E-Commerce Website Project Documentation

1. Project Overview

Project Name: E-Commerce Website

Description:

This project is an e-commerce website built using Java Web technology with a Maven-based structure. The website allows users to browse products, register accounts, add items to the cart, and place orders. The admin can manage products, view orders, and monitor user activities.

2. Objectives

- Build a functional e-commerce platform using Java Web technologies.
- Implement user authentication and authorization.
- Provide a seamless shopping experience for users with cart and checkout functionality.
- Enable admins to manage products and monitor transactions efficiently.

3. Technologies Used

- Backend: Java, Servlets, JSP, JDBC
- Frontend: HTML, CSS, Bootstrap, JavaScript (optional frameworks like React/Angular if integrated)
- Database: MySQL
- Build Tool: Maven
- Testing: JUnit
- Server: Apache Tomcat

4. Project Features

❖ User Features:

  - User registration and login.
  - Product browsing and searching.
  - Adding products to the cart.
  - Checkout and order placement.
  - Viewing order history.

❖ Admin Features:

- Adding, updating, and deleting products.
- Viewing all orders and managing them.
- Monitoring user activities.
- Architecture
- Layered Architecture:
- DAO Layer: Interacts with the database.
- Service Layer: Implements business logic.
- Controller Layer: Handles HTTP requests and responses.
- View Layer: Displays data using JSP pages with JSTL and EL.

<u>Project Directory Structure:</u>

```
src/main/java/com/ecommerce

   /controllers  -> Servlets

   /services     -> Business logic

   /dao          -> Data Access Objects
```

```
    /models       -> POJOs

src/main/webapp

    /WEB-INF      -> web.xml, JSP files

    /pages        -> JSP pages (user/admin)

src/test/java/com/ecommerce

    /tests        -> JUnit test cases
```

6. Maven Project Configuration

pom.xml:

Add dependencies for Servlets, JSP, JSTL, MySQL, and JUnit.

```
<dependencies>

    <!-- Servlet API -->

    <dependency>

        <groupId>javax.servlet</groupId>

        <artifactId>javax.servlet-api</artifactId>
```

```xml
    <version>4.0.1</version>

    <scope>provided</scope>

</dependency>



<!-- JSTL -->

<dependency>

    <groupId>javax.servlet</groupId>

    <artifactId>jstl</artifactId>

    <version>1.2</version>

</dependency>



<!-- MySQL Connector -->

<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <version>8.0.34</version>

</dependency>



<!-- JUnit -->
```

```
    <dependency>

        <groupId>junit</groupId>

        <artifactId>junit</artifactId>

        <version>4.13.2</version>

        <scope>test</scope>

    </dependency>

</dependencies>
```

7. Implementation

Database Schema:

Tables:

- Users: Stores user details (id, username, password, email, role).
- Products: Stores product details (id, name, price, description, stock).
- Orders: Stores order details (order_id, user_id, total_amount, date).
- Order_Items: Stores individual items in an order (item_id, order_id, product_id, quantity).

Database Example:

```
CREATE TABLE Users (

    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),

    password VARCHAR(50),

    email VARCHAR(100),

    role VARCHAR(10)

);

CREATE TABLE Products (

    id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100),

    price DECIMAL(10, 2),

    description TEXT,
    stock INT
);

CREATE TABLE Orders (

    order_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    total_amount DECIMAL(10, 2),
    date TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);

CREATE TABLE Order_Items (
    item_id INT AUTO_INCREMENT PRIMARY KEY,

    order_id INT,

    product_id INT,

    quantity INT

);
```

Key Components

1. Servlet Example: ProductServlet.java

Handles product browsing and admin operations.

```
package com.ecommerce.controllers;

import javax.servlet.*;

import javax.servlet.http.*;

import java.io.IOException;

import com.ecommerce.services.ProductService;

public class ProductServlet extends HttpServlet {

 private ProductService productService;



    @Override

    public void init() {
        productService = new ProductService();
    }
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        req.setAttribute("products", productService.getAllProducts());
        req.getRequestDispatcher("/pages/products.jsp").forward(req, resp);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // Handle product addition for admin
    }
}
```

2. Service Example: ProductService.java

```
package com.ecommerce.services;
import com.ecommerce.dao.ProductDAO;
import com.ecommerce.models.Product;
import java.util.List;
```

```
public class ProductService {

    private ProductDAO productDAO;

    public ProductService() {

        productDAO = new ProductDAO();

    }

    public List<Product> getAllProducts() {

        return productDAO.getAllProducts();

    }

}
```

3. JSP Example: products.jsp

Displays a list of products.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

<head>

    <title>Products</title>

</head>

<body>

    <h2>Product List</h2>
    <ul>

        <c:forEach var="product" items="${products}">

            <li>${product.name} - $${product.price}</li>

        </c:forEach>

    </ul>

</body>

</html>
```

8. Testing

- Unit Testing: Test DAO and Service layers using JUnit.
- Integration Testing: Test interaction between servlets and JSPs.
- Manual Testing: Verify user registration, login, and checkout processes.

Example JUnit Test:

```
@Test

public void testAddProduct() {

    ProductDAO dao = new ProductDAO();

    Product product = new Product("Laptop", 1000.00, "High-end laptop", 10);

    assertTrue(dao.saveProduct(product));

}
```

9. Deployment

- Build the project using Maven: mvn clean install
- Deploy the WAR file to Tomcat: Place ecommerce.war in the webapps folder.
- Access the application at: http://localhost:8080/ecommerce

10. Conclusion

This e-commerce platform provides a foundation for an online store with robust functionality for both users and admins. Future enhancements can include:

- Payment gateway integration.
- Advanced search and filtering.
- RESTful APIs for mobile applications.