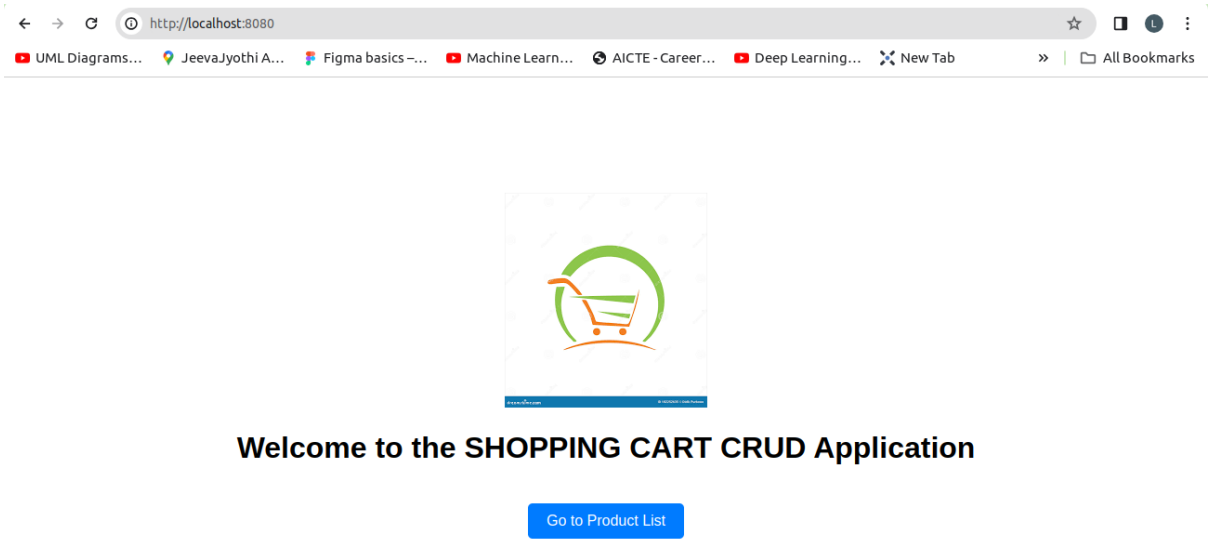
#edit-product.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Edit Product</title>
<link rel="stylesheet" th:href="@{css/style.css}" />
</head>
<body>
<h2>Edit Product</h2>
<form th:action="@{'/shop/edit/' + ${product.id}}" th:object="${product}" method="post"> <!-- Changed
title to Product -->
```

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

<label for="title">ProductName:</label>

<input type="text" id="productName" name="productname" th:field="*{productName}" required /> <!-- Changed th:field to *{productName} -->

<br/>

<label for="price">Price:</label>

<input type="text" id="price" name="price" th:field="*{price}" required /> <!-- Changed id to price -->

<br/>

<button type="submit">Save Changes</button>

</form>

<a href="/shop">Cancel</a>

</body>

</html>
```

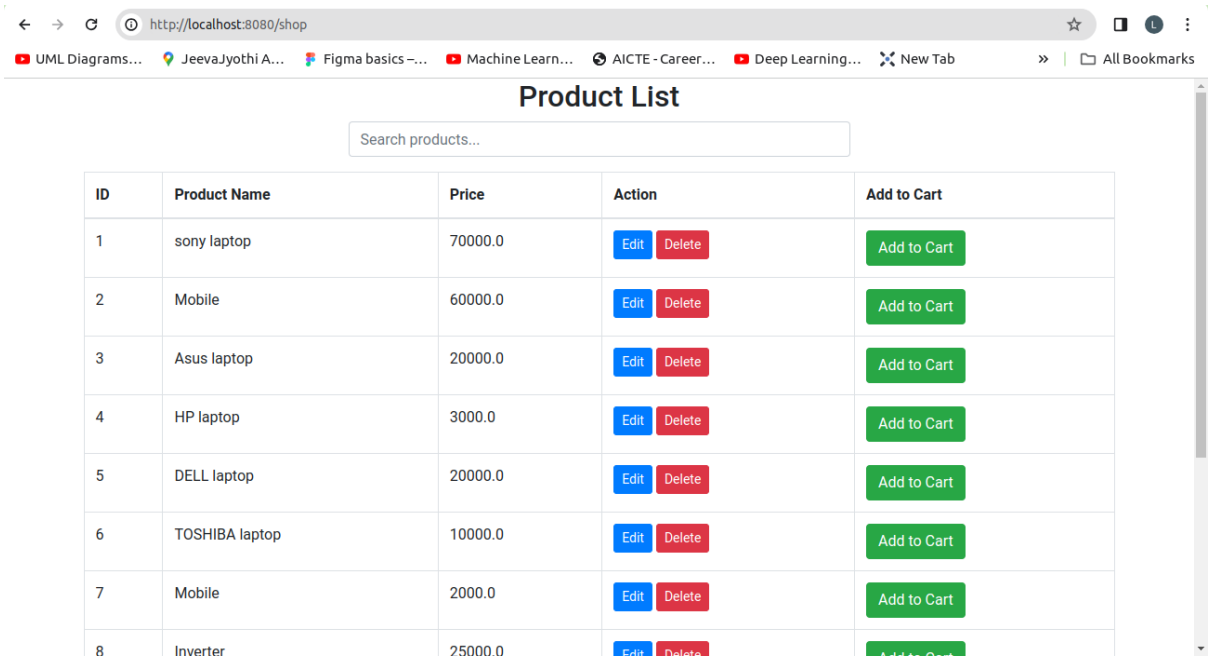
Result and Discussion

-
-
-
-
-
-
-
-
-

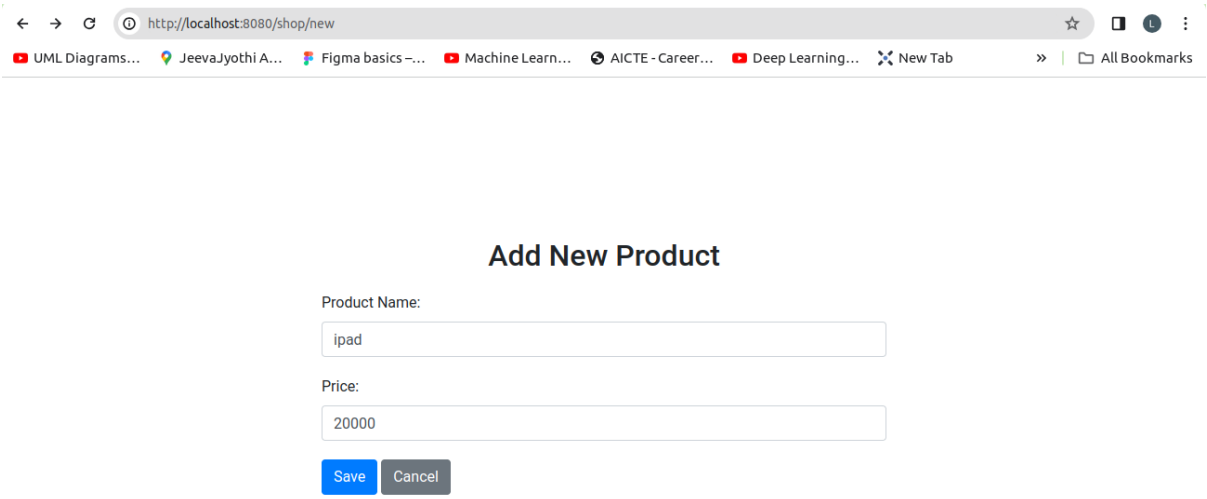


- By clicking the Go to Product List button displays all the book details

- List all the Product details like id, ProductName and price along with action (Edit & Delete)



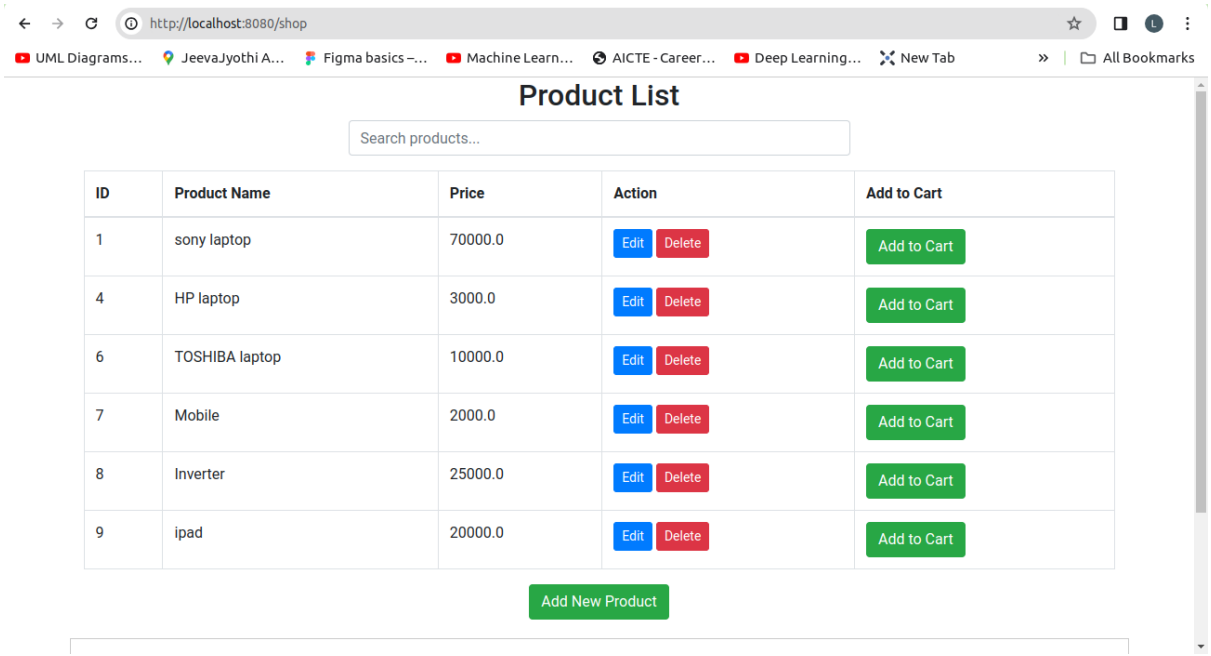
- To add a new product detail by clicking Add New Product button



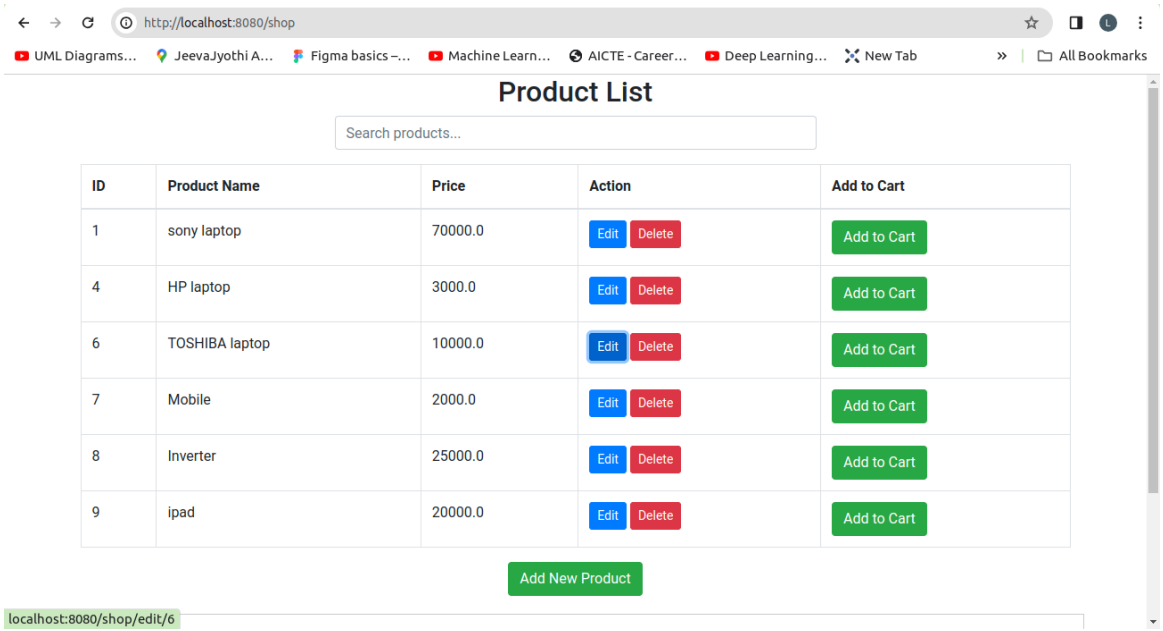
- Then new product details are inserted successfully and it redirects to product-list for displaying result .

- On clicking delete hyperlink, to delete the record from MySQL DB

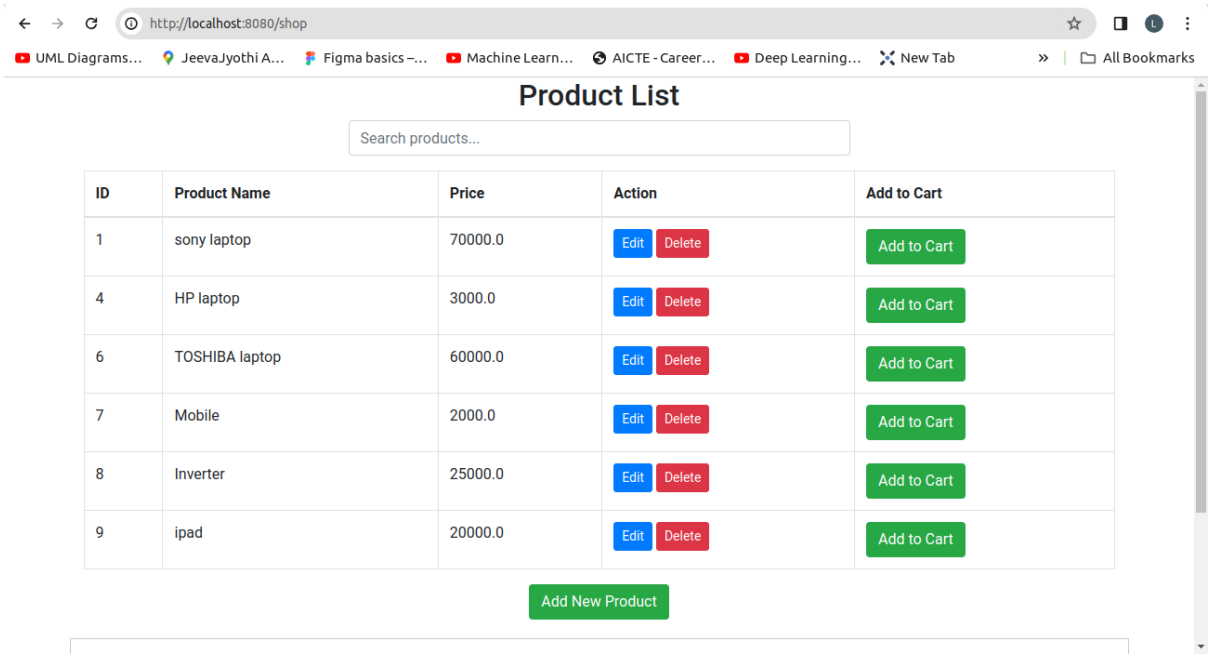
- Then the particular record is deleted successfully



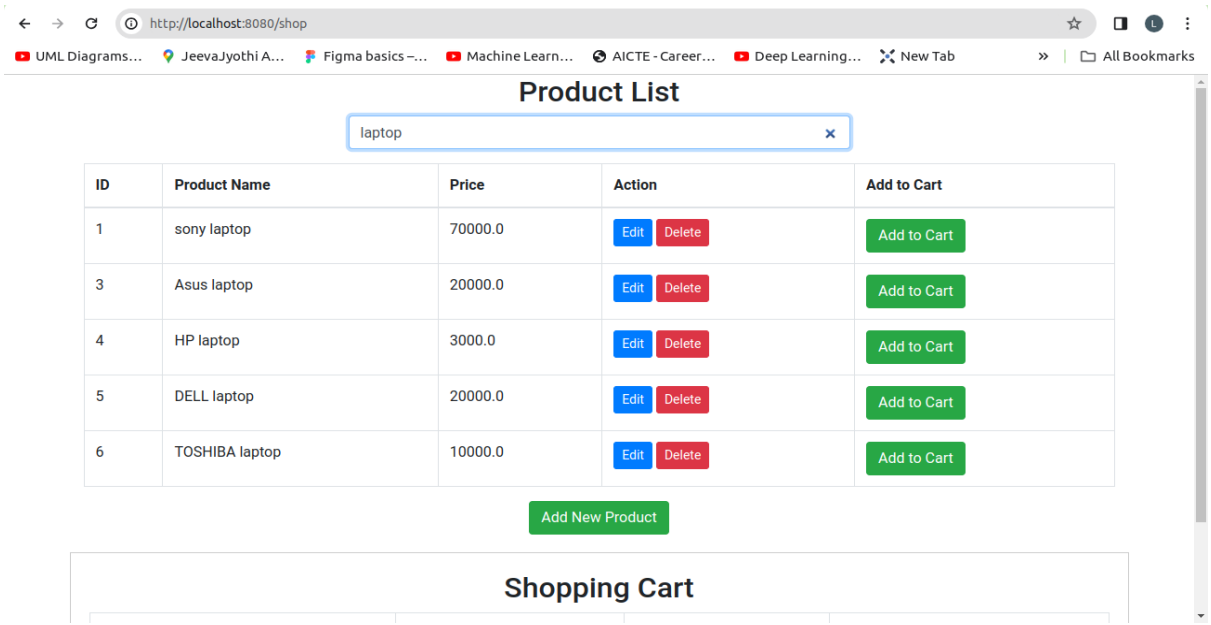
- On clicking edit hyperlink, The content of current record is loaded on edit.html and updating the current record by clicking save button.



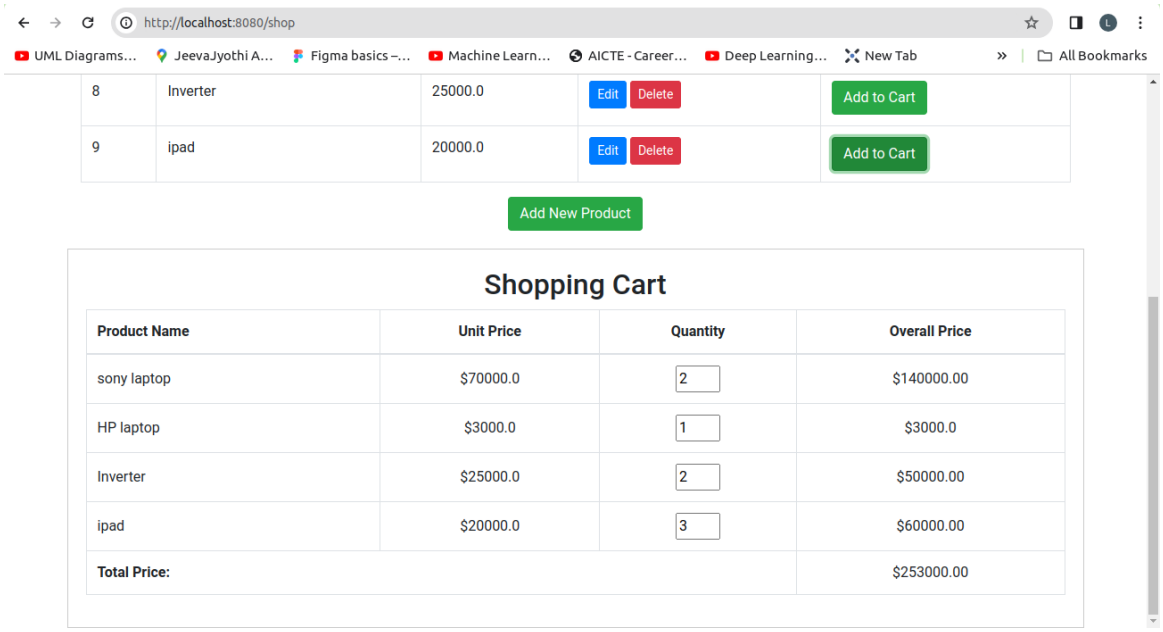
- Then particular record is updated successfully.



- Search Here option to filter the records and display using JQuery



Shopping cart entry is updated


http://localhost:8080/shop
UML Diagrams...
JeevaJyothi A...
Figma basics –...
Machine Learn...
AICTE - Career...
Deep Learning...
New Tab
All Bookmarks
8
Inverter
25000.0
Edit
Delete
Add to Cart
9
ipad
20000.0
Edit
Delete
Add to Cart
Add New Product
Shopping Cart
Product Name
Unit Price
Quantity
Overall Price
sony laptop
$70000.0
2
$140000.00
HP laptop
$3000.0
1
$3000.0
Inverter
$25000.0
2
$50000.00
ipad
$20000.0
3
$60000.00
Total Price:
$253000.00