INDIVIDUAL ASSIGNMENT

LEVEL 6

HF24A1COM/SE

CLEAN CODING AND CONCURRENT PROGRAMMING 02

Name:Irosha Rajapaksha

CB011366

Hand Out Date:24/03/2025
Hand In Date: 25/05/2025

Contents

1. Introduction 5
2. GUI Development 6
2.1. Complete Use Case Implementation 6
2.1.1. Admin Dashboard 6
2.1.2. Cashier Dashboard 15
2.2. Standard Web Features Integration 19
2.3 Enhanced User Experience Features 24
3. Server-side Concurrency Architecture 27
3.1 Single Client Request Handling 27
3.2 Multiple Client Sequential Processing 28
3.3 Simultaneous Multi-Client Support 28
3.4 High-Volume Request Queuing System 29
4. Client-side Concurrency Implementation 32
4.1 Basic Client-Server Communication 32
4.2 Multi-Computer Client Interaction 32
4.3 Real-time Data Synchronization 33
4.4 Asynchronous Test Client Development 34
..... 35
4.4.1. GET Request 35
4.4.2. POST Request 38
5. Clean Architecture Design 40
5.1 Multi-tier Architecture Preservation 40
5.2. Folder Structure 41
5.3. Implementation Discussion 43
5.3.1.Online and Physical Stock Management 43
5.3.2. Java Servlet Architecture Implementation Discussion 43
5.3.3. Java Server Pages 44
5.3.4. Testing 46
6. Critical Analysis and Discussion 52
6.1. Strengths in Clean Architecture Implementation 52
6.2. Concurrency Architecture Analysis 52
6.3. User Experience and Interface Design 52

6.3.1. GUI Implementation Assessment 52
6.4. Technical Implementation Critique 53
6.4.1. Database and Performance Considerations 53
6.4.2. Security and Data Integrity 53
6.5. Testing and Quality Assurance 54
JMeter Testing Analysis 54
7.Conclusion 55

Figure 1 : Admin Dashboard..... 6
Figure 2 : Products Management 7
Figure 3 : Add New Product (Admin) 7
Figure 4 : Categories Management 8
Figure 5 : Add New Category (Admin) 8
Figure 6 : Stock Management 9
Figure 7 : Add New Stock 9
Figure 8 : Shelf Management..... 10
Figure 9 : Add to shelf (Admin) 10
Figure 10 : Order Management..... 11
Figure 11 : Customers Management 12
Figure 12 : add New Customer 12
Figure 13 : Suppliers Management 13
Figure 14 : Add new supplier..... 14
Figure 15 : Cashier Dashboard 15
Figure 16 : Cashier Generate Bill 17
Figure 17 : View All Bills..... 18
Figure 18 : Bill 18
Figure 19 : Home page..... 19
Figure 20 : Product List page..... 20
Figure 21 : Categories List Page 20
Figure 22 : Product View..... 21
Figure 23 : Register Page 21
Figure 24 : Login Page..... 22
Figure 25 : Checkout Page 22
Figure 26 : Thank You Page 23
Figure 27 : Cart Page 23
Figure 28 : Validation 24
Figure 29 : Real-time alerts Notification 25
Figure 30 : Interactive metric cards display..... 26
Figure 31 : Login Servlet request processing flow diagram 27
Figure 32 : Multiple client sequential processing architecture 28

Figure 33 : server.xml 29
Figure 34 : Concurrent multi-client processing with thread pool management 29
Figure 35 : Request queuing system with overflow protection 30
Figure 36 : implementation of DBCP 31
Figure 37 : Client-Server Synchronous Communication (Add to Cart Form Submission) 32
Figure 38 : Multi-computer client coordination and session management 33
Figure 39 : Real-time data synchronization patterns 34
Figure 40 : Jmeter GET , POST Request 35
Figure 41 : Jmeter /cart (get request) 35
Figure 42 : Jmeter /category (get request) 36
Figure 43 : Jmeter /home (get request) 36
Figure 44 : Jmeter /login (get request) 36
Figure 45 : Jmeter /register (get request) 37
Figure 46 : Jmeter /logout (get request) 37
Figure 47 : Jmeter /orderConfirmation (get request) 37
Figure 48 :Jmeter /orders (get request) 38
Figure 49 : Jmeter /products (get request) 38
Figure 50 : Jmeter /cart (post request) 38
Figure 51 : Jmeter /login (post request) 39
Figure 52 : Jmeter /register (post request) 39
Figure 53 : Clean Architecture preservation across distributed tiers 40
Figure 54 : Folder structure 42
Figure 55 : Online and Physical Stock Management Diagram 43
Figure 56 : Login Servlet Code 44
Figure 57 : base layout jsp 45
Figure 58 : welcome jsp 45
Figure 59 : AdminServletTest 46
Figure 60 : CartServletTest 46
Figure 61 : CashierDashboardTest 47
Figure 62 : CategoriesListServletTest 47
Figure 63 : CheckoutServletTest 48
Figure 64 : HomeServletTest 48
Figure 65 : LoginServletTest 49
Figure 66 : OrderConfirmationServletTest 49
Figure 67 : OrdersServletTest 50
Figure 68 : ProductDetailsServletTest 50
Figure 69 : ProductListServletTest 51
Figure 70 : RegisterServletTest 51

1. Introduction

In this report, I detail how SYOS (Synex Outlet Store) Point of Sale, originally developed for consoles, was successfully rebuilt as a multi-user, multi-tier, web application. Using still the Clean Architecture, the system now acts as a full e-commerce platform that supports many users at the same time through advanced parallel processing. The new software is impressive because it showcases excellence in GUI design, handling concurrent work on the server, handling asynchronous functions on the client, designing a distributed system and validating extensive load testing.

2. GUI Development

2.1. Complete Use Case Implementation

Admin, Cashier and Online Customer each have their own dashboard through the interface, based on what they do. All the product management, category management, stock management, shelves management, order management, customer management, supplier management, billing and reporting functions can be used with ease through web forms and interactive displays.

The GUI holds onto the original system's full functionality but adds additional features for the web, including thorough form validation and responsive design. Users are able to carry out all actions such as product creation, view, updating and deletion, handling orders, processing sales, controlling inventory, managing shelves and categories, dealing with users, creating reports, billing, customers managing and managing suppliers.

2.1.1. Admin Dashboard

- Admin - can manage Products, Orders, Stocks, Shelves, Categories, Users, Reports, Customers, Suppliers
- View - /admin/dashboard

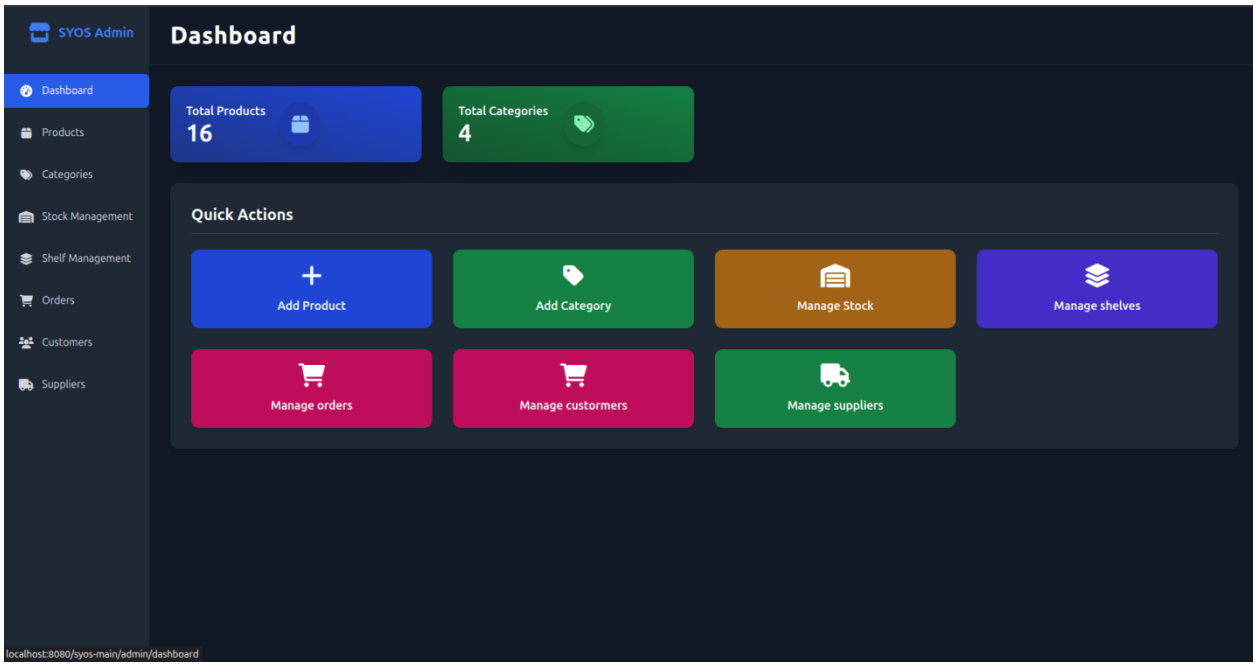


Figure 1 : Admin Dashboard

- Manage Products – Admin can add new Products ,Edit Existing products, Delete Products, View Products
- View - /admin/products

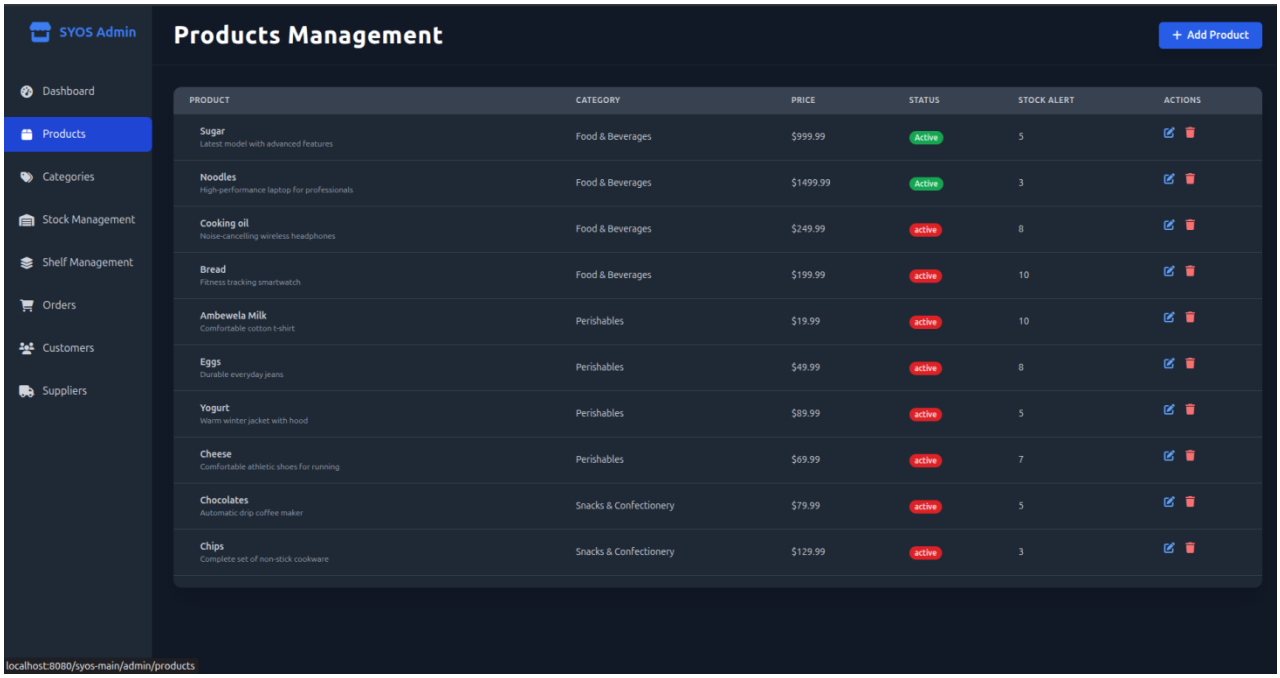


Figure 2 : Products Management

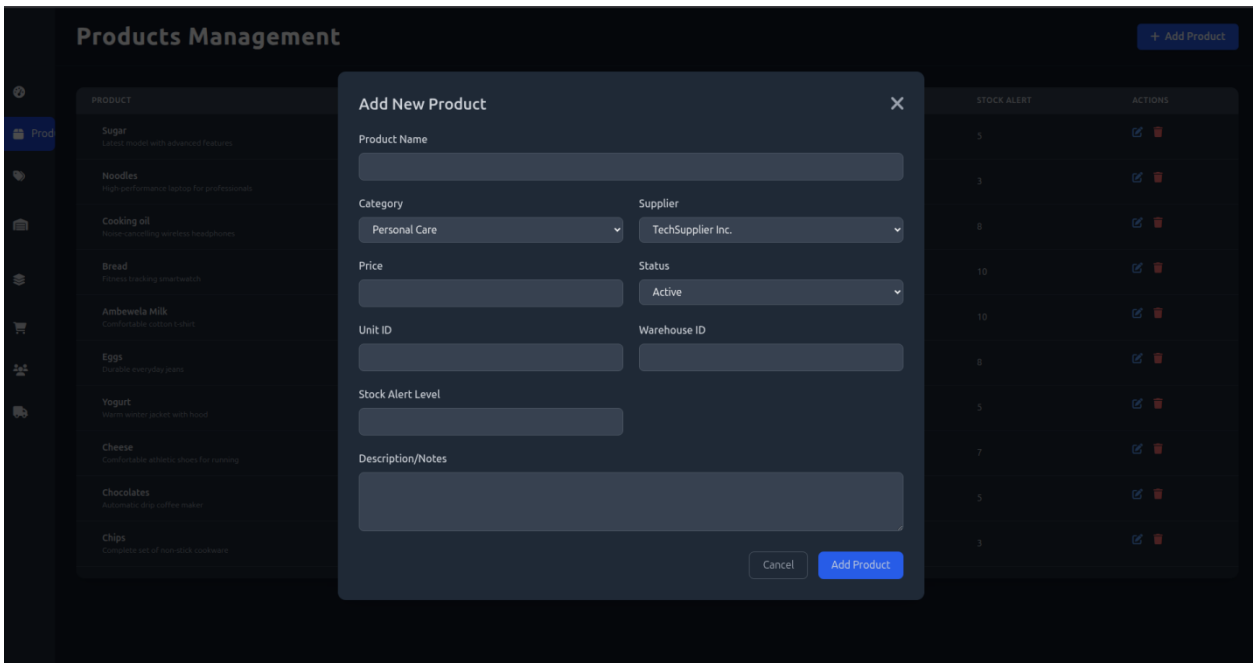


Figure 3 : Add New Product (Admin)

- Manage Categories– Admin can add new Category, Edit Existing Category, Delete Category, View Category

- View - /admin/categories

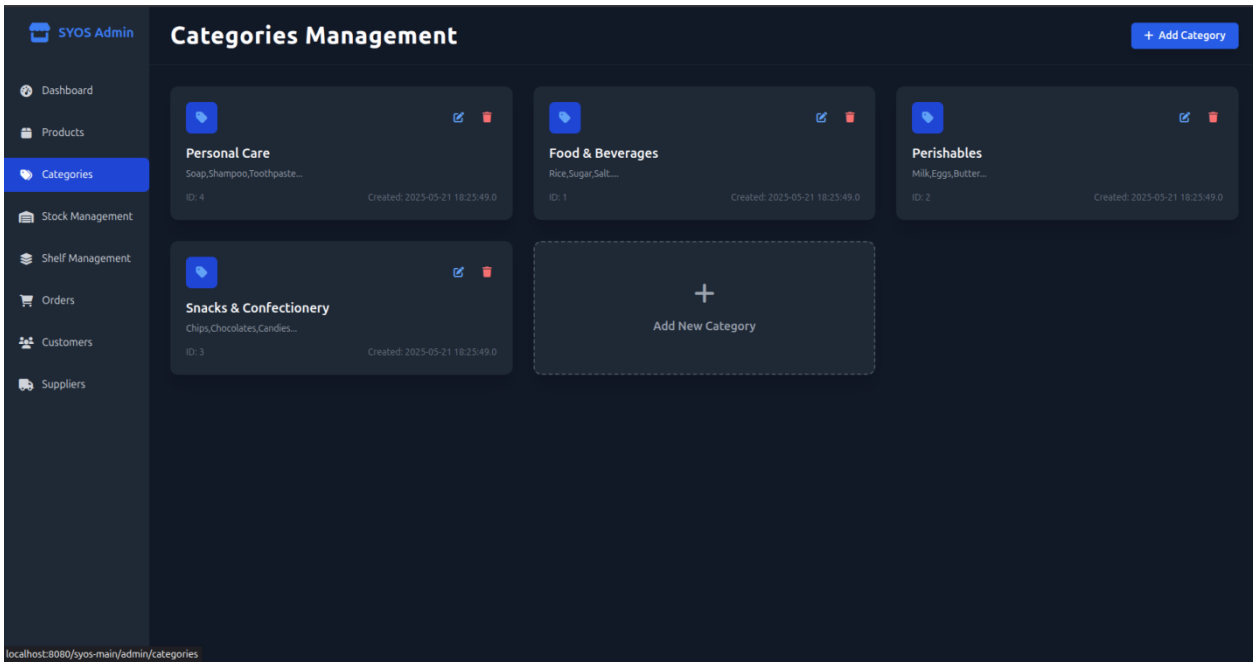


Figure 4 : Categories Management

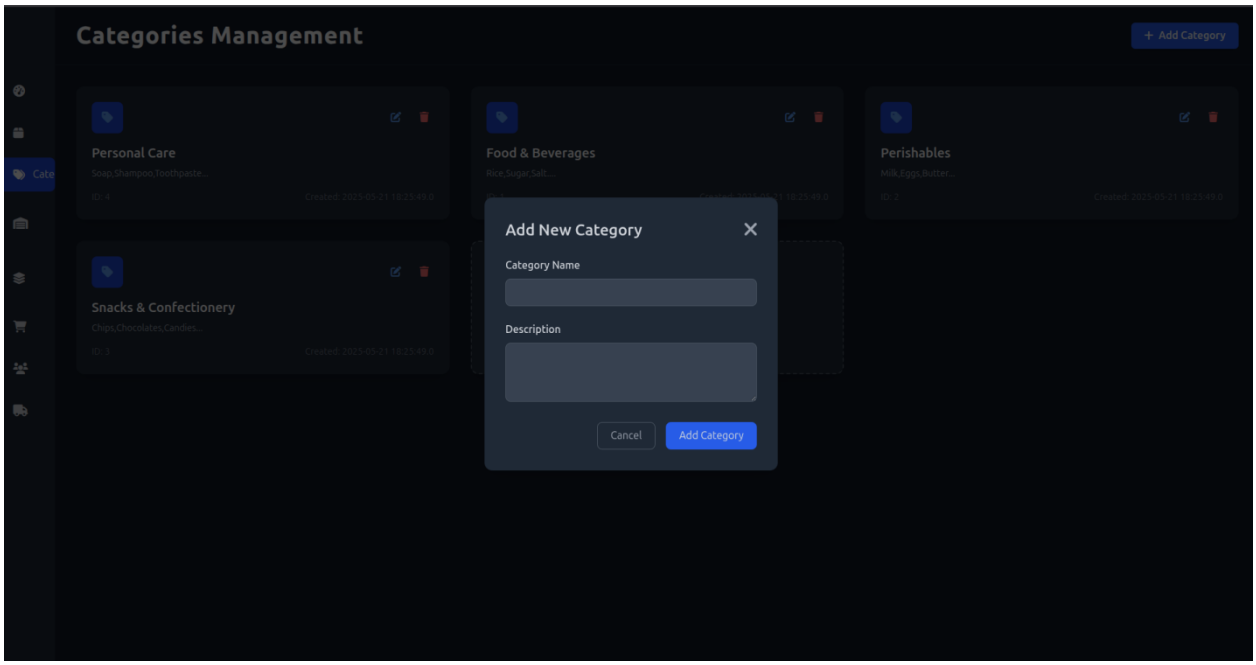


Figure 5 : Add New Category (Admin)

- Manage Stocks– Admin can add new Stocks, Edit Existing Stocks, Delete Stocks, View Stocks
- View - /admin/stocks

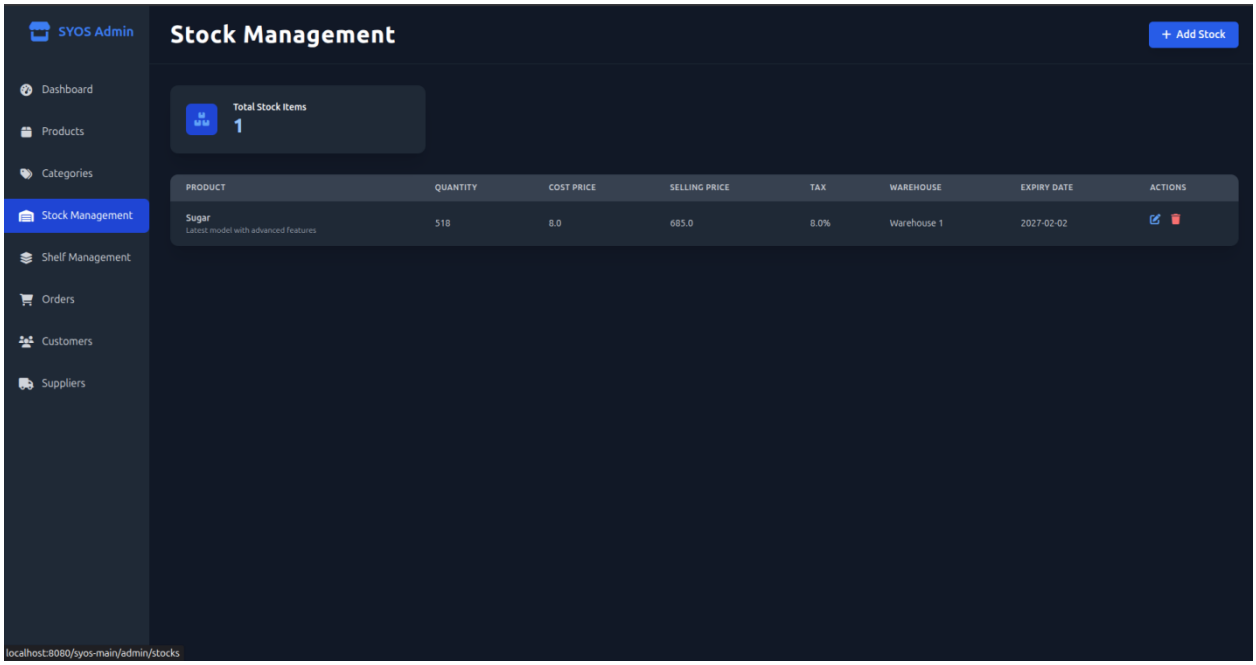


Figure 6 : Stock Management

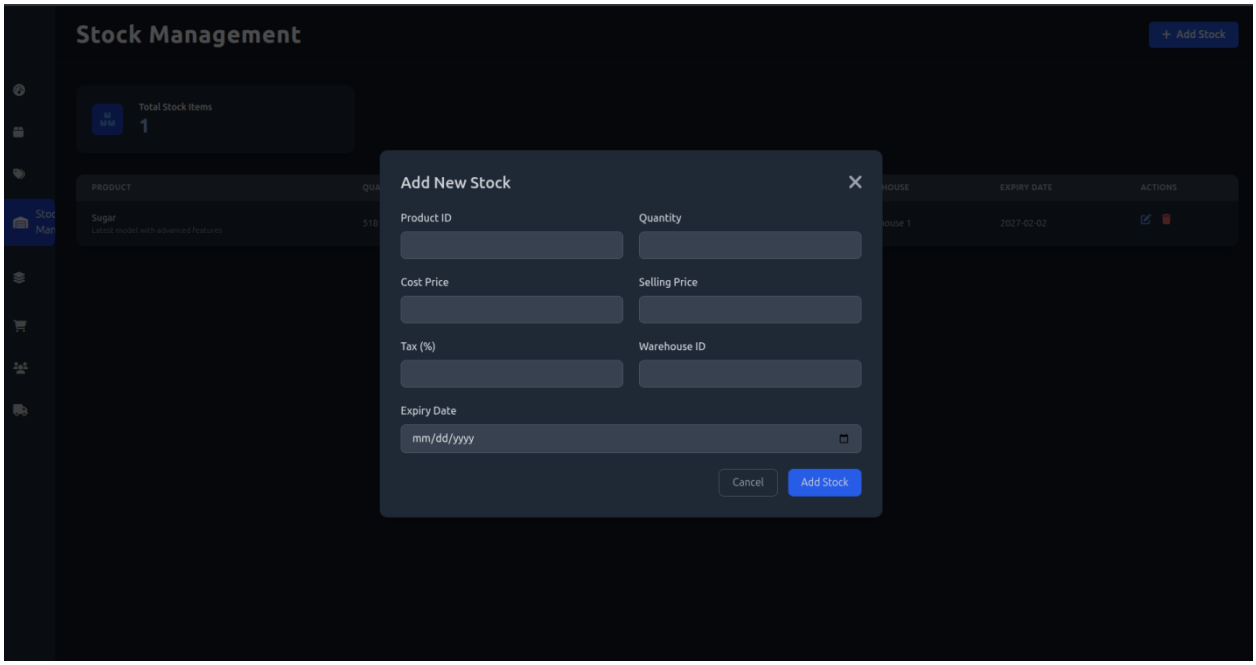


Figure 7 : Add New Stock

- Manage Shelf– Admin can add new Shelf, Edit Existing Shelf, Delete Shelf, View Shelf
- View - /admin/shelves

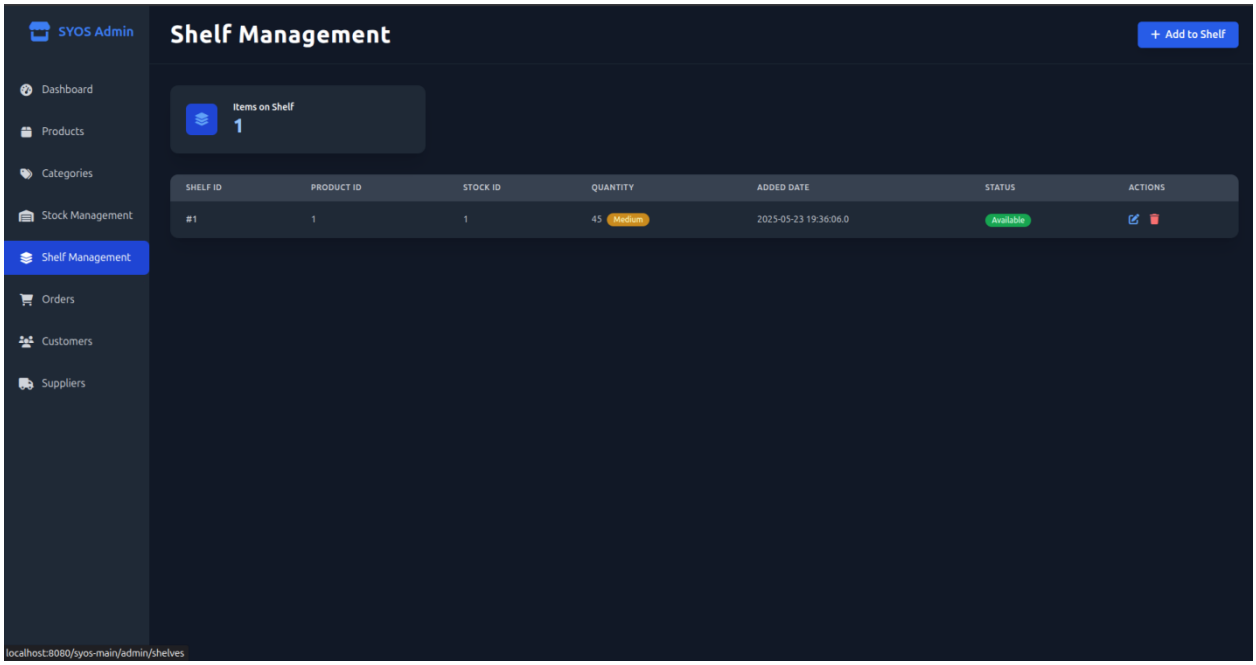


Figure 8 : Shelf Management

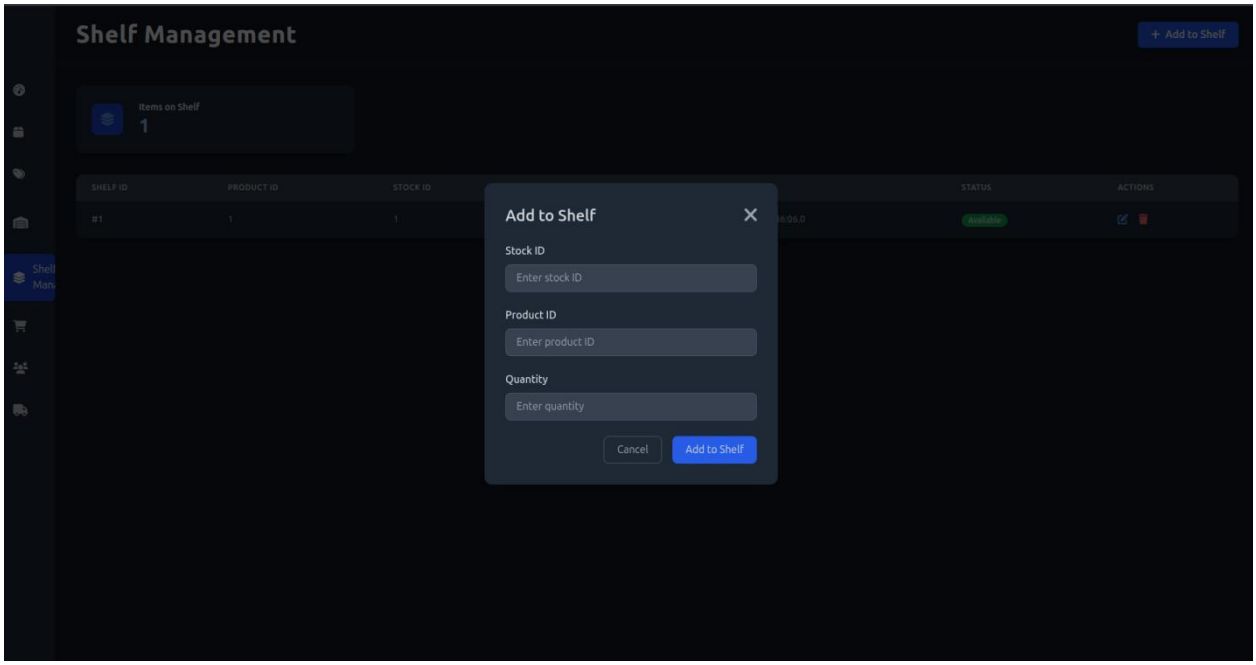


Figure 9 : Add to shelf (Admin)

- Manage Orders– Admin can View orders, and also can change order status.
- View - /admin/orders

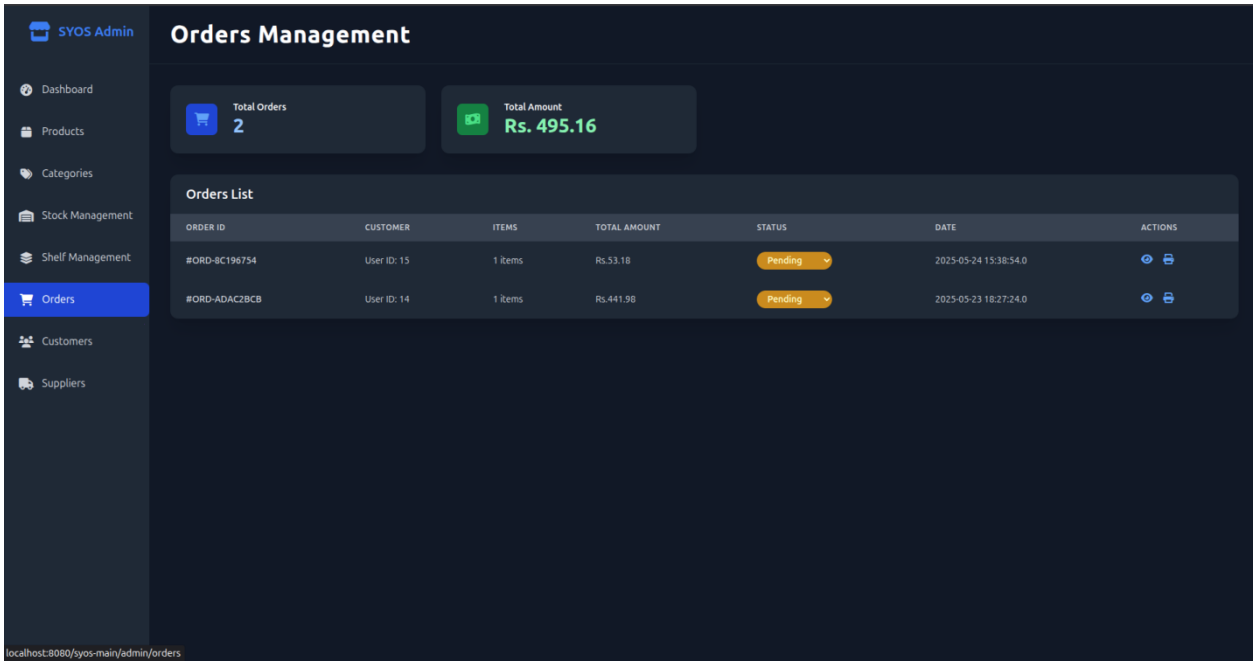


Figure 10 : Order Management

- Manage Customers– Admin can add new Customer, Edit Existing Customer, Delete Customer, View Customer
- View - /admin/customers

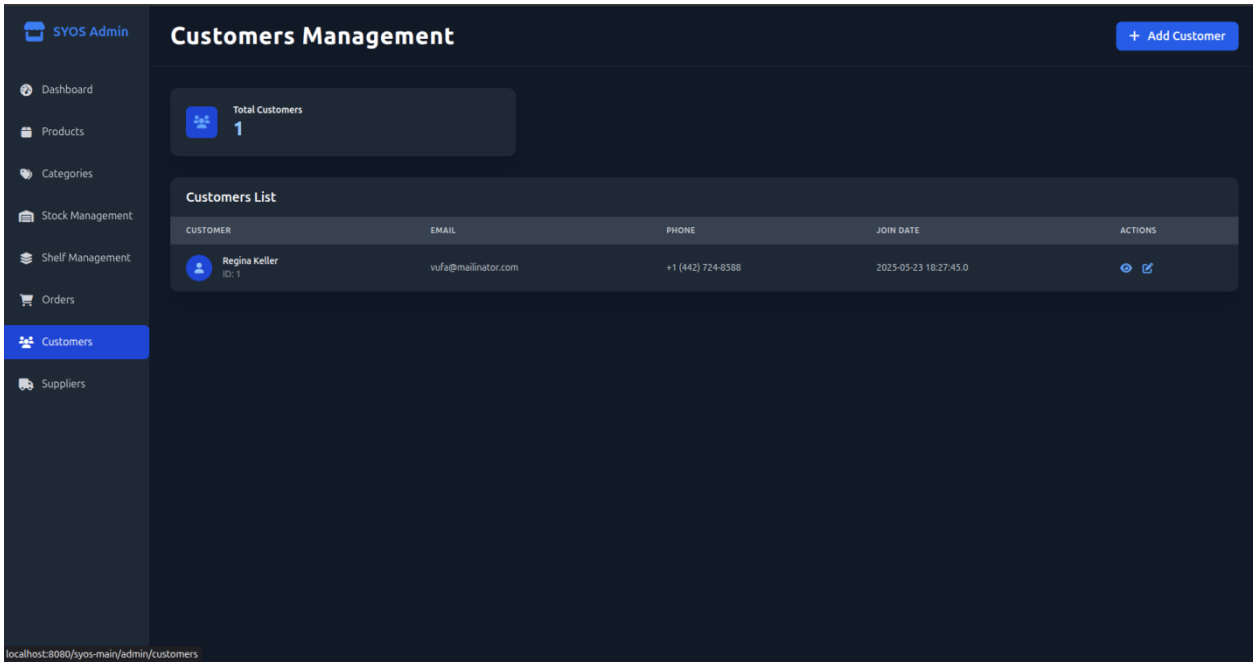


Figure 11 : Customers Management

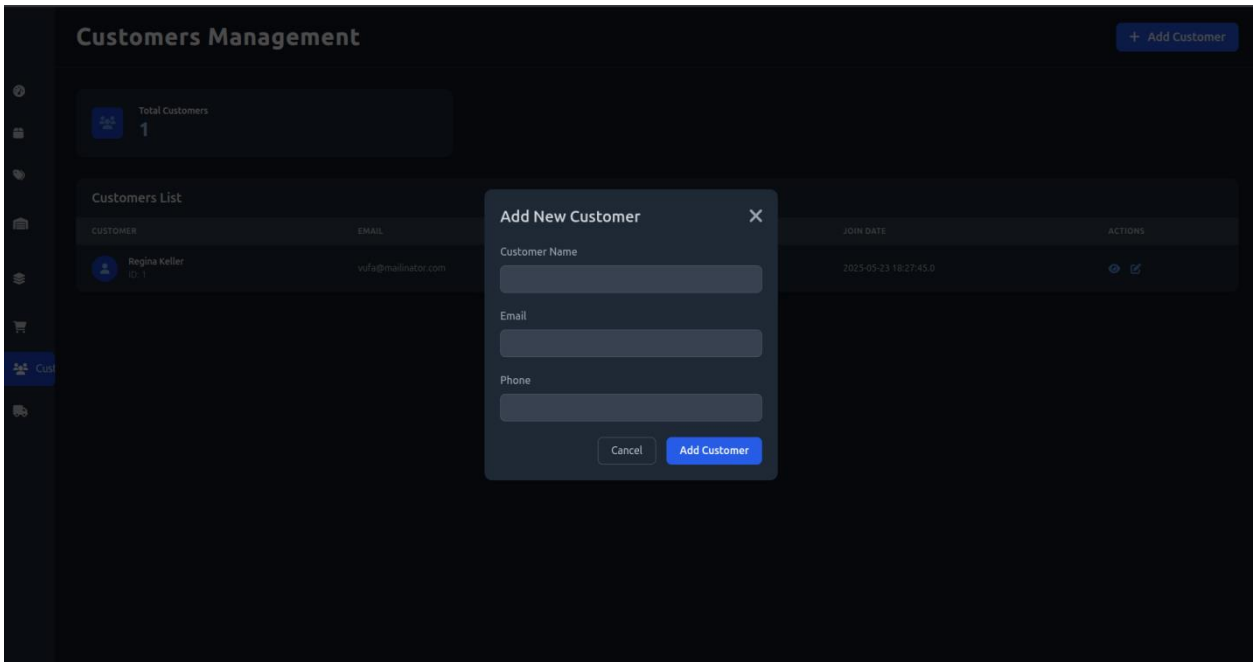


Figure 12 : add New Customer

- Manage Suppliers– Admin can add new Suppliers, Edit Existing Suppliers, Delete Suppliers, View Suppliers
- View - /admin/suppliers

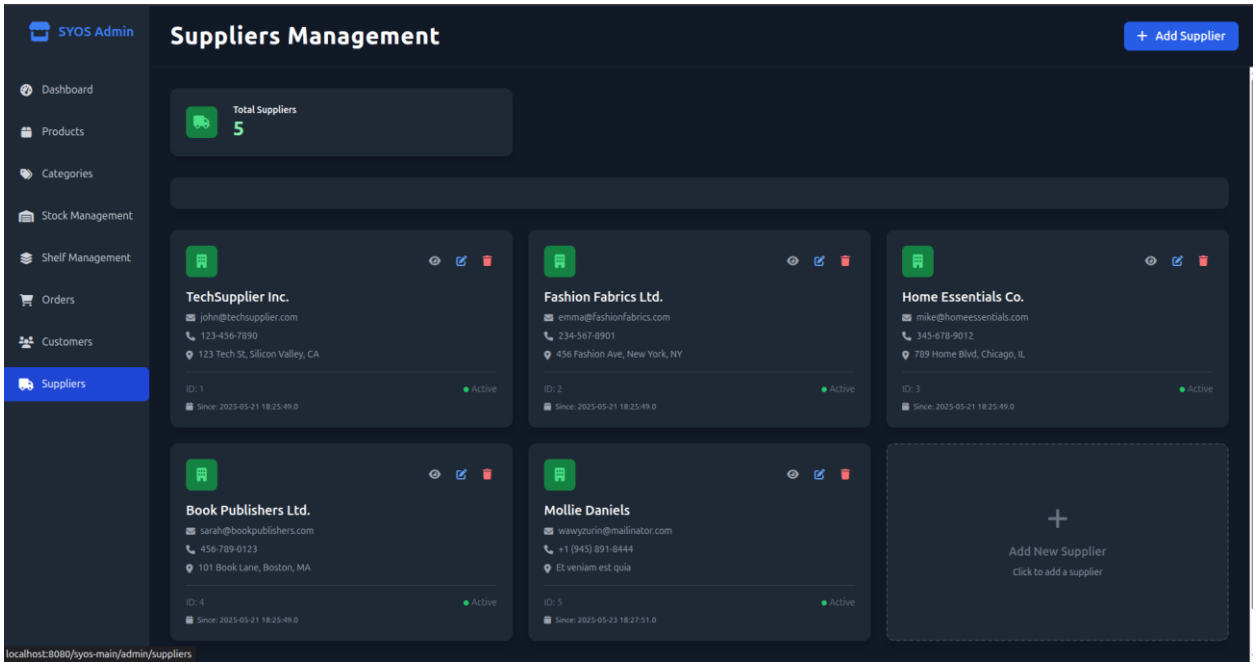
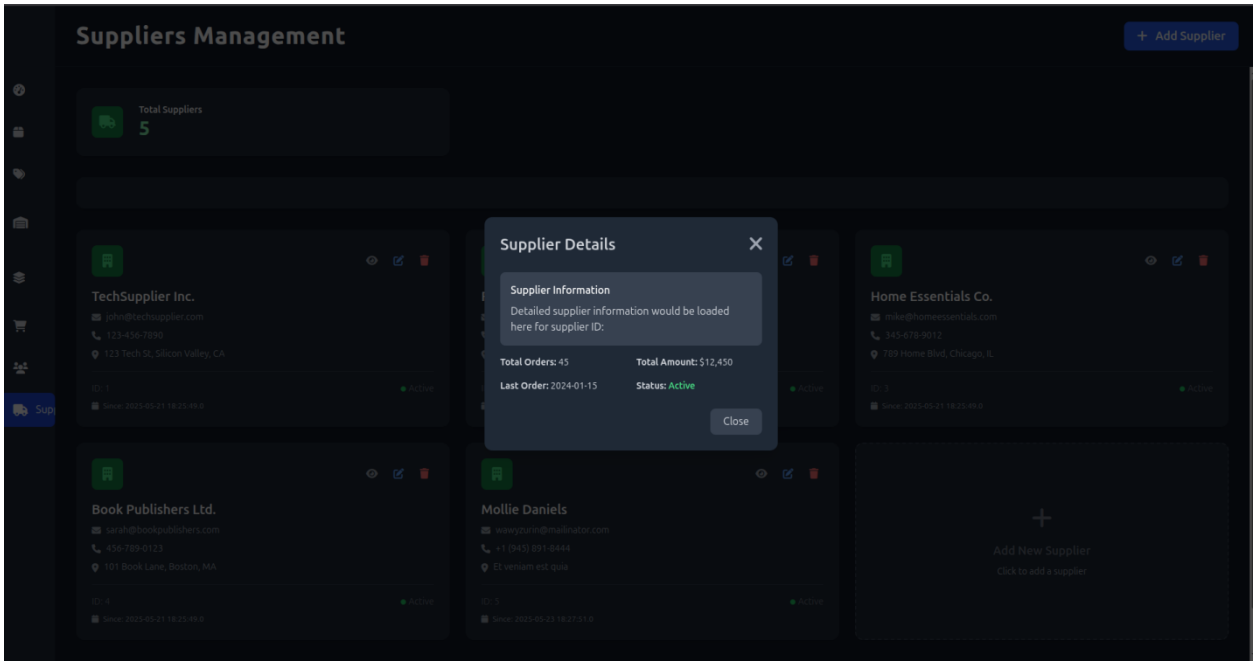


Figure 13 : Suppliers Management



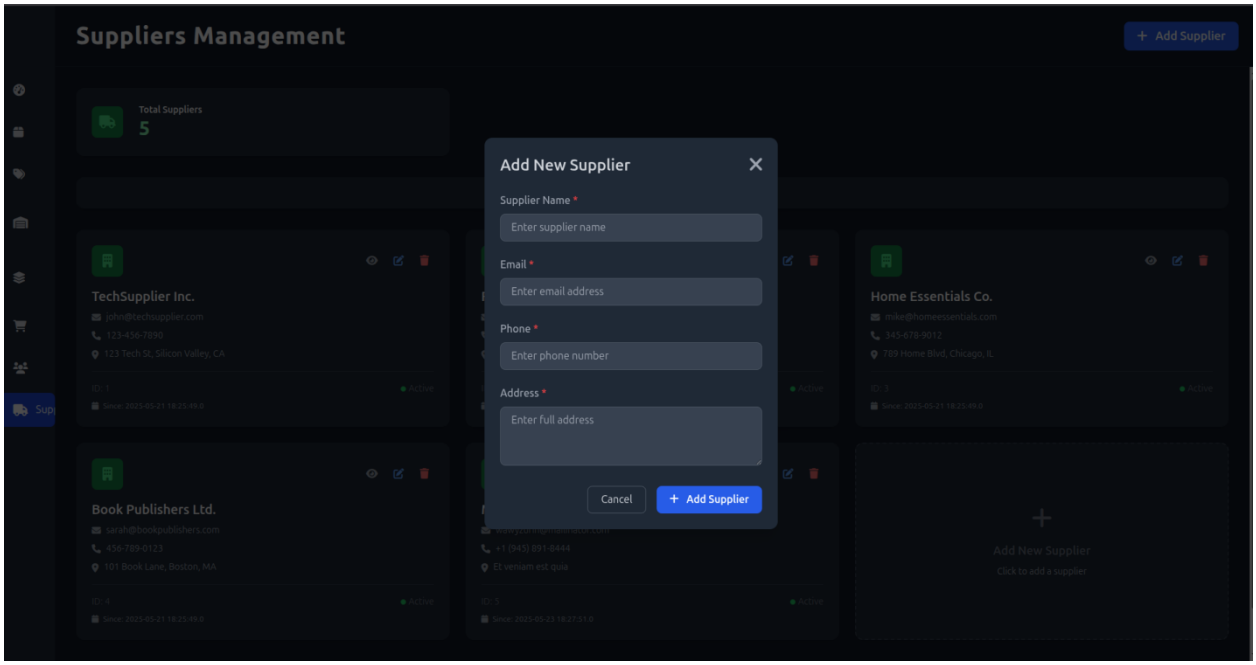


Figure 14 : Add new supplier

2.1.2. Cashier Dashboard

- This is Cashier dashboard. Cashier can Generate bills , can view all bills , and also can print bills .

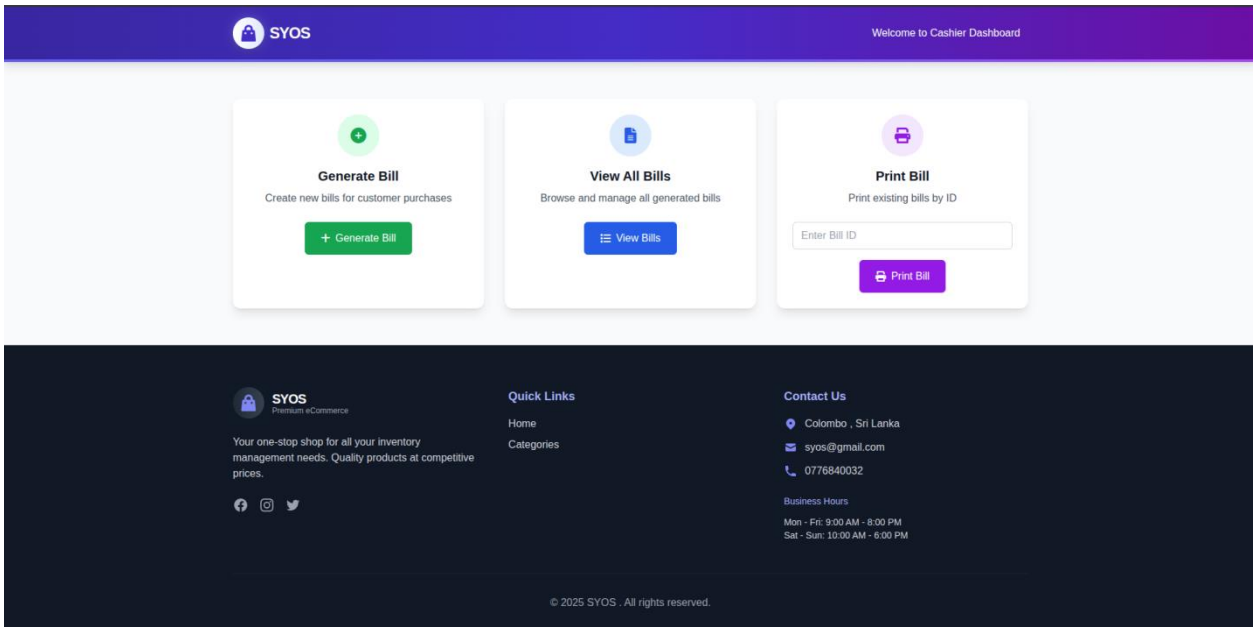


Figure 15 : Cashier Dashboard

- Customer Management
    - Customer ID field (optional) for tracking repeat customers
    - Option to leave blank for walk-in customers
- Product Management
    - Product selection dropdown menu
    - Quantity input field
    - Price per unit display
    - Total calculation per line item
    - "Add Product" button to include multiple items
    - Delete option (trash icon) to remove items
- Pricing & Calculations
    - Subtotal - Base amount before adjustments
    - Discount System
        - Discount type selector (Fixed , Percentage)
        - Discount value input field

- Tax Management
    - Tax type dropdown (Fixed , Percentage)
    - Tax value input field
- Final Total - Automatically calculated final amount
- Payment Processing
    - Received Amount field for cash transactions
    - Change Return calculation (automatically computed)
    - Payment Status dropdown (currently set to "Paid")
- Transaction Management
    - Cancel option to abort the transaction
    - "Save Bill" to record the transaction
    - "Save & Print" to generate receipt

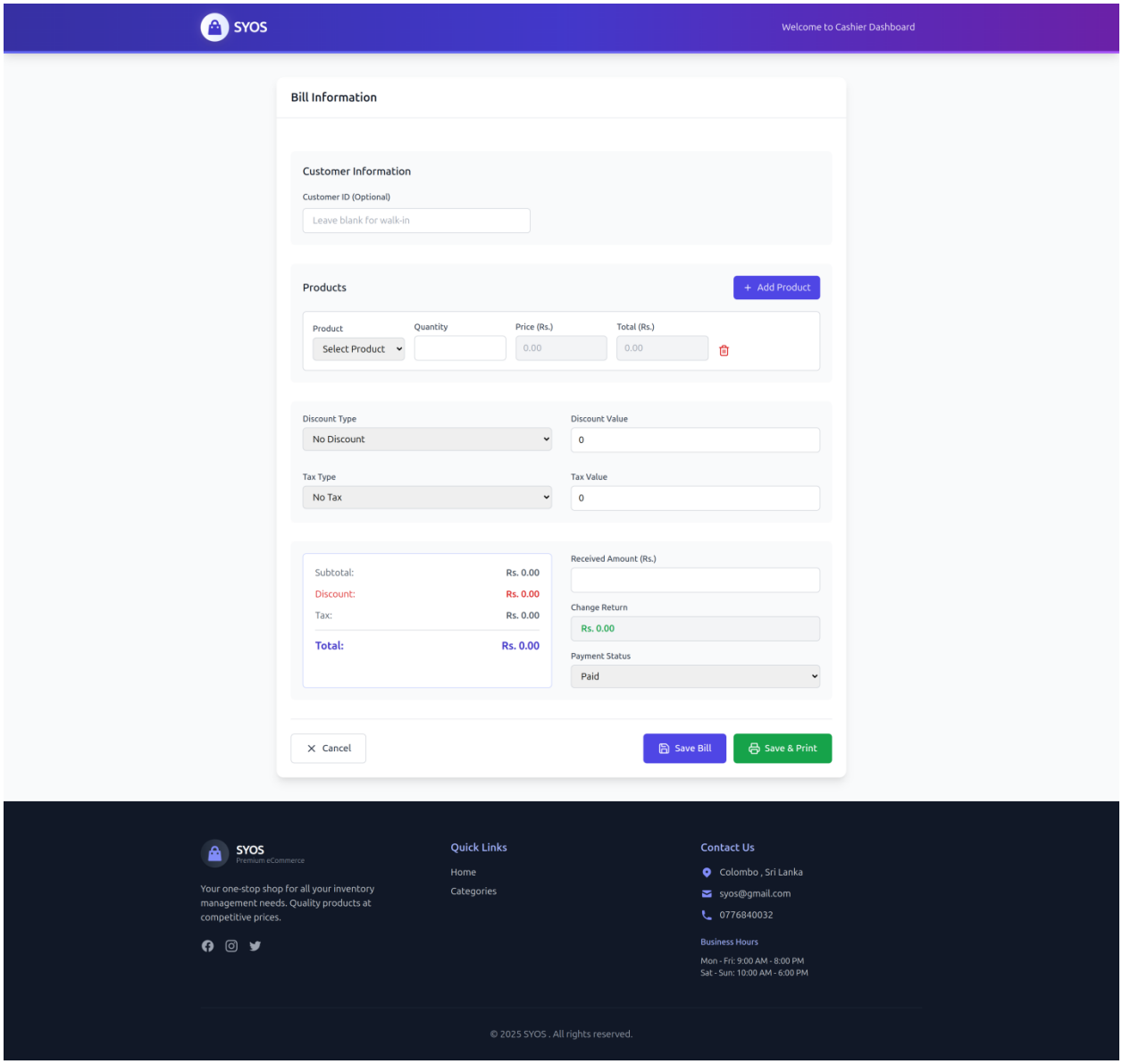


Figure 16 : Cashier Generate Bill

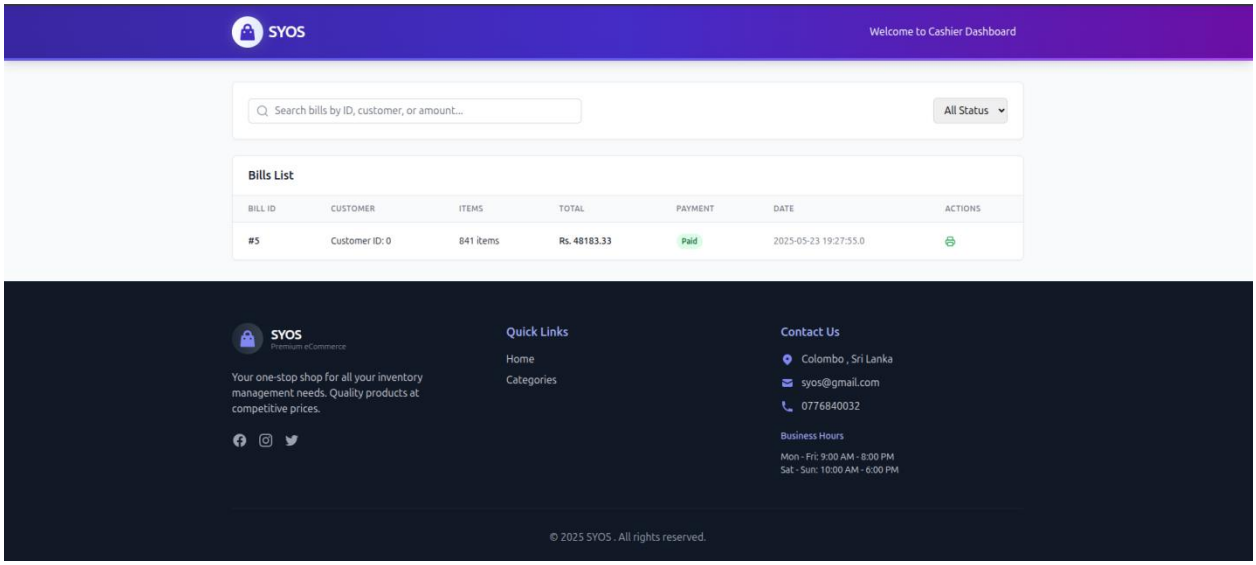


Figure 17 : View All Bills

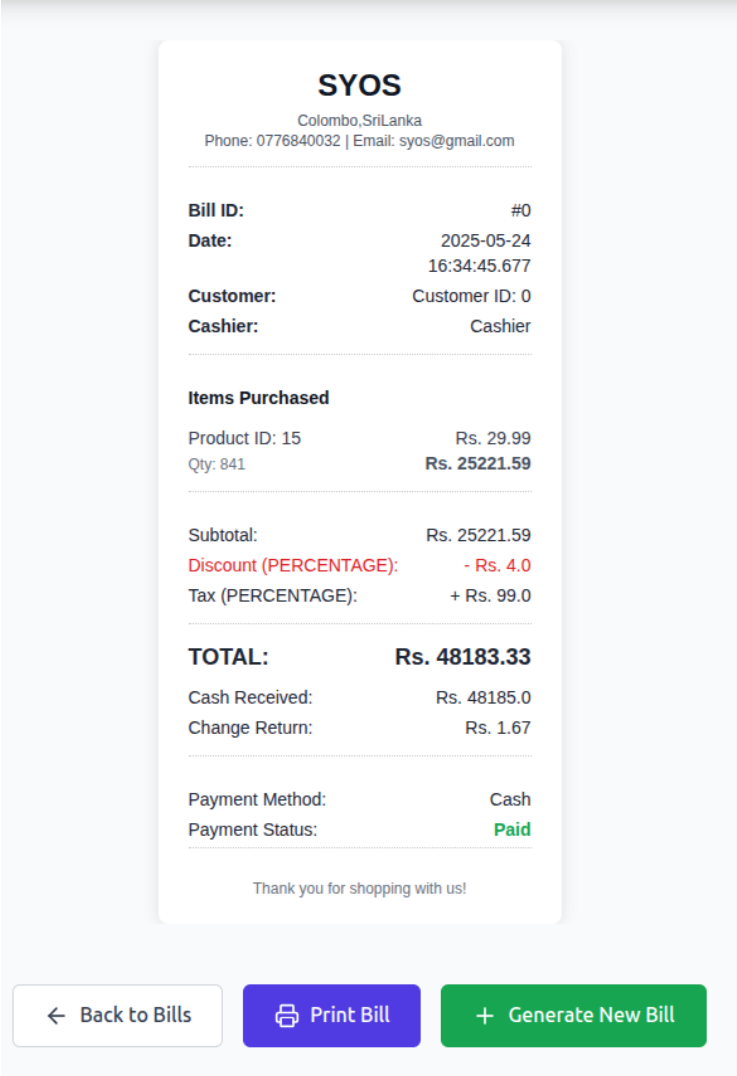


Figure 18 : Bill

2.2. Standard Web Features Integration

It includes all features found in web development today such as responsive CSS, JavaScript form validation, AJAX-based asynchronous communication, session management and secure authentication, with taiwindcss for user interface design. The way it looks changes automatically based on the device you use, so it’s useful on desktop computers, tablets and mobile phones.

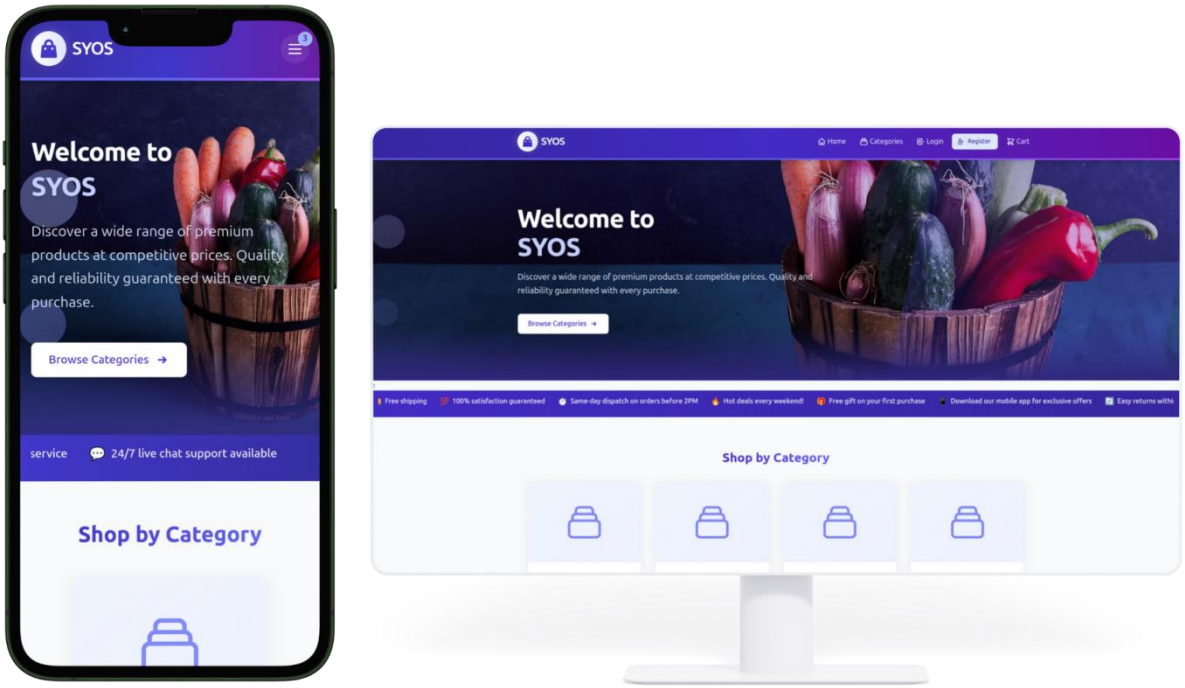


Figure 19 : Home page

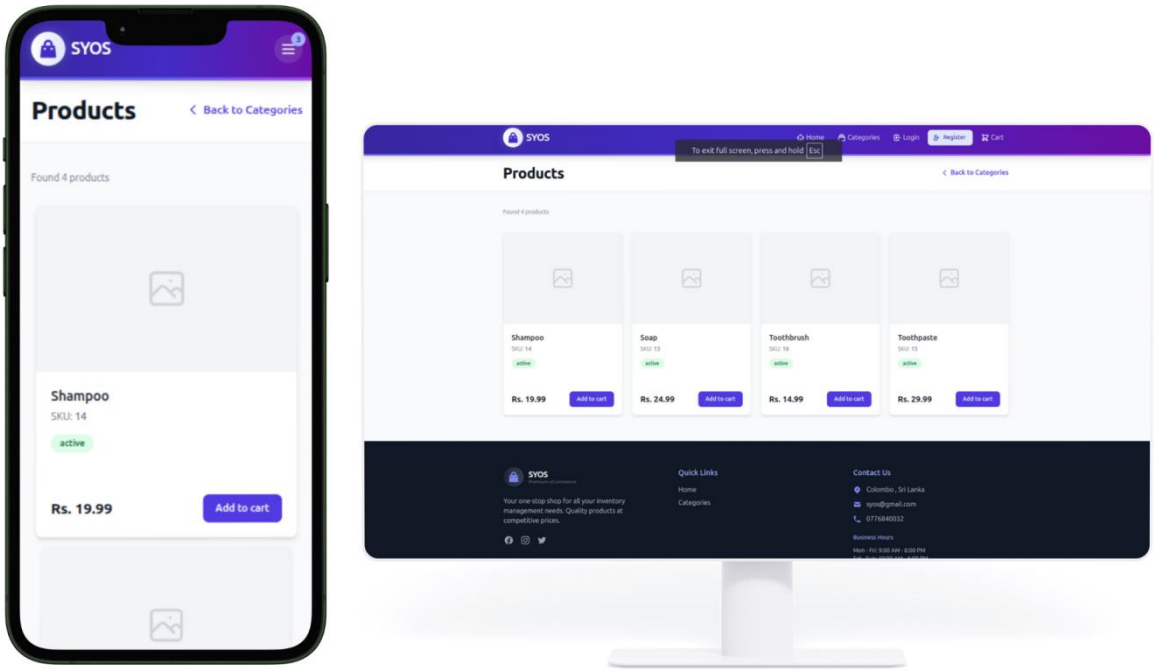


Figure 20 : Product List page

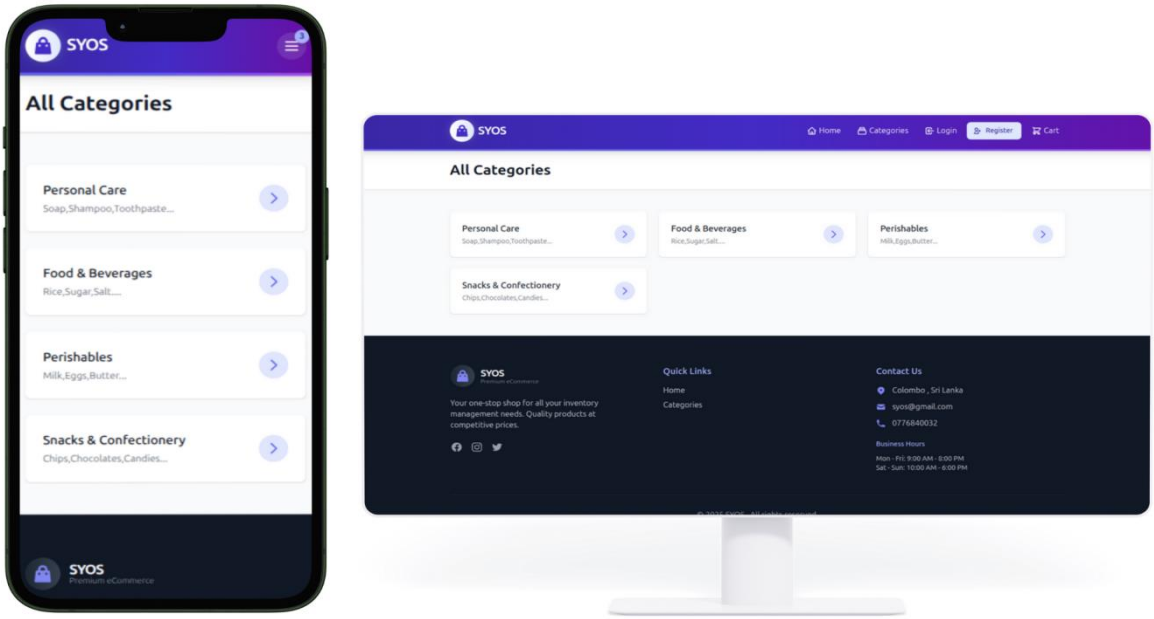


Figure 21 : Categories List Page

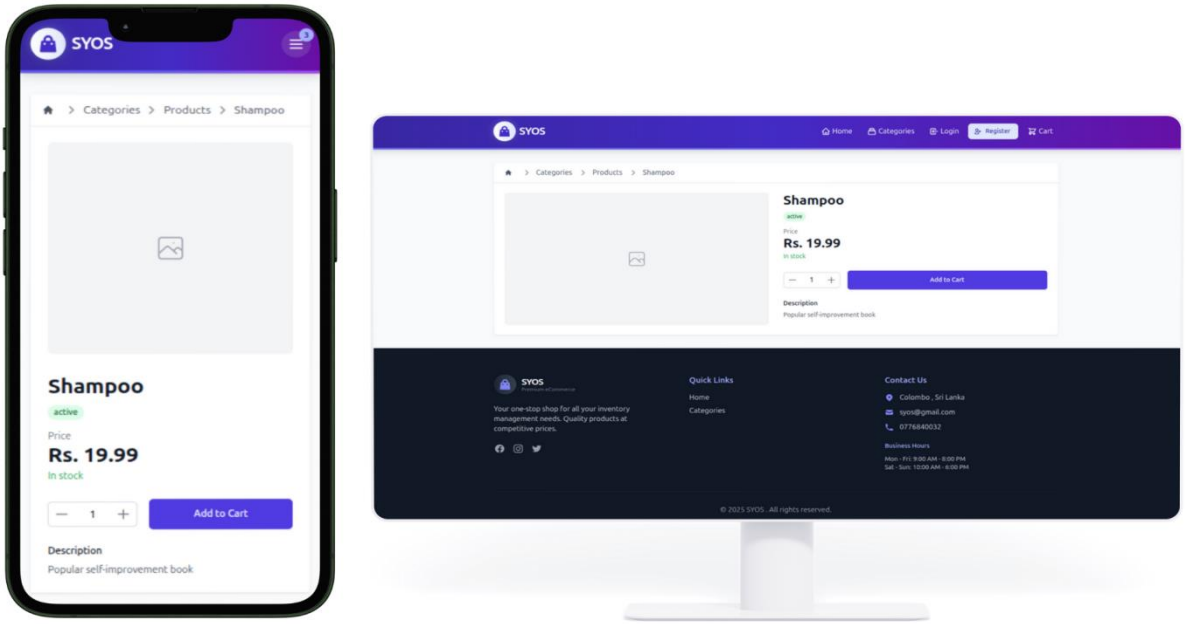


Figure 22 : Product View

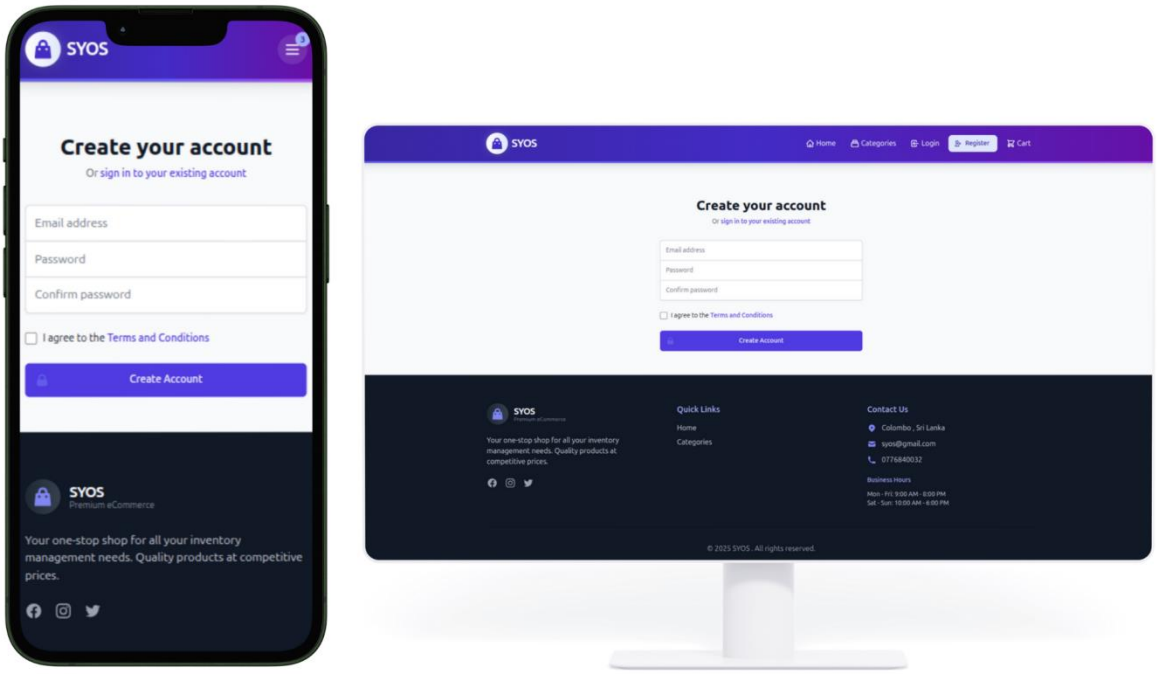


Figure 23 : Register Page

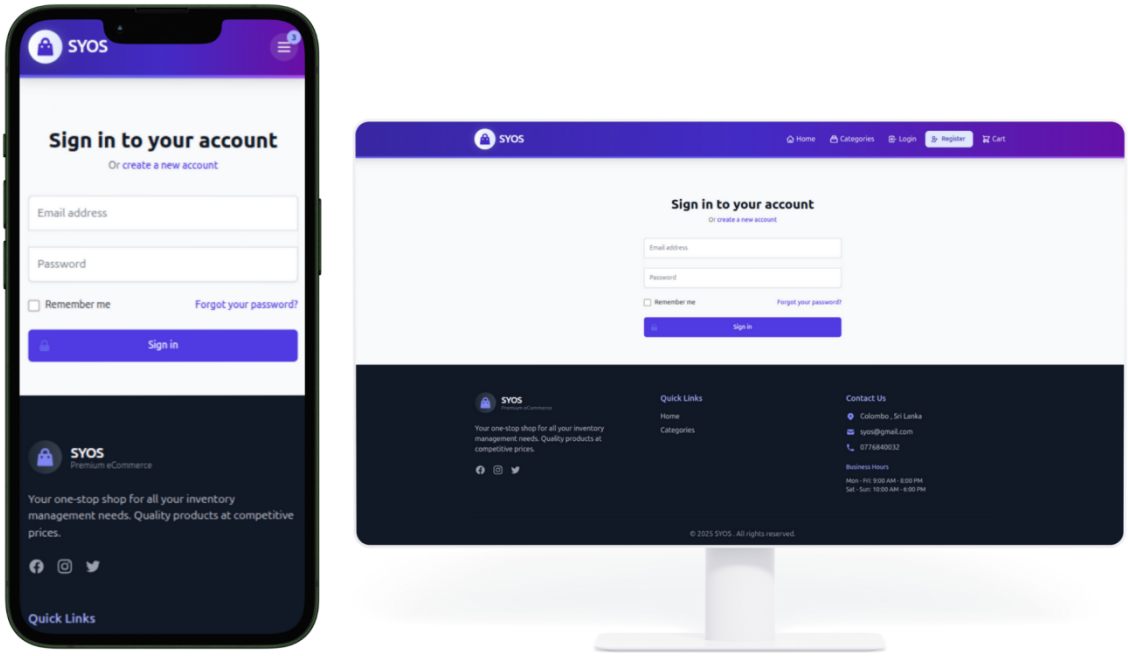


Figure 24 : Login Page

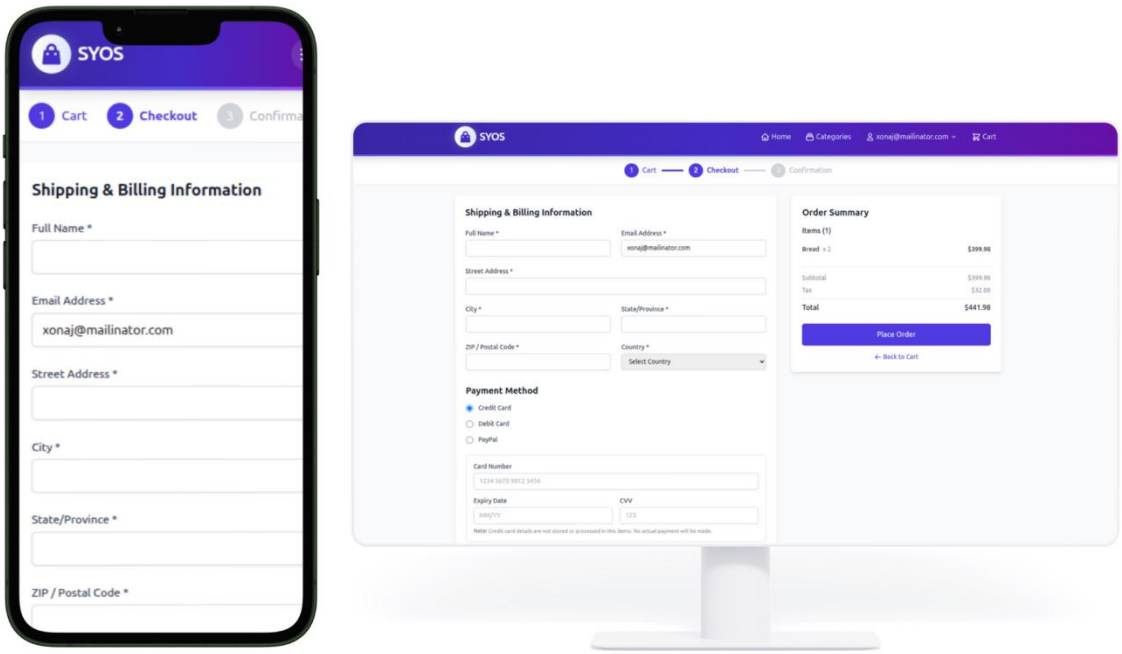


Figure 25 : Checkout Page

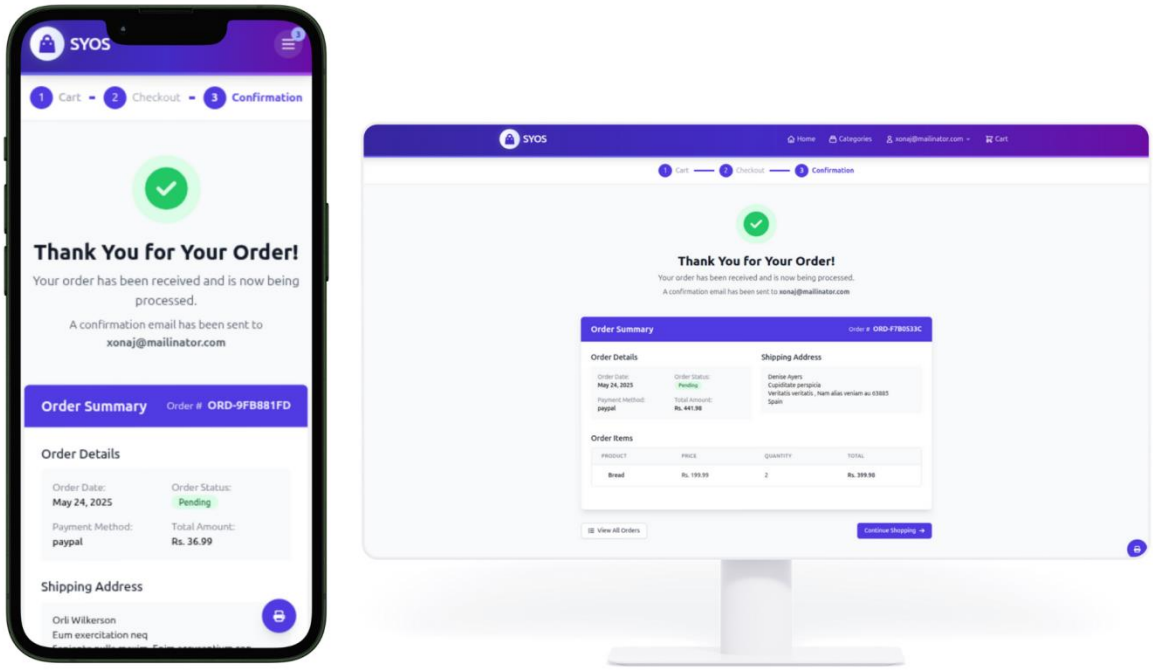
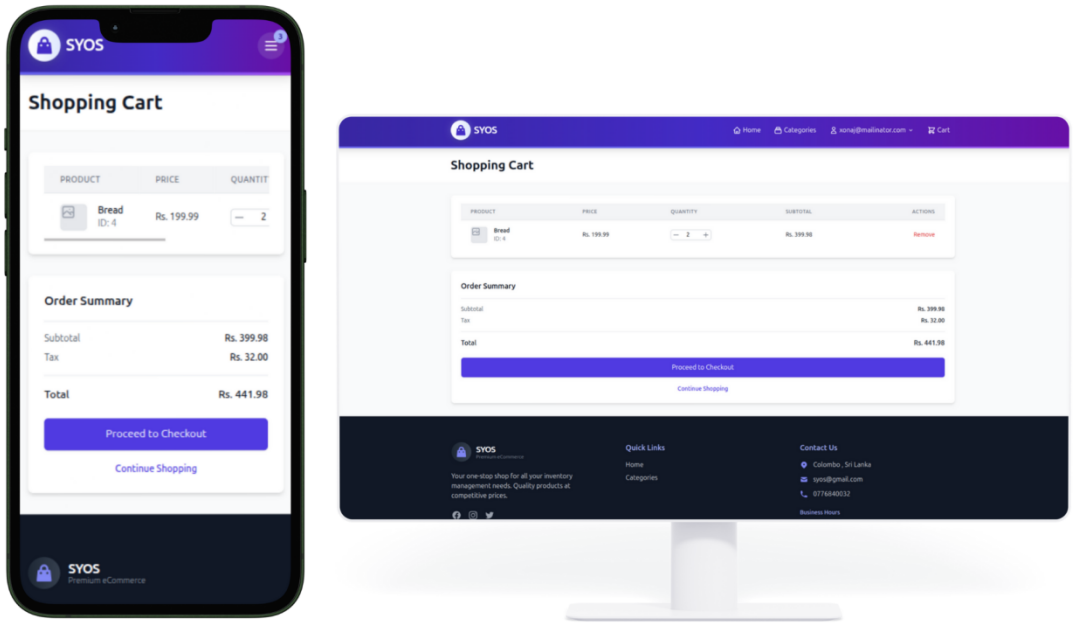


Figure 26 : Thank You Page

Figure 27 : Cart Page



2.3 Enhanced User Experience Features

The user interface is built with advanced UX patterns to improve how easy it is to use and access the system. The main characteristics are that

- Real-Time Feedback - Forms provide instant validation using visual indicators (green/red), enabling users to correct input errors immediately.
- Contextual Help Systems - Integrated assistance tools offer users guidance based on their current interactions.
- Smart Admin Dashboard - Interactive metric cards display real-time business metrics, enabling quick and informed decision-making.
- Sophisticated Notification System - Real-time alerts notify users of critical events

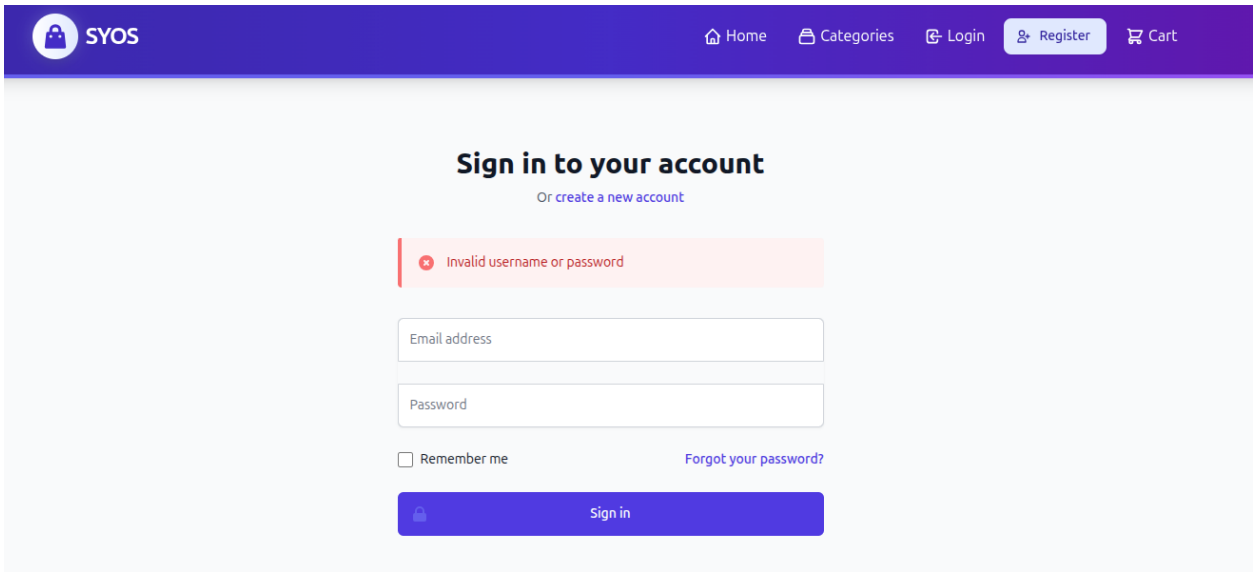


Figure 28 : Validation

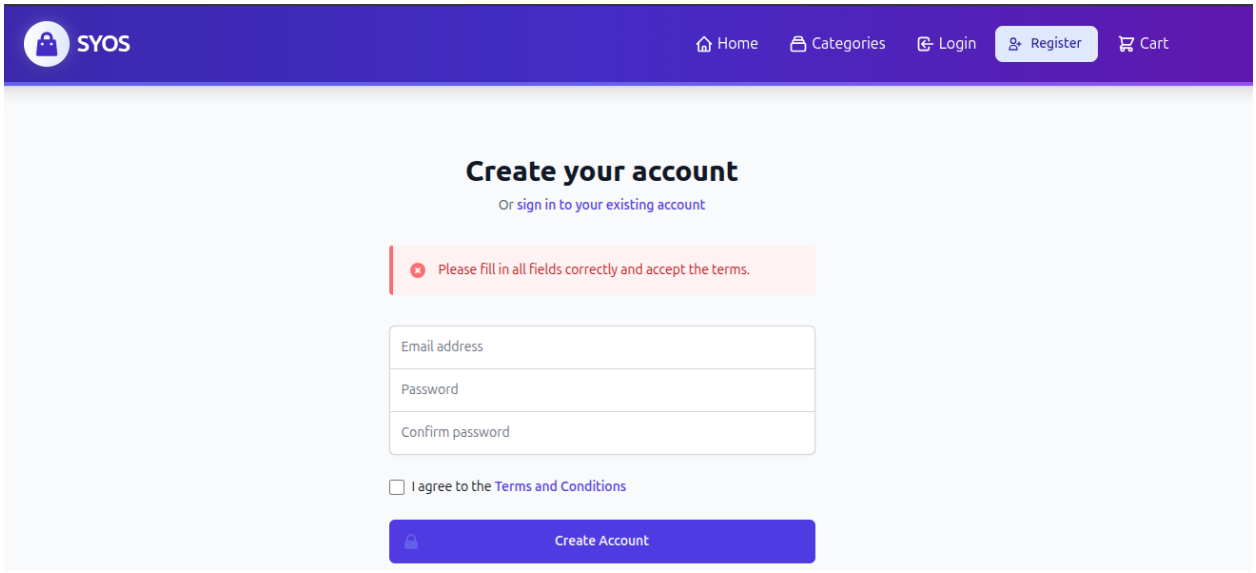
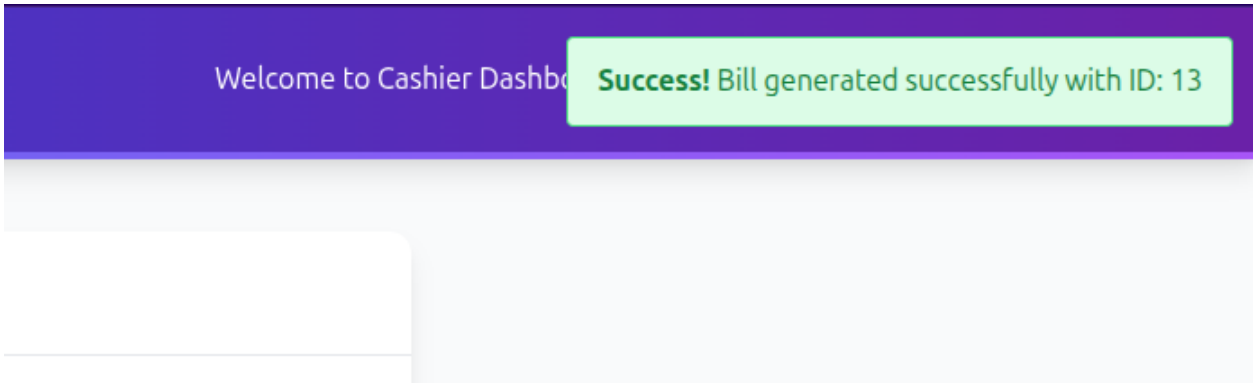


Figure 29 : Real-time alerts Notification



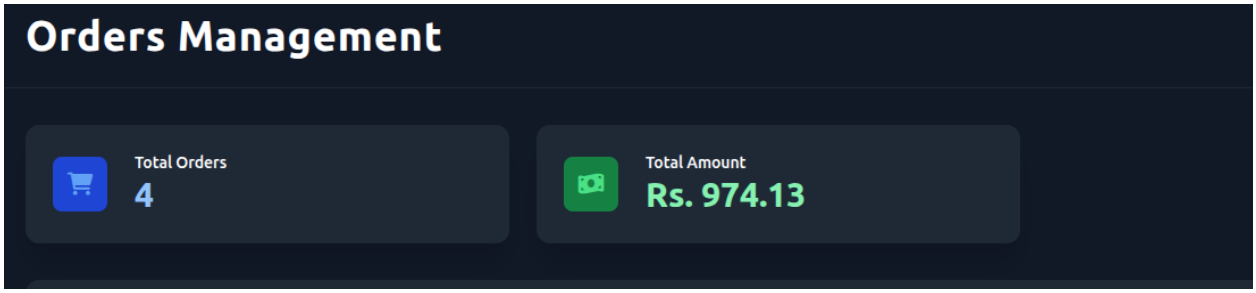
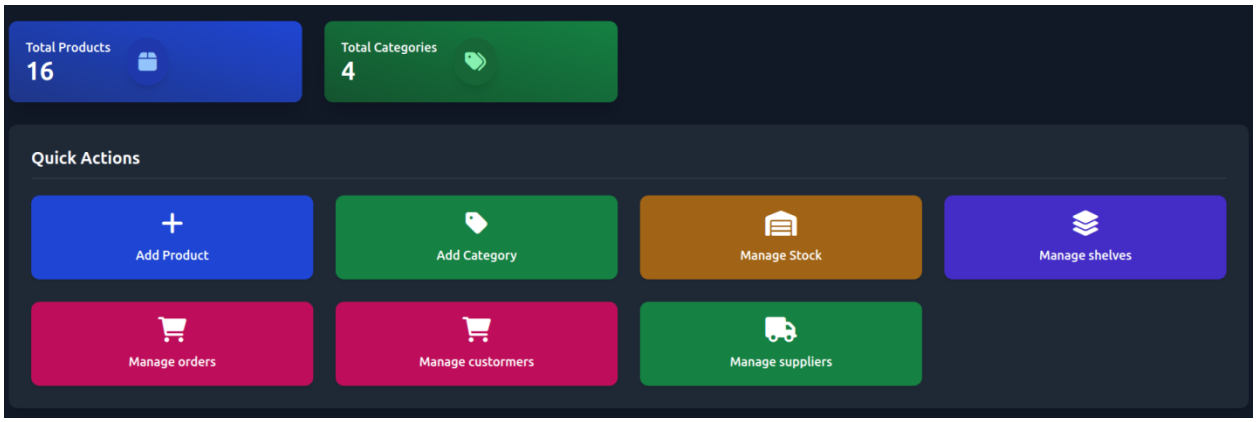


Figure 30 : Interactive metric cards display

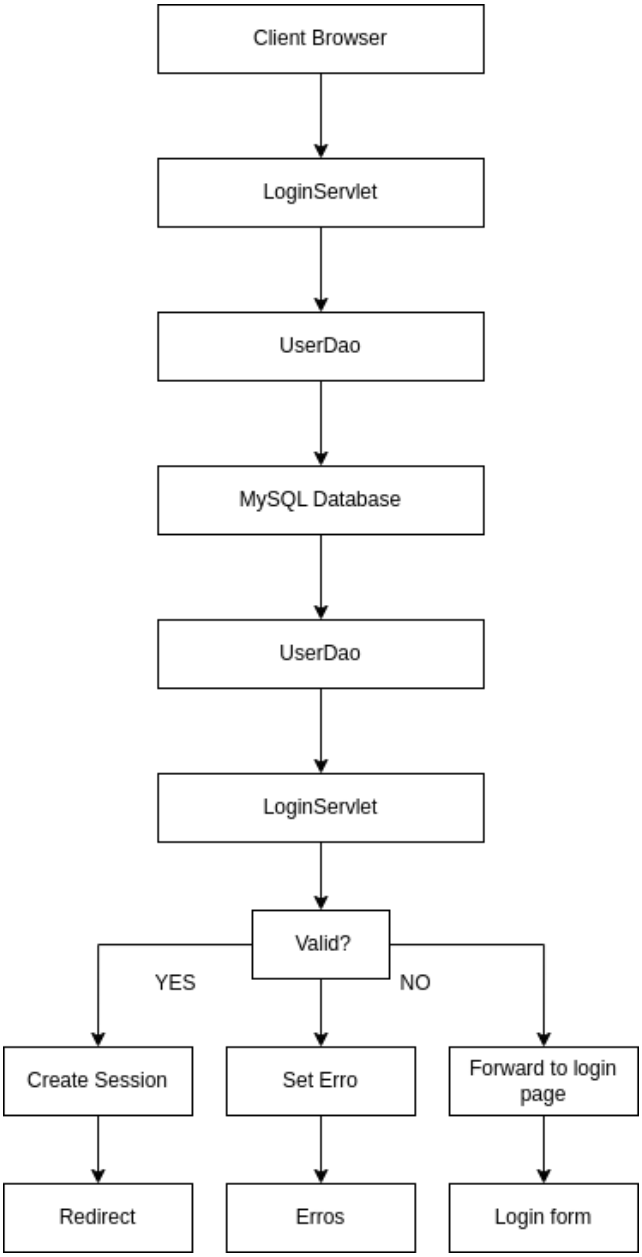


3. Server-side Concurrency Architecture

3.1 Single Client Request Handling

The controller, service and data layers exist within the Clean Architecture in the system to guarantee thread safety. It covers error handling, input validation and reliable methods for displaying responses. The database is reached using prepared statements and connection pooling so SQL injection is prevented and the application performs better.

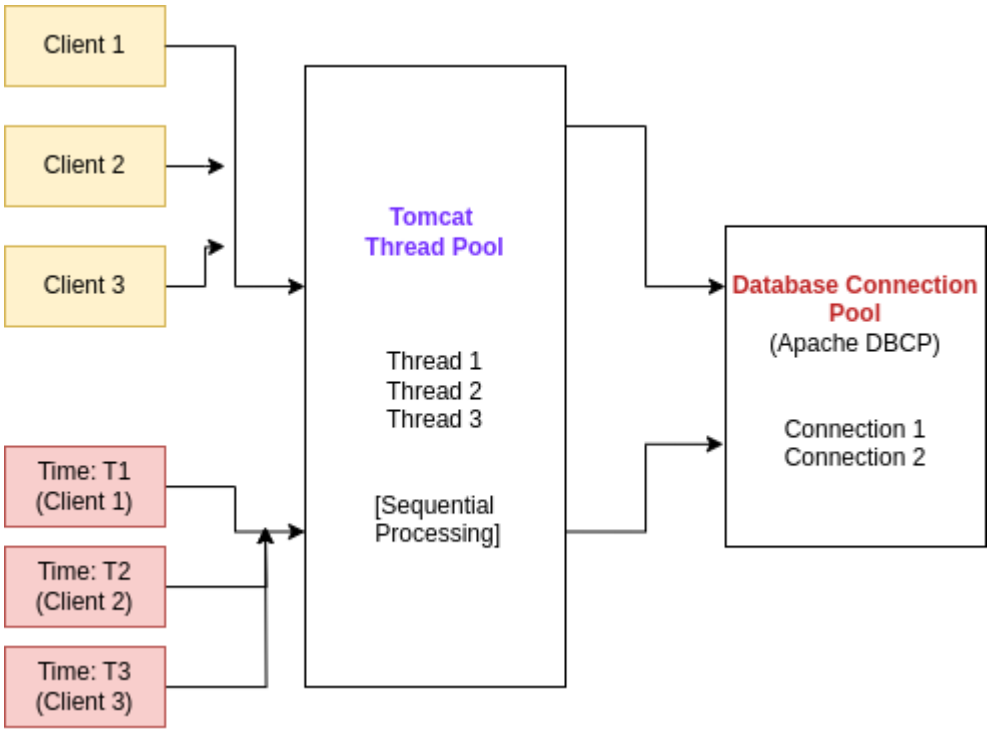
Figure 31 : Login Servlet request processing flow diagram



3.2 Multiple Client Sequential Processing

Clients are served in sequence by Tomcat's thread pool and each client keeps their session independent, using separate memory and authentication. It separates systems, so there are no conflicts, handles resources effectively and uses connection pooling with Apache DBCP for more efficient use of the database.

Figure 32 : Multiple client sequential processing architecture



3.3 Simultaneous Multi-Client Support

- Simultaneously accept multiple client requests by using thread and connection pooling
- Uses Apache DBCP for dedicated database connections per request
- Configured with maxThreads 200, acceptCount 100 (change server.xml in tomcat)
- Connection pool sizes - maxIdle 10,maxTotal 20, minIdle 5
- Supports high concurrent load with reliable response times

```
GNU nano 7.2                          server.xml
      HTTP Connector: /docs/config/http.html
      AJP  Connector: /docs/config/ajp.html
      Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
  -->
  <Connector port="8080" protocol="HTTP/1.1"
             connectionTimeout="20000"
             maxThreads="200"
             minSpareThreads="25"
             acceptCount="100"
             redirectPort="8443" />
  <!-- A "Connector" using the shared thread pool-->
  <!--
  <Connector executor="tomcatThreadPool"
             port="8080" protocol="HTTP/1.1"
```

Figure 33 : server.xml

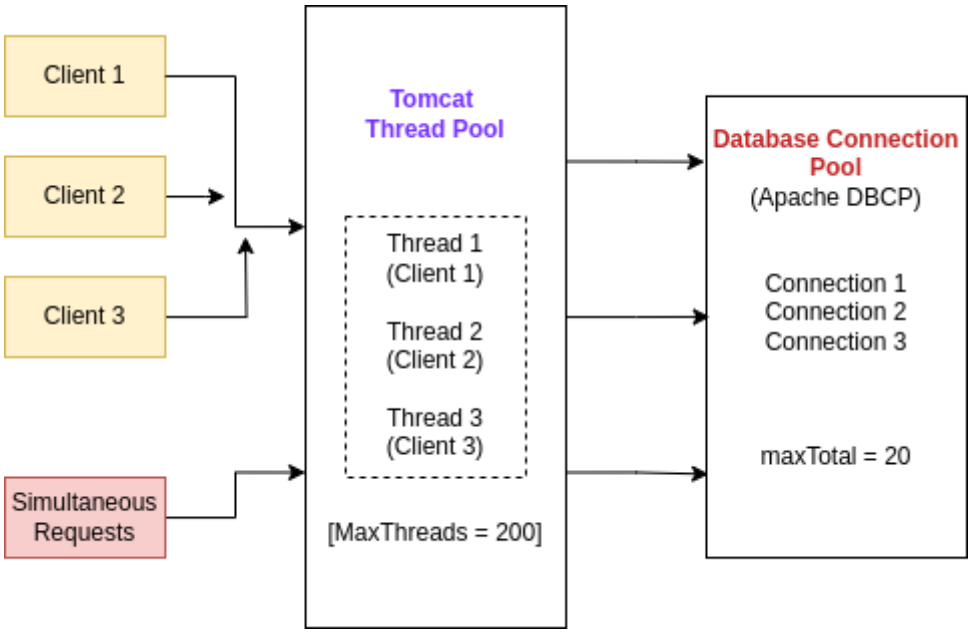


Figure 34 : Concurrent multi-client processing with thread pool management

3.4 High-Volume Request Queuing System

- Tomcat's request queuing and connection pool overflow management
- maxWaitMillis (30000) limits database wait time
- Queues requests when threads are maxed out
- acceptCount (100) buffers requests during peaks
- Prevents system failure and maintains response times during spikes

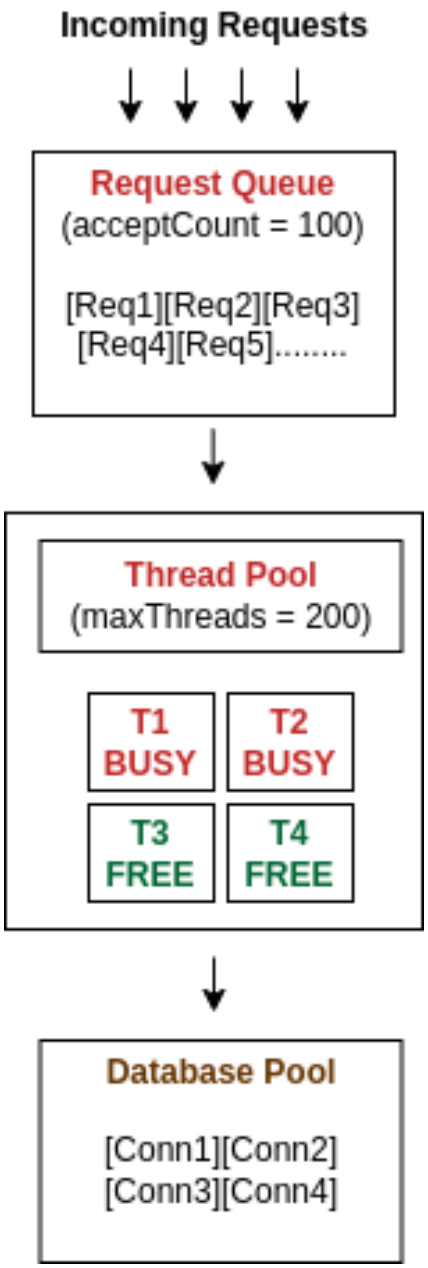
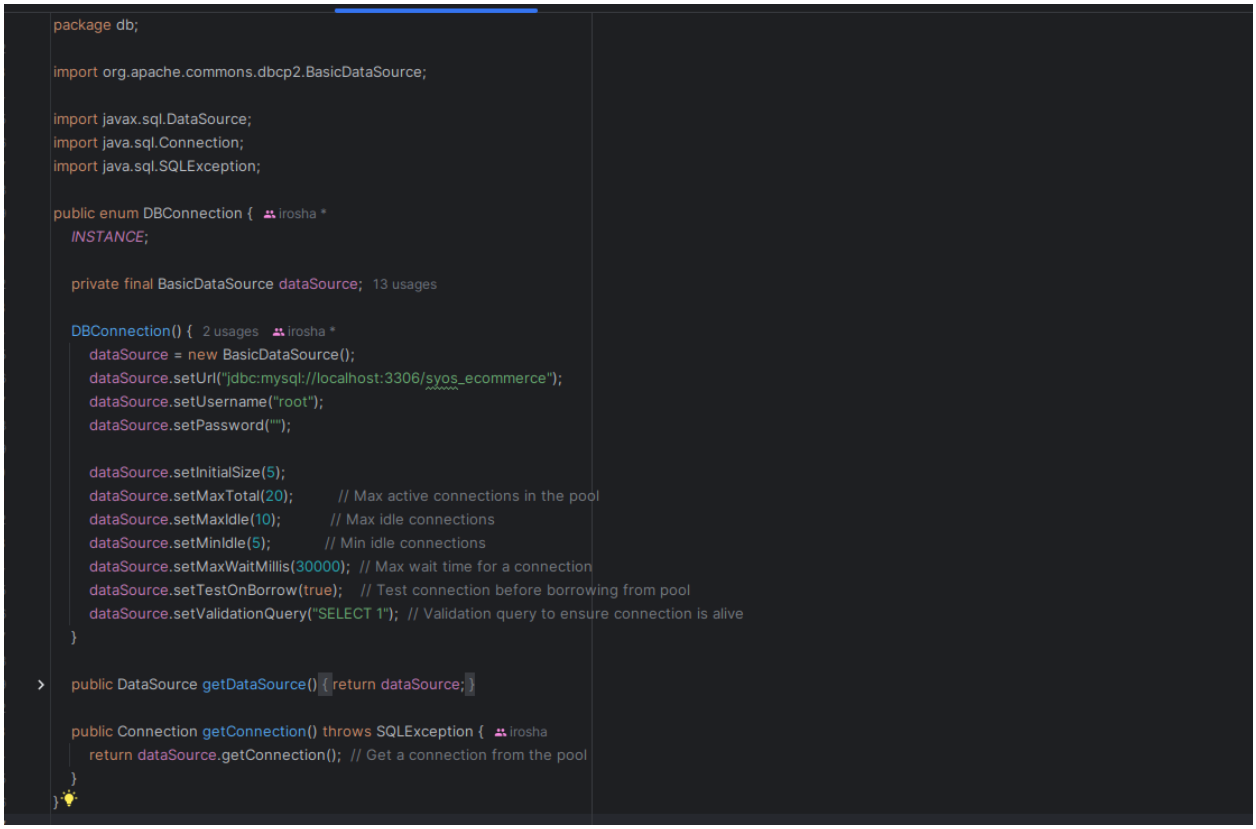


Figure 35 : Request queuing system with overflow protection



```java
package db;

import org.apache.commons.dbcp2.BasicDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

public enum DBConnection {  irosha *
    INSTANCE;

    private final BasicDataSource dataSource;  13 usages

    DBConnection() {  2 usages   irosha *
        dataSource = new BasicDataSource();
        dataSource.setUrl("jdbc:mysql://localhost:3306/syos_ecommerce");
        dataSource.setUsername("root");
        dataSource.setPassword("");

        dataSource.setInitialSize(5);
        dataSource.setMaxTotal(20);          // Max active connections in the pool
        dataSource.setMaxIdle(10);           // Max idle connections
        dataSource.setMinIdle(5);            // Min idle connections
        dataSource.setMaxWaitMillis(30000);  // Max wait time for a connection
        dataSource.setTestOnBorrow(true);    // Test connection before borrowing from pool
        dataSource.setValidationQuery("SELECT 1");  // Validation query to ensure connection is alive
    }

    public DataSource getDataSource() { return dataSource; }

    public Connection getConnection() throws SQLException {  irosha
        return dataSource.getConnection();  // Get a connection from the pool
    }
}
```

Figure 36 : implementation of DBCP

4. Client-side Concurrency Implementation

4.1 Basic Client-Server Communication

The client-side architecture handles communication by using the fetch API, Error handling with Promises and automatically trying again for transient problems. Server communications do not restrict user interaction with the app, so users can respond instantly during network activities.

The process also manages different errors by reducing entirely working parts rather than causing the app to crash. User feedback systems offer a clear way to follow long running operations without disrupting how the interface is used.

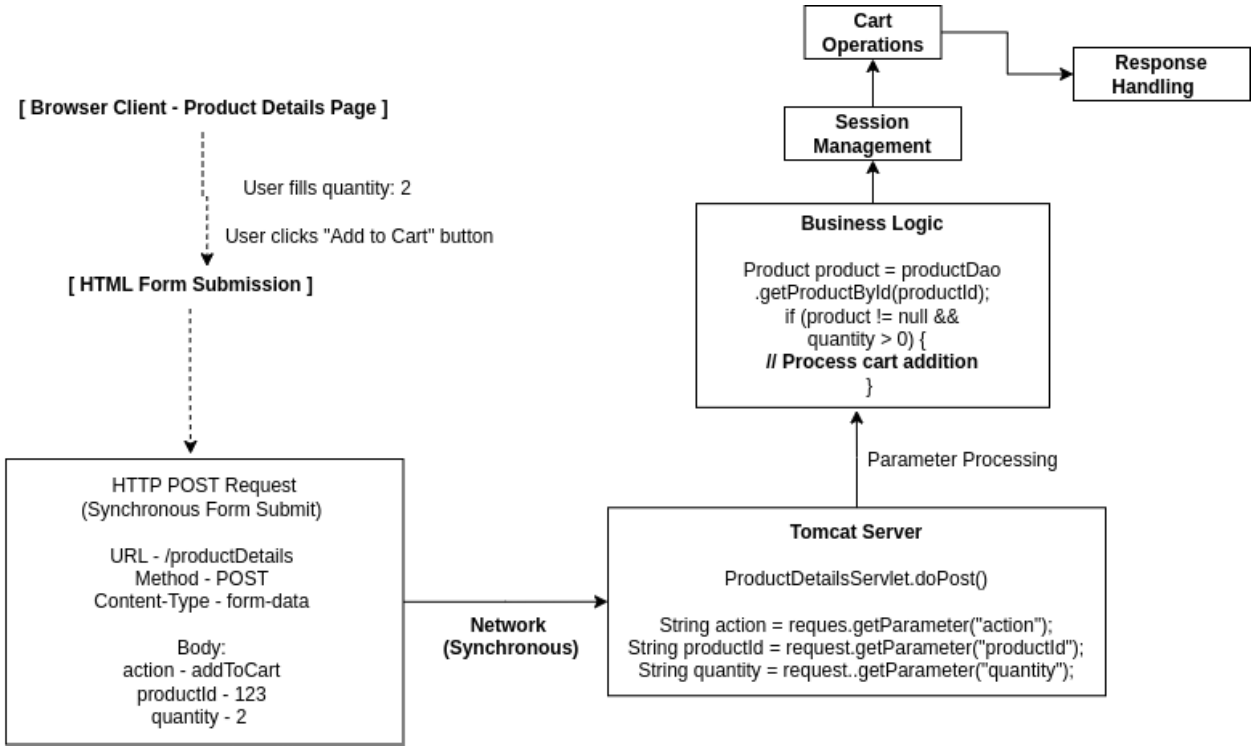


Figure 37 : Client-Server Synchronous Communication (Add to Cart Form Submission)

4.2 Multi-Computer Client Interaction

Multiple clients on different systems can use the server's services without their actions interfering with one another. During operations, the system creates separate session space for each voice call. At the same time, it allows people to collaborate in real time when the situation demands.

As a result of session management, user logins, cart contents and orders work the same both within different browser sessions and even after network interruptions. Implementation stops session hijacking and supports secure operation by multiple clients in common environments.

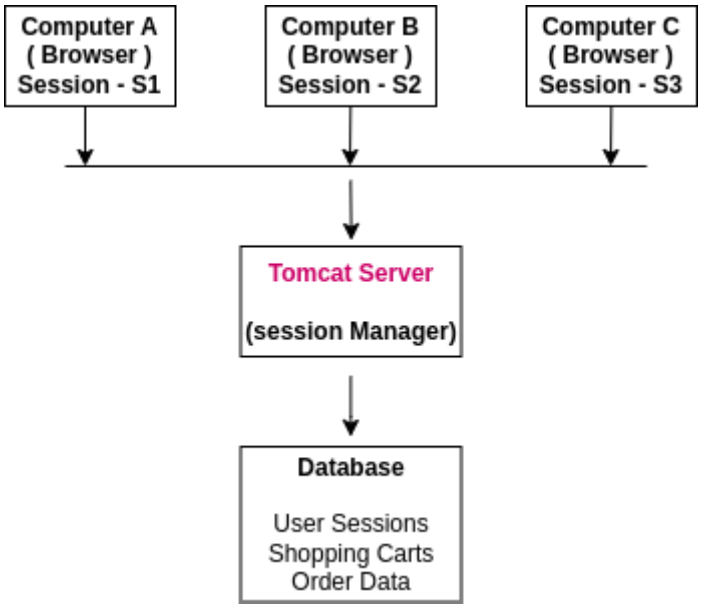


Figure 38 : Multi-computer client coordination and session management

4.3 Real-time Data Synchronization

While the current implementation focuses on individual user sessions and the architecture supports real-time synchronization capabilities for inventory updates and order processing. Product availability changes and pricing updates propagate to client interfaces through efficient polling mechanisms that minimize server load.

The system implements intelligent caching strategies that balance data freshness with performance requirements, ensuring that users always see current information while minimizing unnecessary server requests.

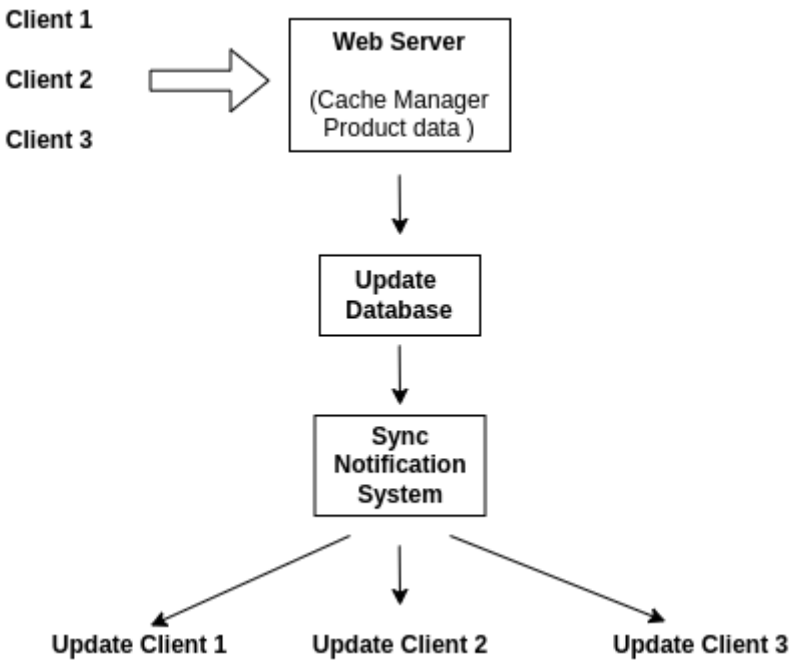


Figure 39 : Real-time data synchronization patterns

4.4 Asynchronous Test Client Development

Apache JMeter allows a full check of the system's concurrent performance by using automated clients that imitate how users interact with the system. Both GET and POST methods are tested using requests at all the main endpoints in the application.

In test configurations, multiple groups of threads replicate various simultaneous user activities with various start and request timing. All the tests under concurrent load conditions showed that the app could handle sign-in, registration, working with the shopping cart and order processing, all without any mistakes.

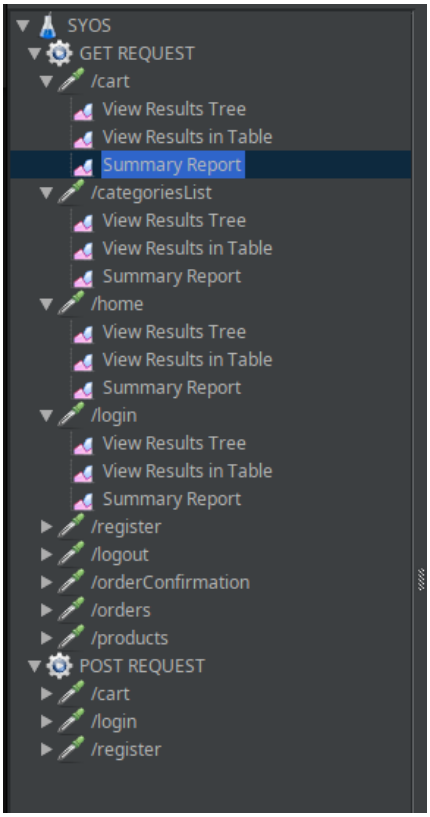


Figure 40 : Jmeter GET , POST Request

4.4.1. GET Request

- /cart

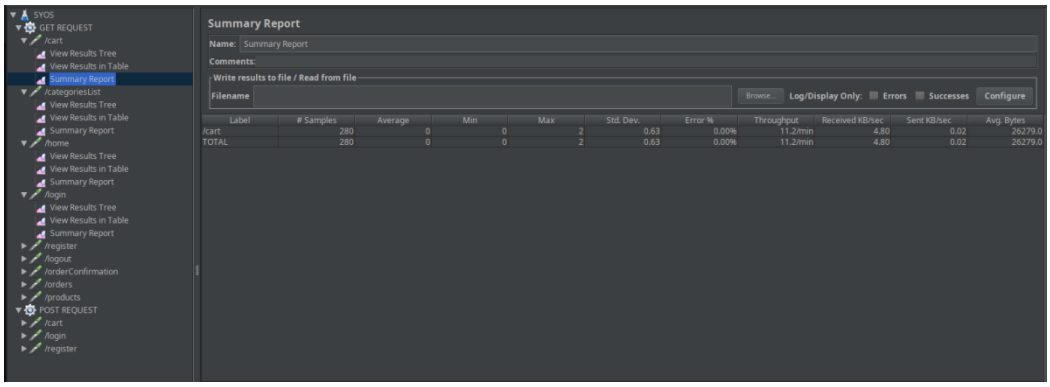


Figure 41 : Jmeter /cart (get request)

- /categoriesList

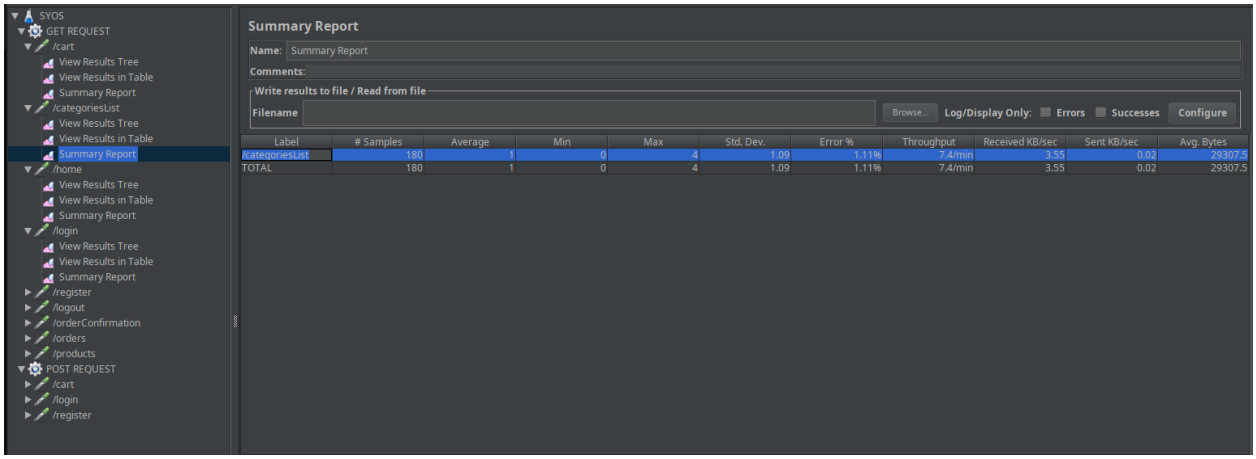


Figure 42 : Jmeter /category (get request)

- /home

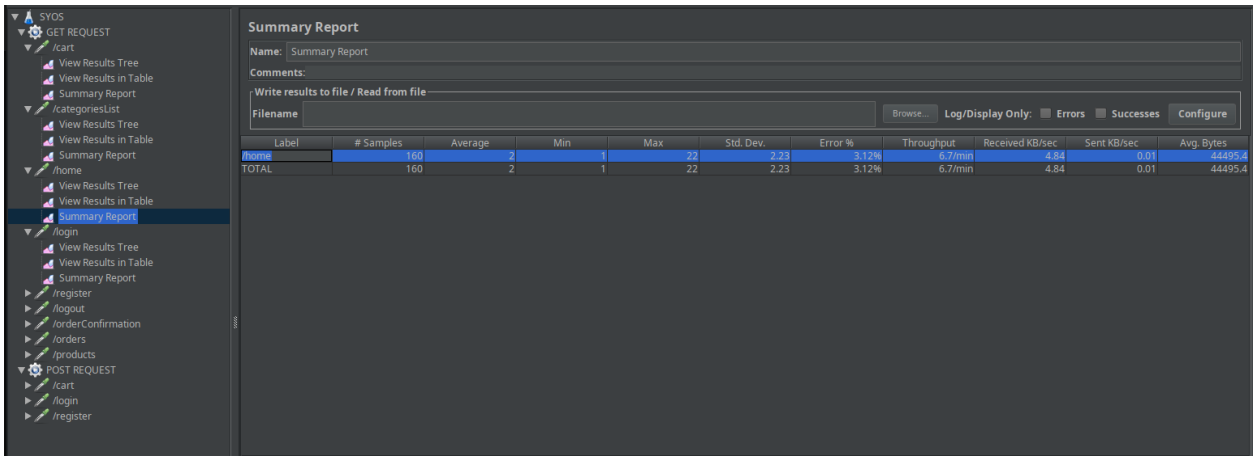


Figure 43 : Jmeter /home (get request)

- / login

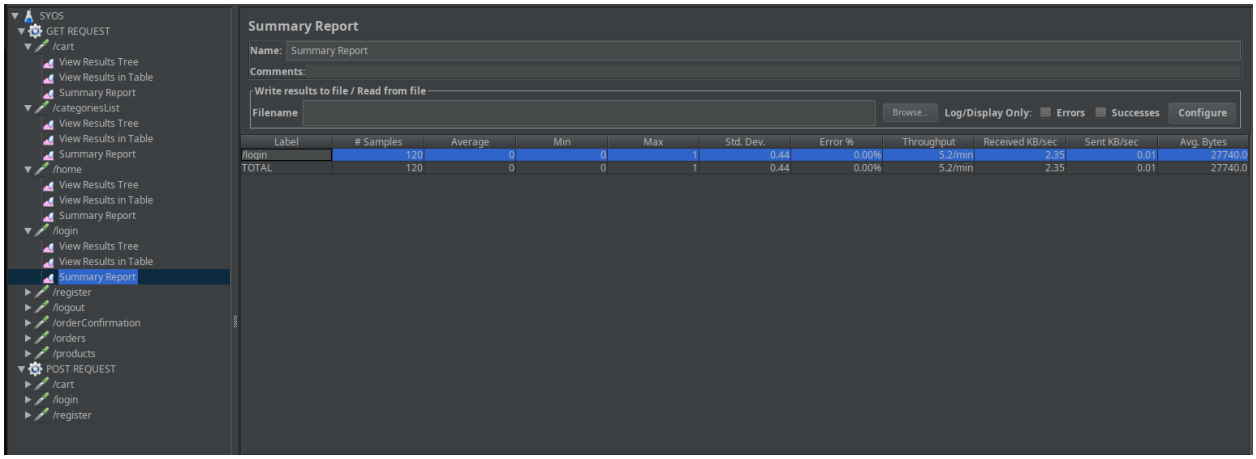


Figure 44 : Jmeter /login (get request)

- /register

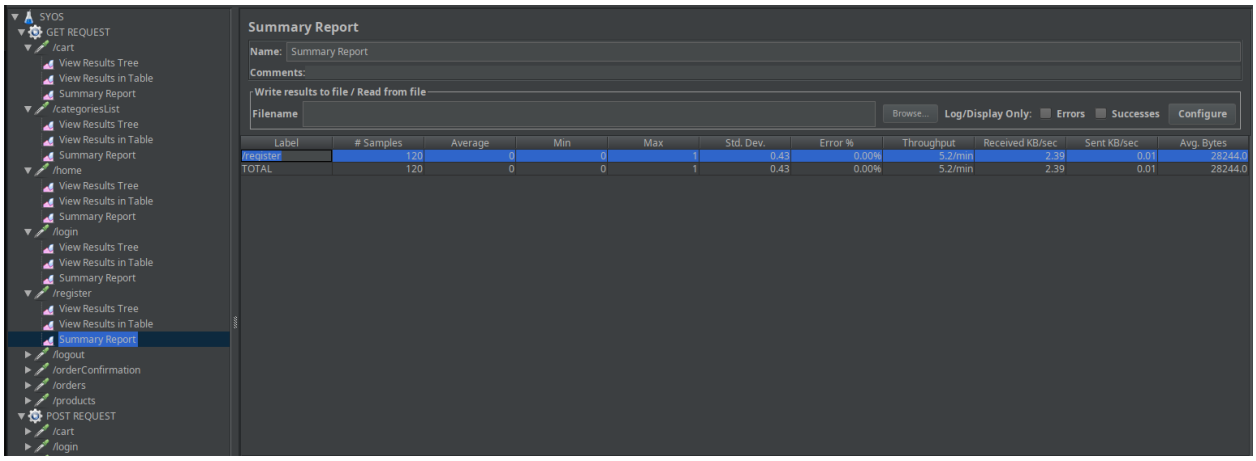


Figure 45 : Jmeter /register (get request)

- /logout

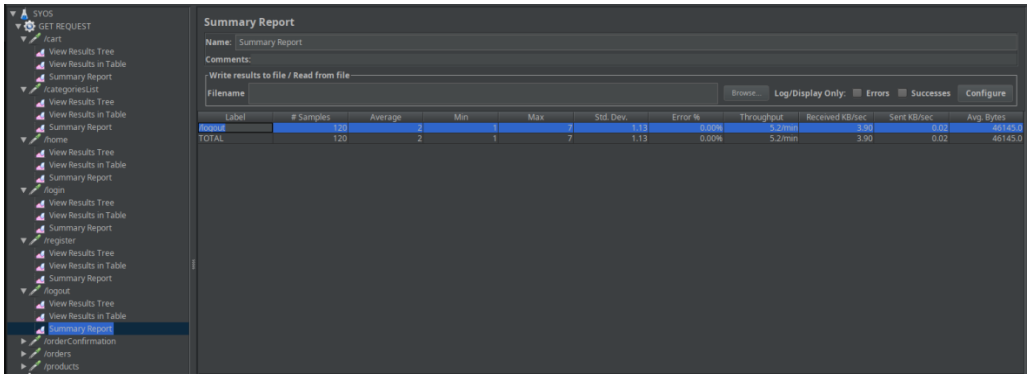


Figure 46 : Jmeter /logout (get request)

- /orderConfirmation

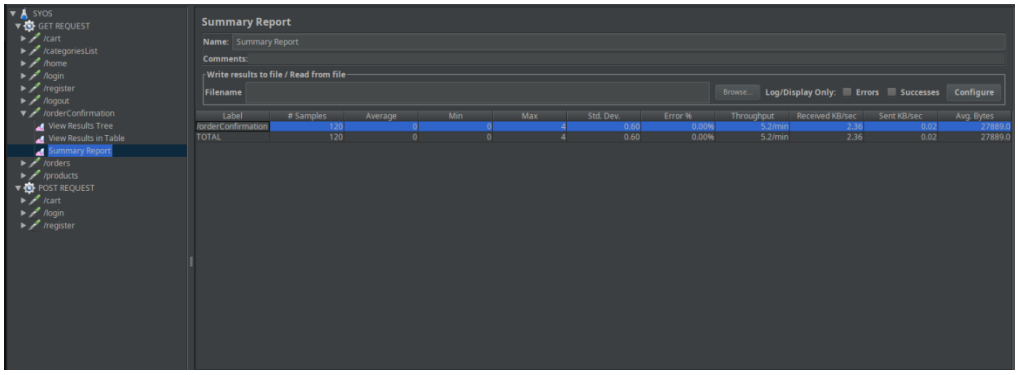


Figure 47 : Jmeter /orderConfirmation (get request)

- /orders

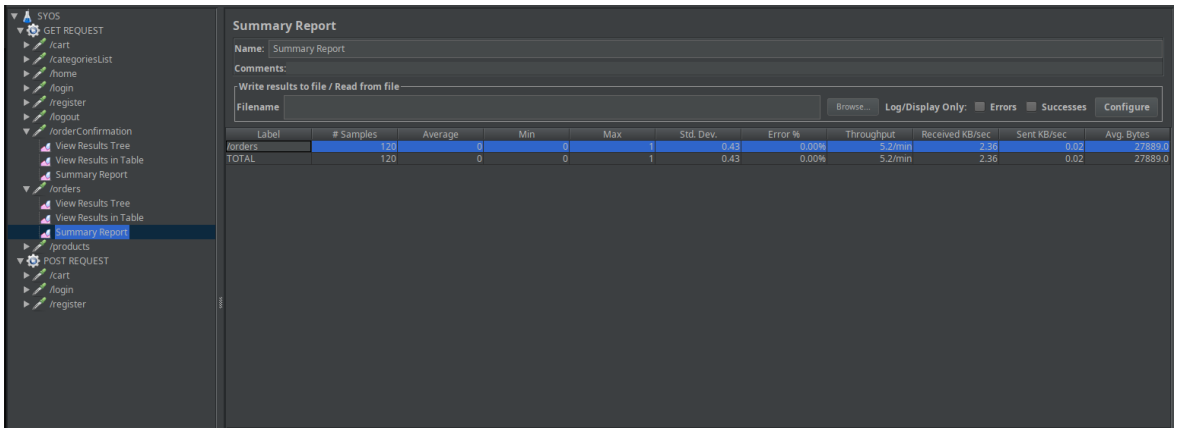


Figure 48 :Jmeter /orders (get request)

- /products

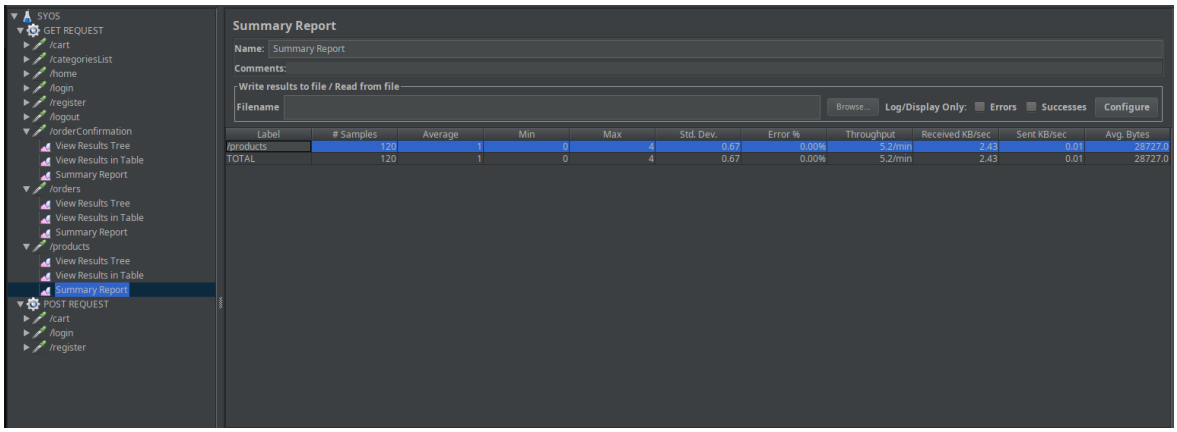


Figure 49 : Jmeter /products (get request)

4.4.2. POST Request

- /cart

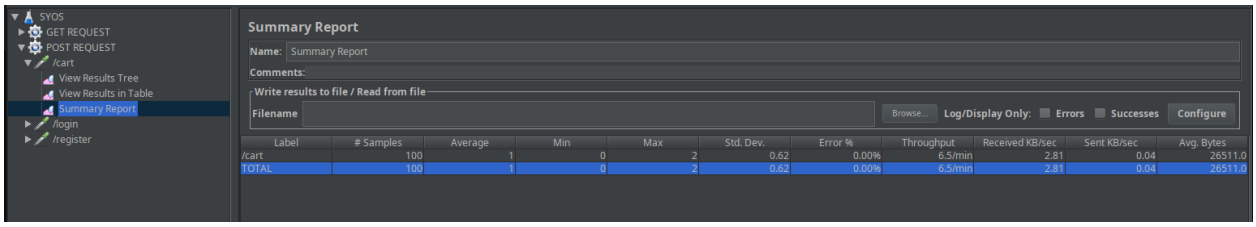


Figure 50 : Jmeter /cart (post request)

- /login

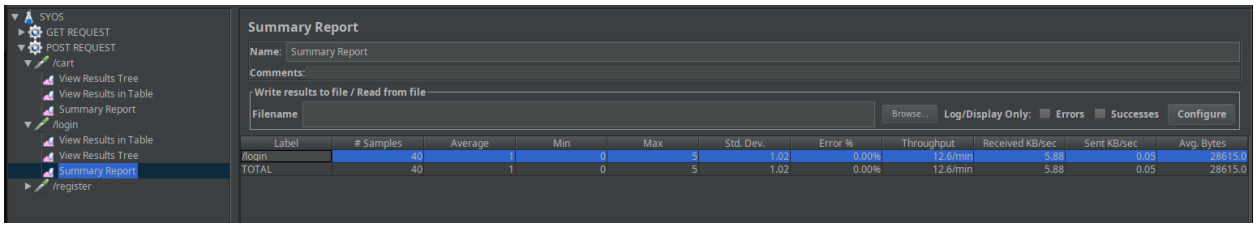


Figure 51 : Jmeter /login (post request)

- /register

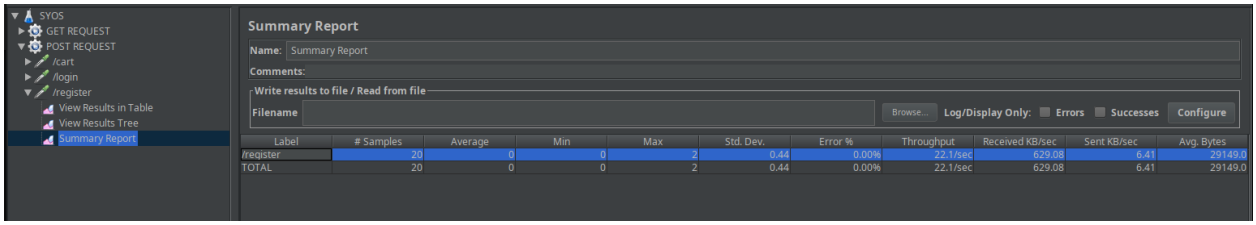


Figure 52 : Jmeter /register (post request)

5. Clean Architecture Design

5.1 Multi-tier Architecture Preservation

The change shows that Clean Architecture can handle dependencies independently, no matter where they are used. Controllers rely on service interfaces, services use repositories in their abstractions and entities are free from external connections, so the system can be tested and used flexibly.

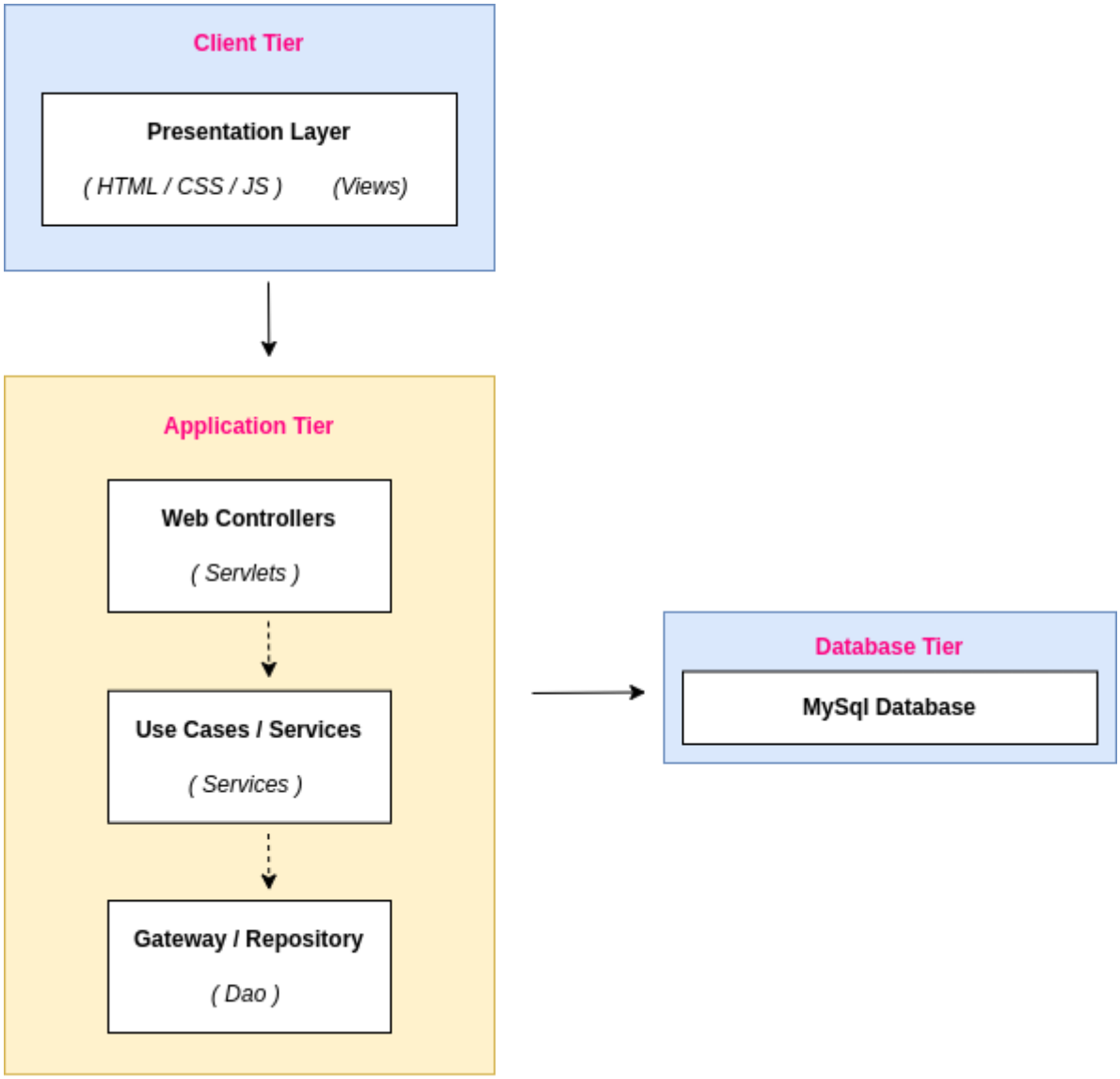


Figure 53 : Clean Architecture preservation across distributed tiers

5.2. Folder Structure

This section will discuss on the current folder structure of the project ensuring to logically

Separate the client and server separately.

Key Feature of folder architecture,

- Clear Separation Between Client and Server
    - WEB-INF - Contains all client-side web interface components
    - Src - Contains all server-side business logic
- Evolution from Assignment 1
    - Original controller package from CLI system is now unused
    - servlets package replaces controllers for web requests
    - All business logic preserved in service layer

- Clear Architecture Preservation
    - Presentation Layer - JSP files in WEB-INF/views/
    - Business Layer - Services in src/service/
    - Data Layer - DAOs in src/dao/

- Design Pattern Organization
    - Command - Command pattern implementations
    - Observer - Observer pattern for notifications
    - Strategy - Strategy pattern for reports
    - State - State pattern for user roles
    - Factory - Factory pattern for object creation

- Web Interface Structure
    - admin/ subfolder for administrative interfaces
    - assets/ for static resources (CSS, JS)
    - base-layout.jsp as master template

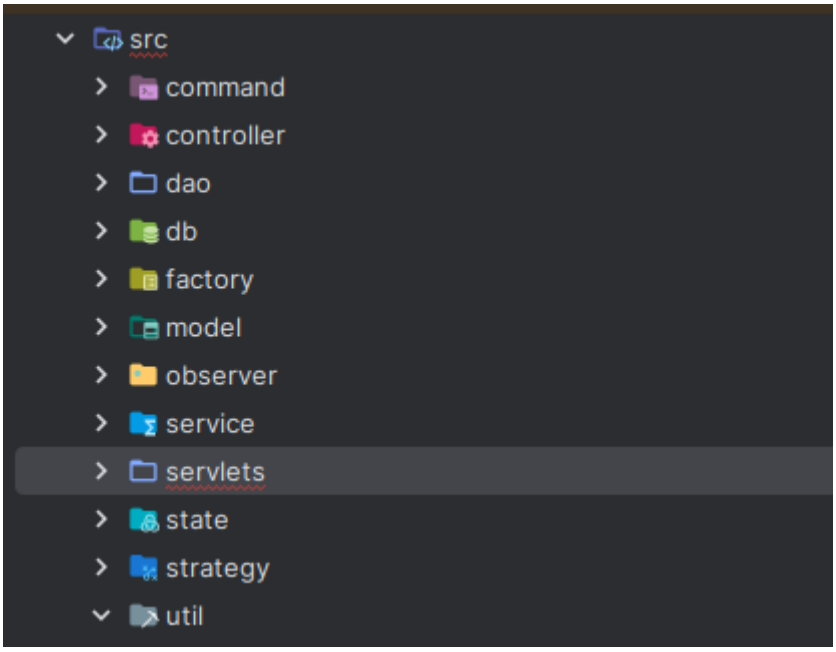
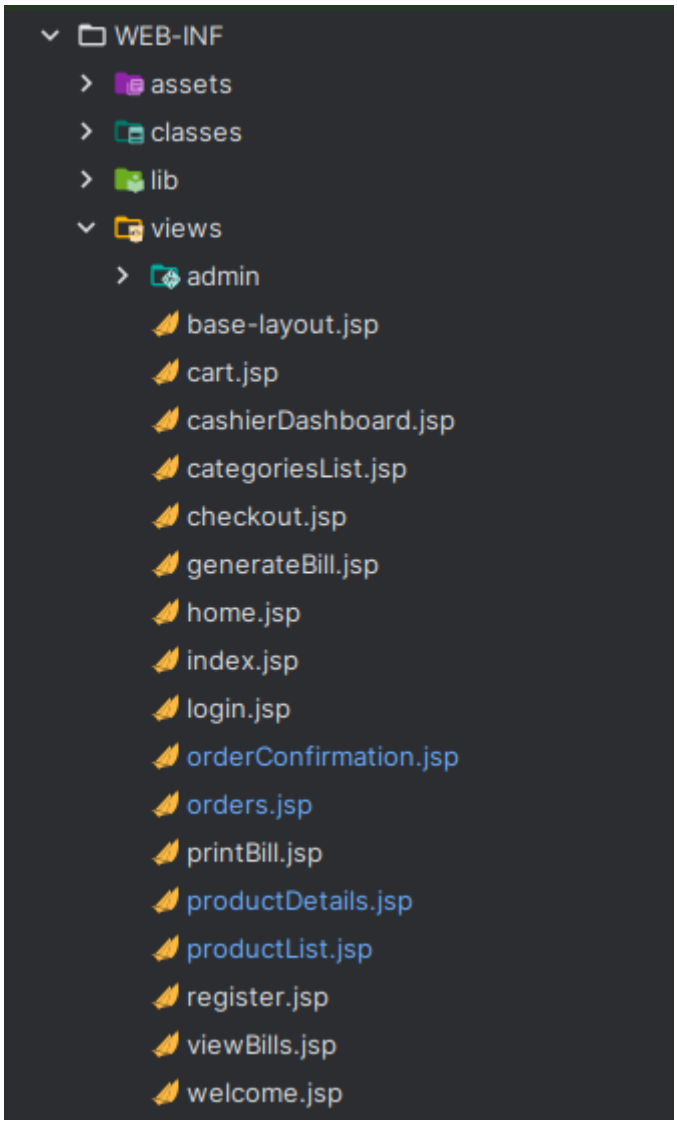


Figure 54 : Folder structure



5.3. Implementation Discussion

5.3.1.Online and Physical Stock Management

Orders for customers who buy online products are saved in the order table. And the inventory of those products is managed in the stock table. Bills for customers who buy physical products are saved in the bills table as before, and the inventory of those who buy physical products is managed in the shelves.

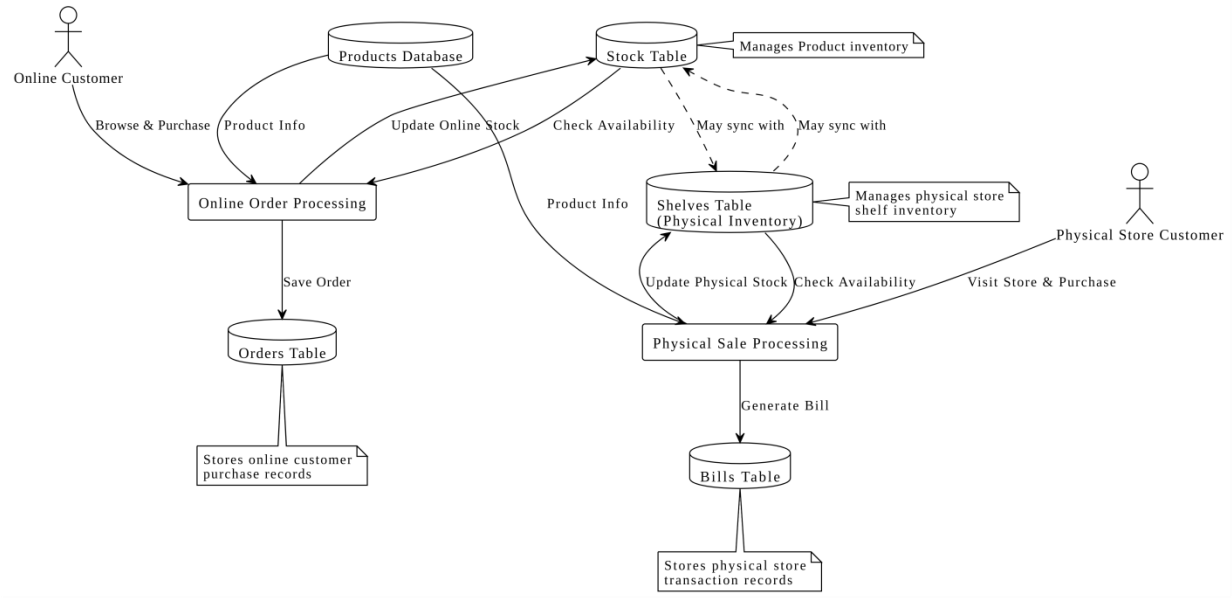


Figure 55 : Online and Physical Stock Management Diagram

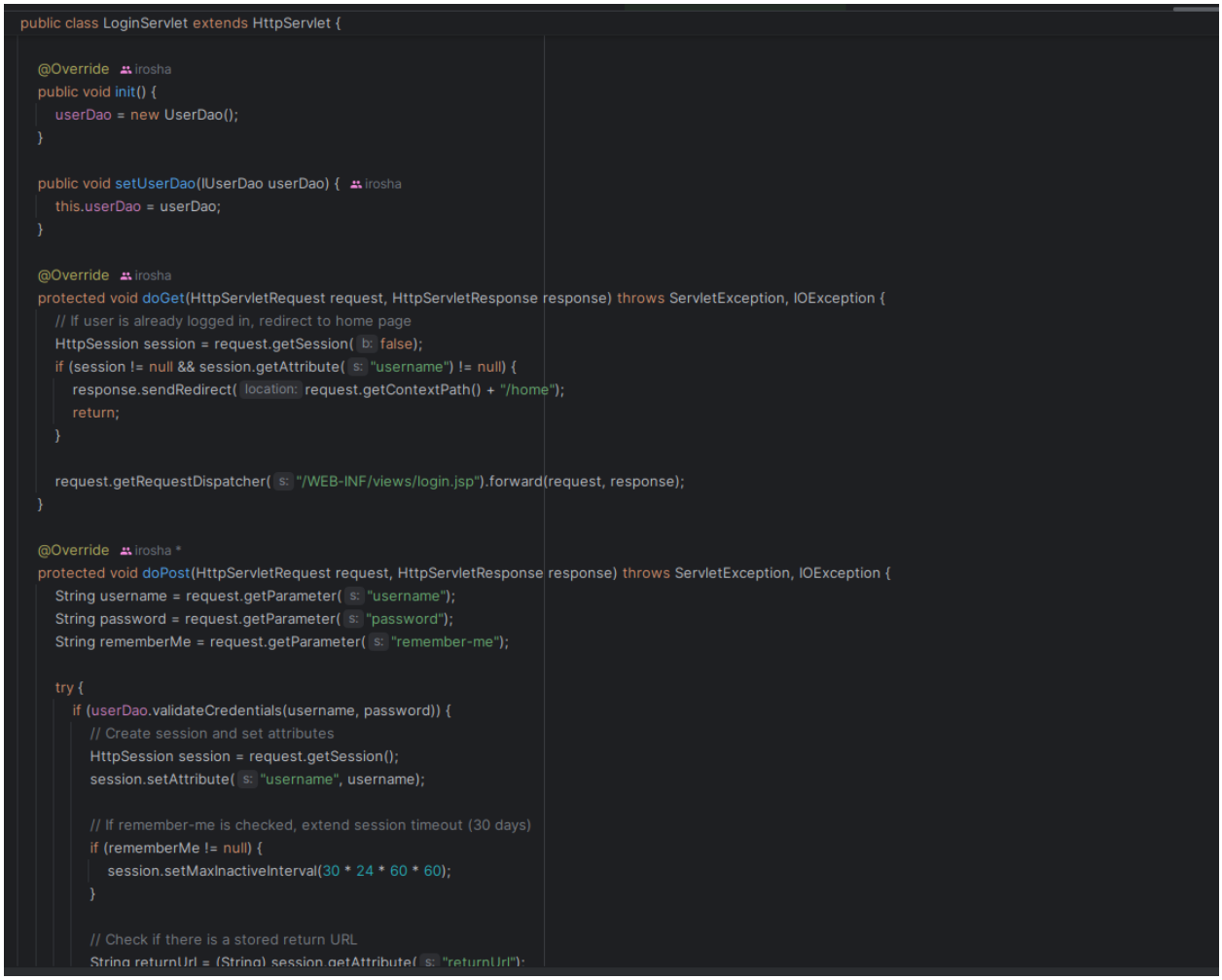
5.3.2. Java Servlet Architecture Implementation Discussion

Java Servlets in the SYOS system carry out the MVC pattern by controlling the back-and-forth HTTP request-response process between the client and the server. Let's use LoginServlet as an example to understand why. Using @WebServlet, LoginServlet was mapped to URL /login to ensure clean URL routing and help separate presentation, business logic and data access.

Sophisticated session control is shown by the LoginServlet with its dual methods. The doGet() method validates a session by verifying that a user is logged in with if (session != null && session.getAttribute("username") != null). If a user is active and logged in, an automatic redirect is provided so they can easily get back to the home page.

Validation of credentials in the doPost() method follows Clean Architecture by using userDao.validateCredentials(username, password). After successful authentication, the servlet

sets up a new session and keeps user information by setting session.setAttribute("username", username). Remember-me is a feature added through session.setMaxInactiveInterval(30 * 24 * 60 * 60) which allows users to decide how long their session stays active before expiring. When JSPs are put in the WEB-INF directory, they cannot be accessed directly with HTTP which ensures all navigation goes through the servlet application. Anyone who uses /login will be taken care of by LoginServlet which will authorize the request and then forward it to login.jspprotected in WEB-INF. This set-up makes it easy to process multiple requests and keeps the platform fully capable.



```
public class LoginServlet extends HttpServlet {

    @Override  irosha
    public void init() {
        userDao = new UserDao();
    }

    public void setUserDao(IUserDao userDao) {  irosha
        this.userDao = userDao;
    }

    @Override  irosha
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // If user is already logged in, redirect to home page
        HttpSession session = request.getSession( b: false);
        if (session != null && session.getAttribute( s: "username") != null) {
            response.sendRedirect( location: request.getContextPath() + "/home");
            return;
        }

        request.getRequestDispatcher( s: "/WEB-INF/views/login.jsp").forward(request, response);
    }

    @Override  irosha *
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter( s: "username");
        String password = request.getParameter( s: "password");
        String rememberMe = request.getParameter( s: "remember-me");

        try {
            if (userDao.validateCredentials(username, password)) {
                // Create session and set attributes
                HttpSession session = request.getSession();
                session.setAttribute( s: "username", username);

                // If remember-me is checked, extend session timeout (30 days)
                if (rememberMe != null) {
                    session.setMaxInactiveInterval(30 * 24 * 60 * 60);
                }

                // Check if there is a stored return URL
                String returnUrl = (String) session.getAttribute( s: "returnUrl");
```

Figure 56 : Login Servlet Code

5.3.3. Java Server Pages

This system for Java Server Pages uses components to promote modularity and allow code to be used multiple times. To keep user access secure, the WEB-INF directory holds all the JSP files, so navigation requires using servlet controllers. Following this design supports Java EE security and continues to use the MVC pattern cleanly. Using the component-based pattern,

programmers build reusable UI features for recurring things like headers, footers and navigation menus. Instead of copying the same features throughout, the system adds these components to various areas using a module system. As a result of this, it becomes much more convenient to make changes to the common elements, because these are updated for every page that uses them. As an example, changing the navigation or branding needs to be done to just one header component instead of touching many pages. The home page illustrates this style by organizing shared features and including them randomly as needed. The page makes use of a main template that structures the page, displays a navigation and branding header, holds home content features and has a footer with information found on all pages. Because of this approach, the site maintains a similar design everywhere but allows every page to modify its content areas separately, making it easier to maintain the platform.

Figure 57 : base layout jsp

Figure 58 : welcome jsp

5.3.4. Testing

I used JUnit and Mockito to check if the servlet functions worked.

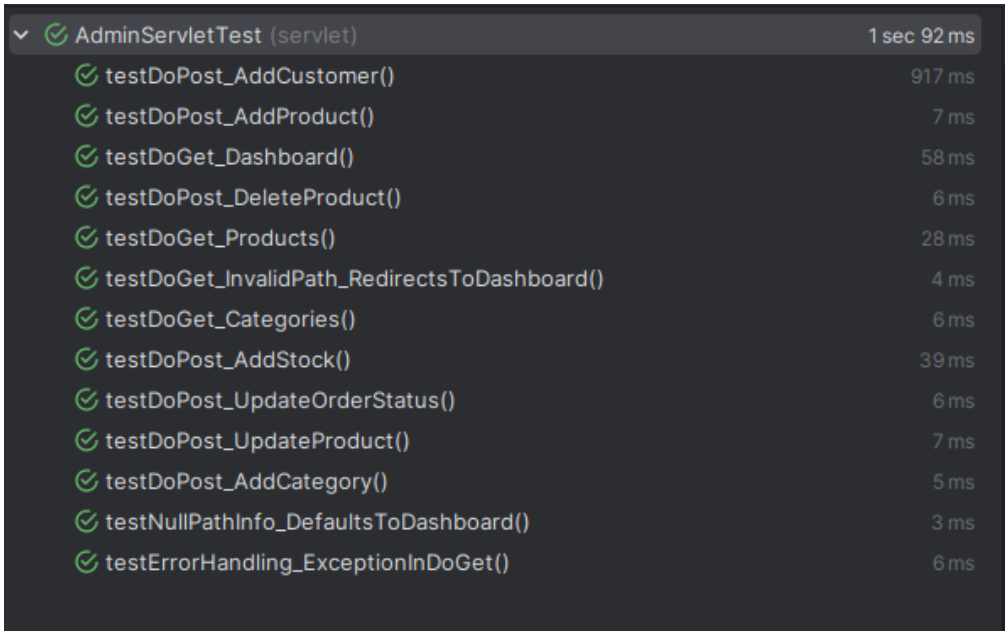


Figure 59 : AdminServletTest

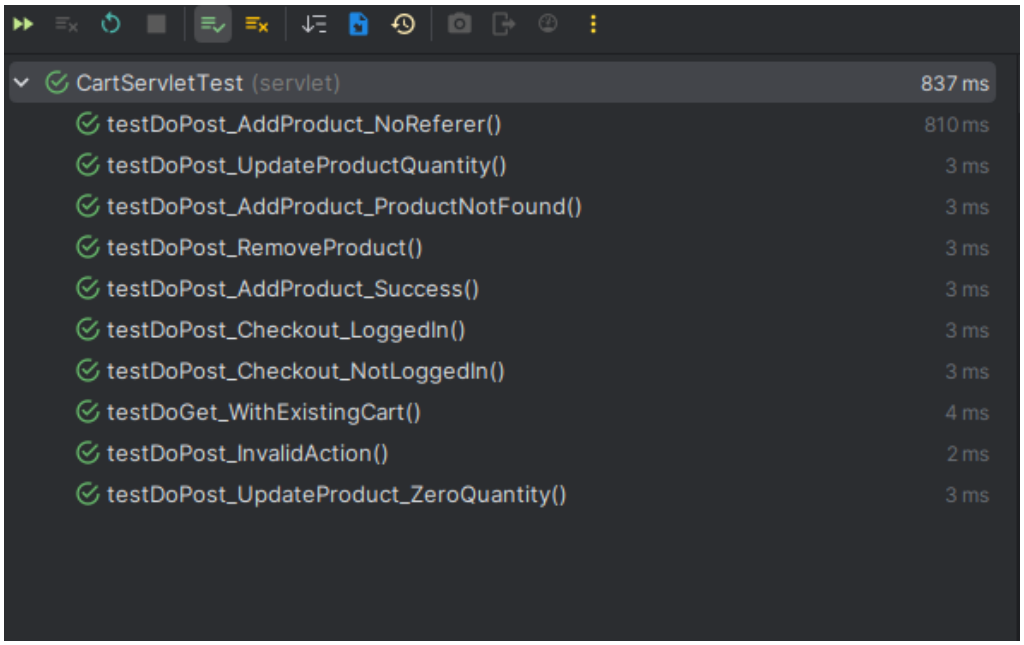


Figure 60 : CartServletTest

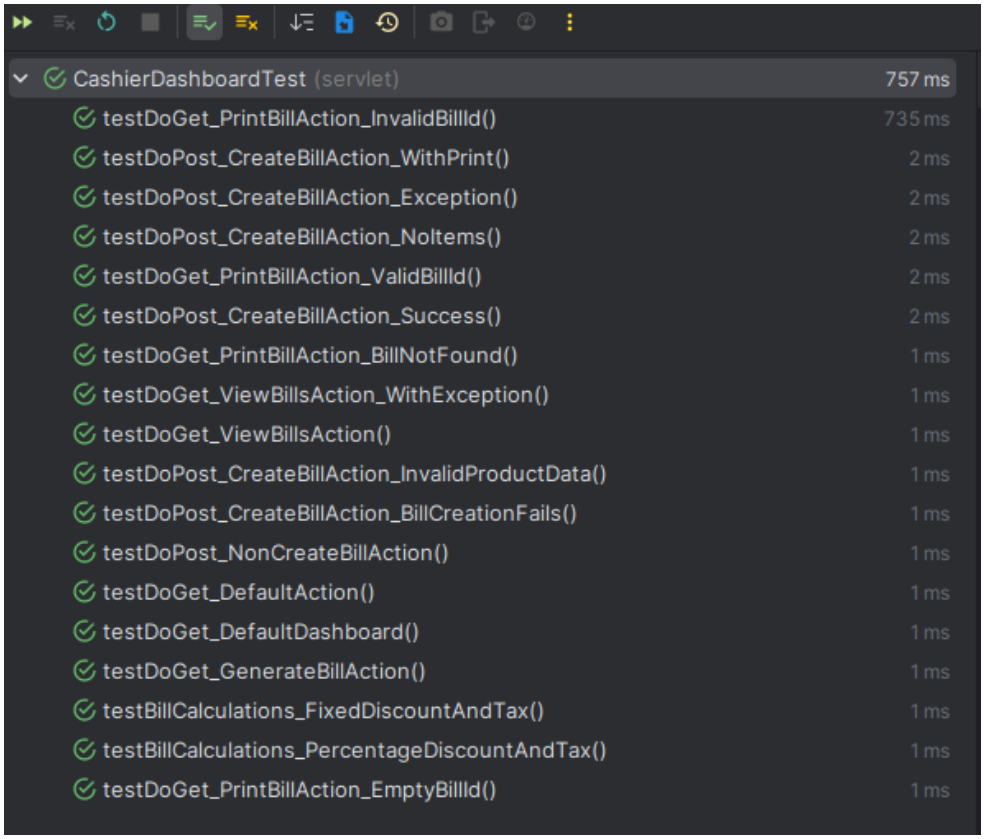


Figure 61 : CashierDashboardTest

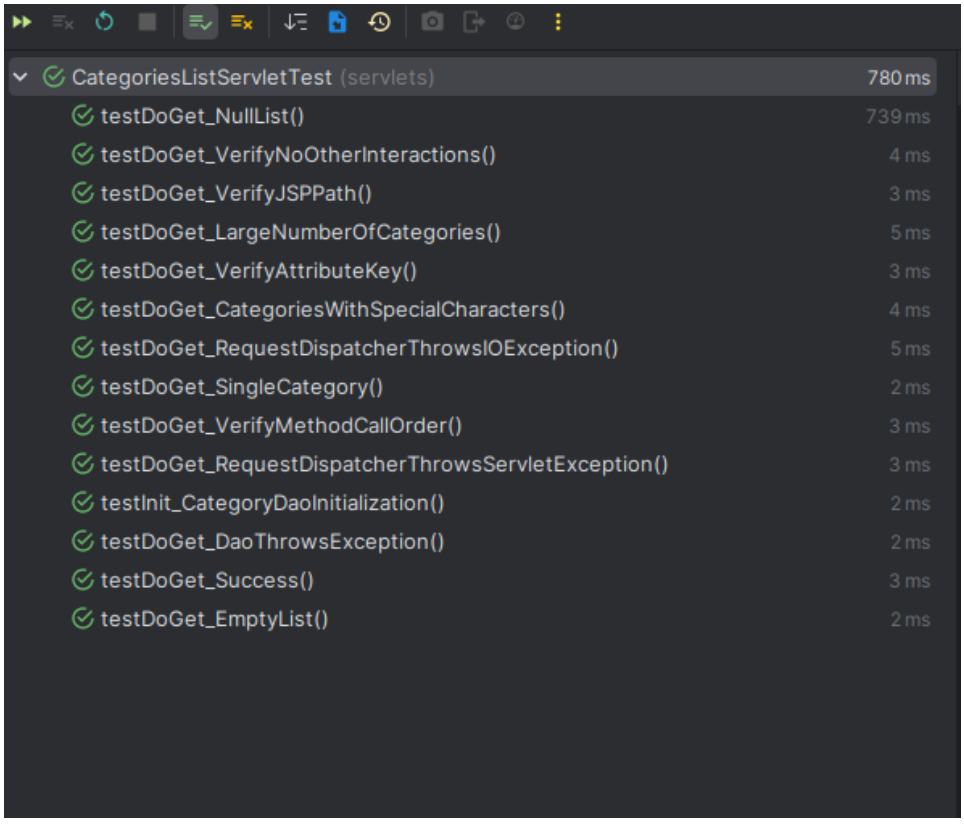


Figure 62 : CategoriesListServletTest

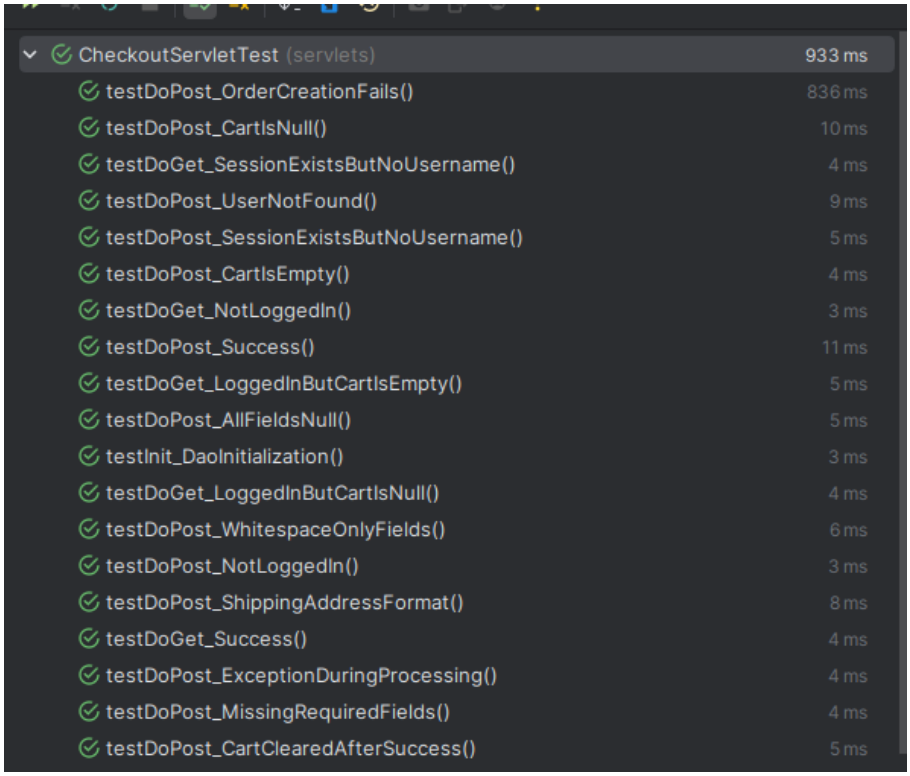


Figure 63 : CheckoutServletTest

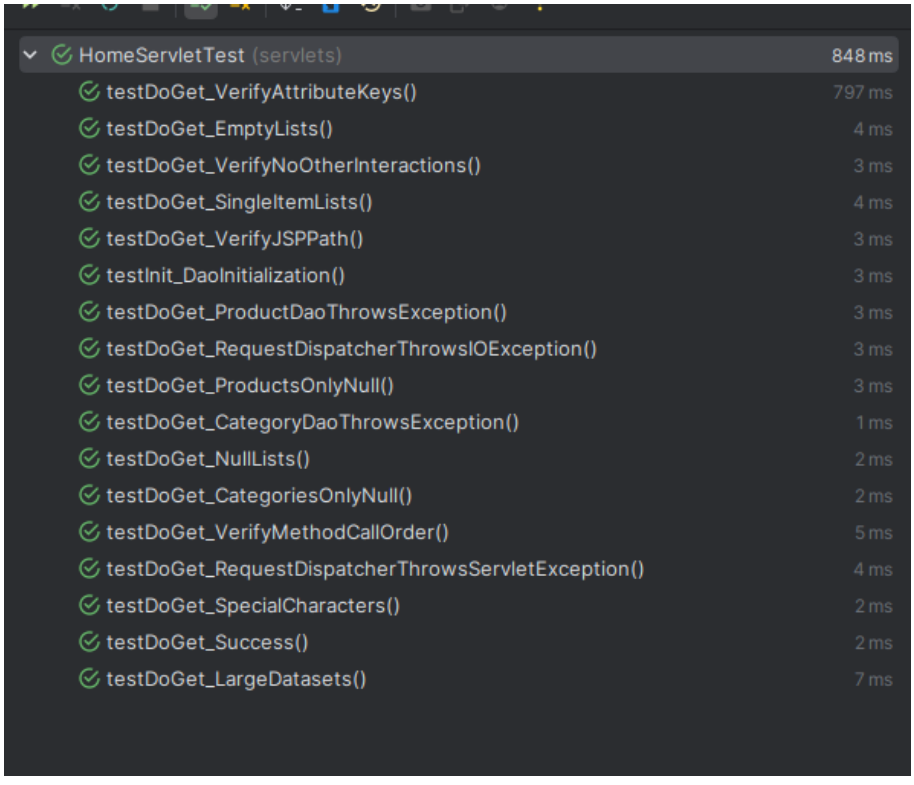


Figure 64 : HomeServletTest

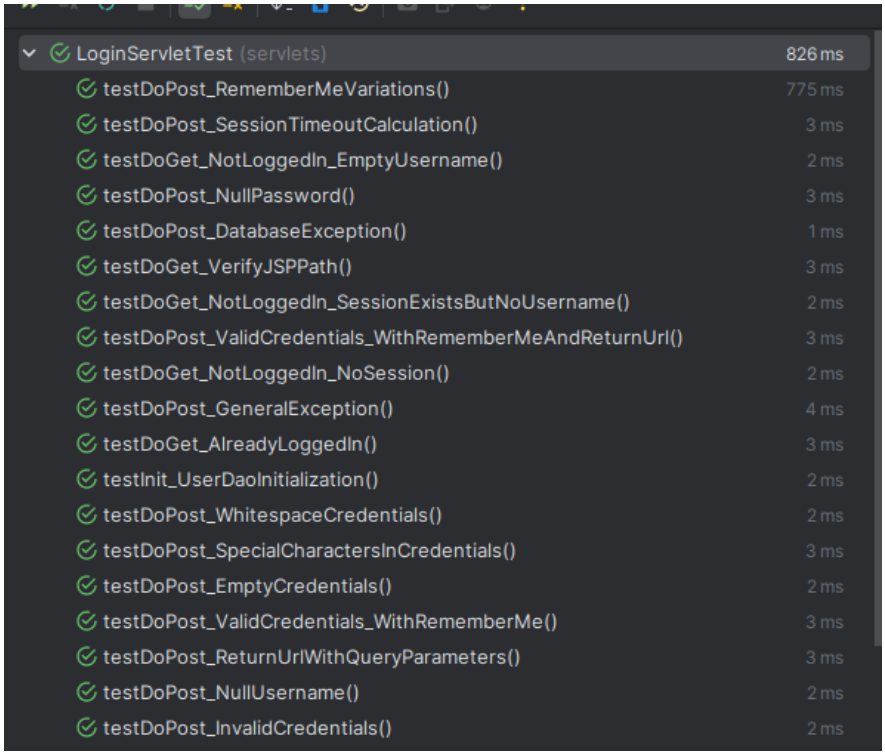


Figure 65 : LoginServletTest

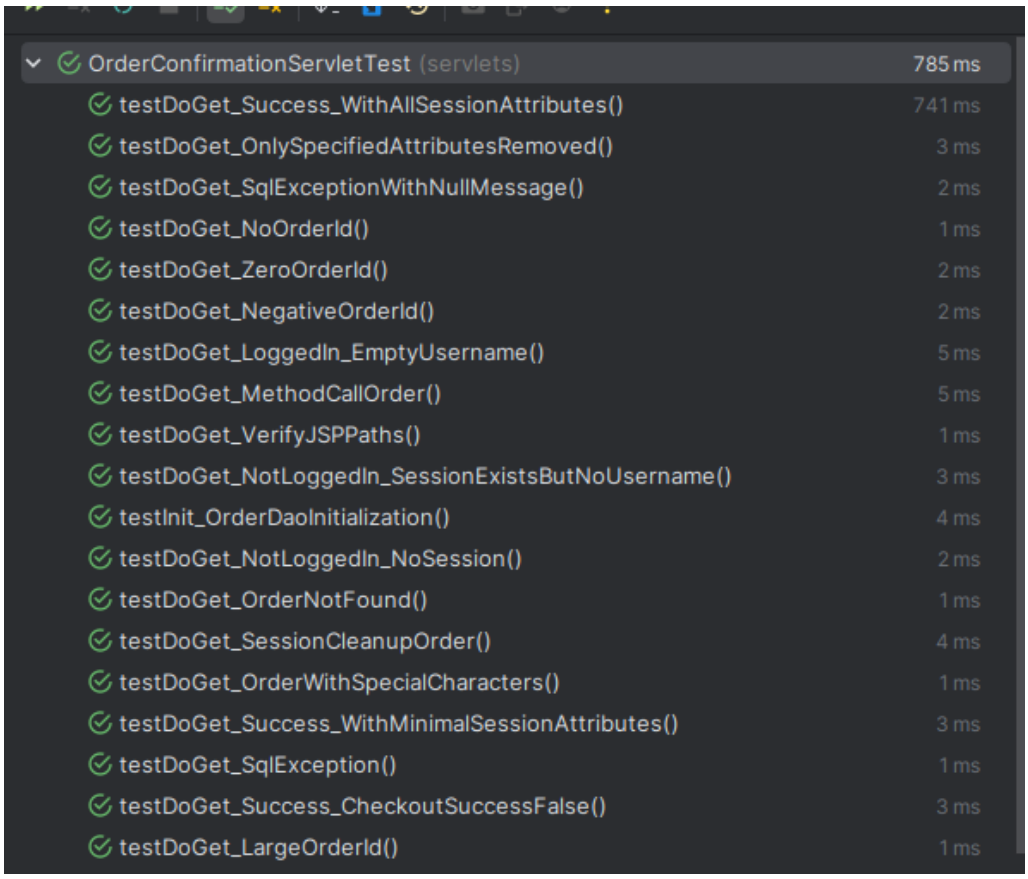


Figure 66 : OrderConfirmationServletTest

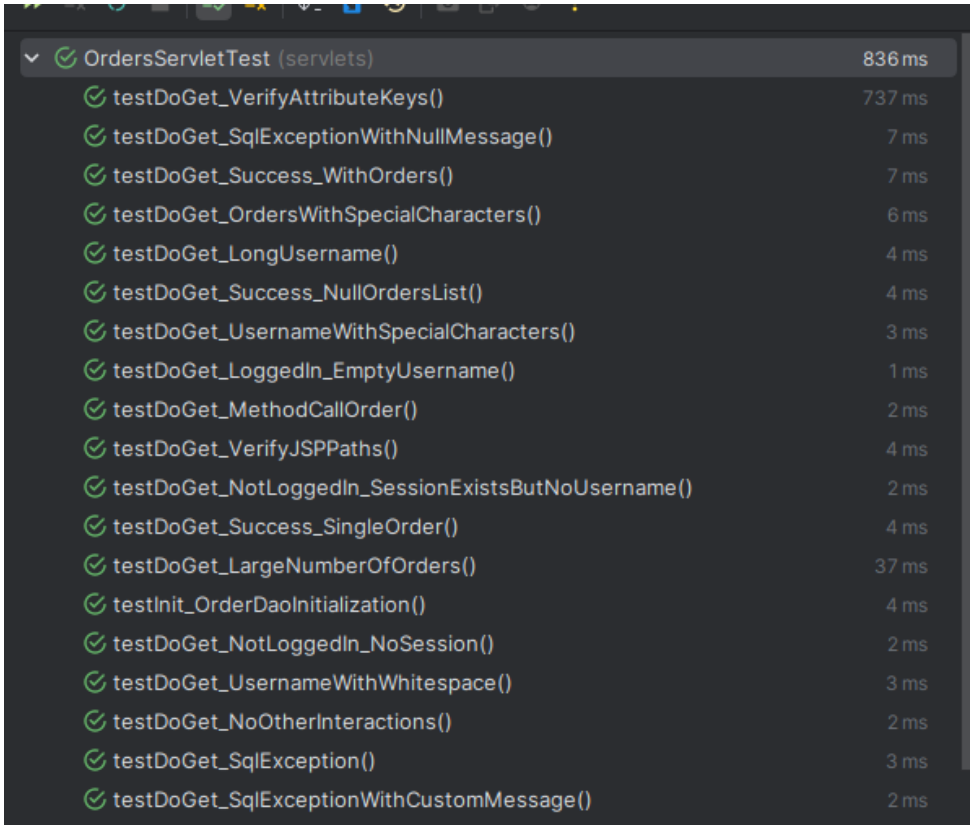


Figure 67 : OrdersServletTest

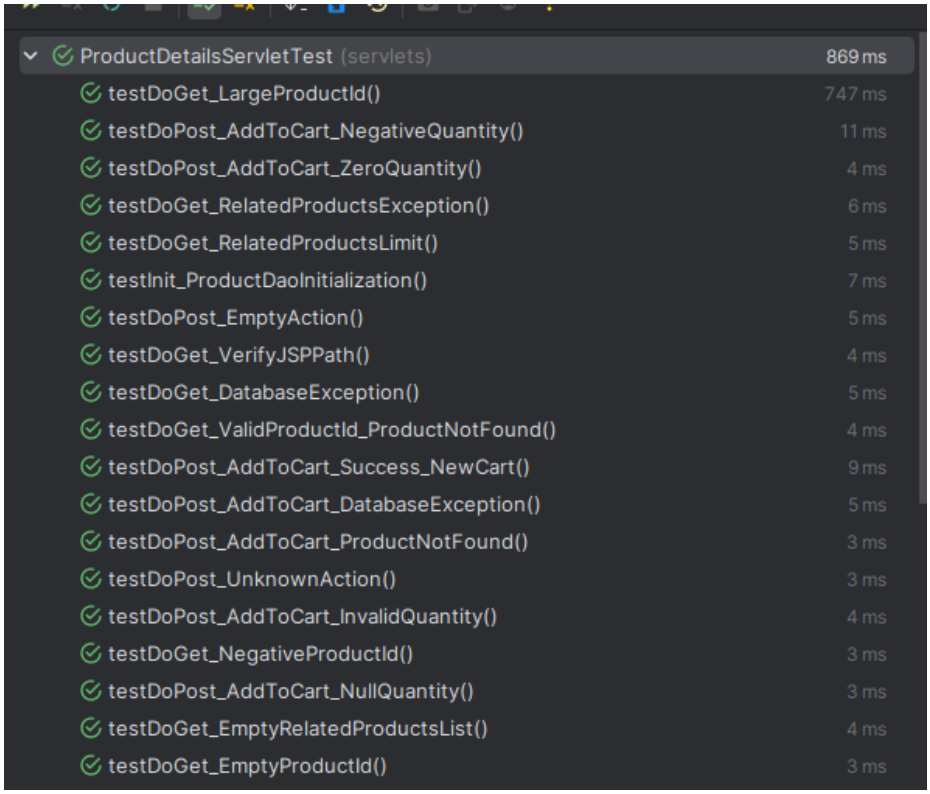


Figure 68 : ProductDetailsServletTest

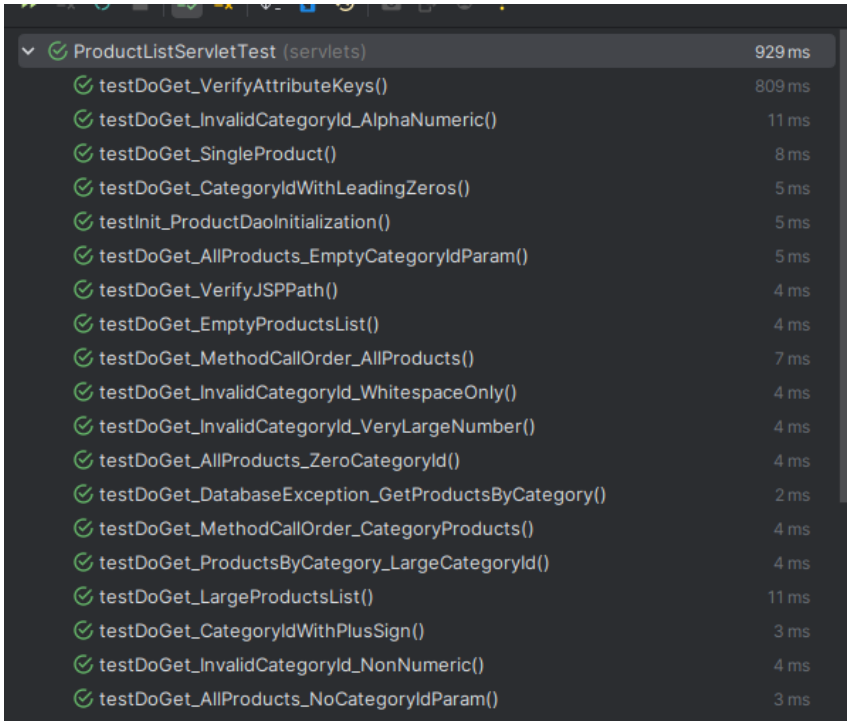


Figure 69 : ProductListServletTest

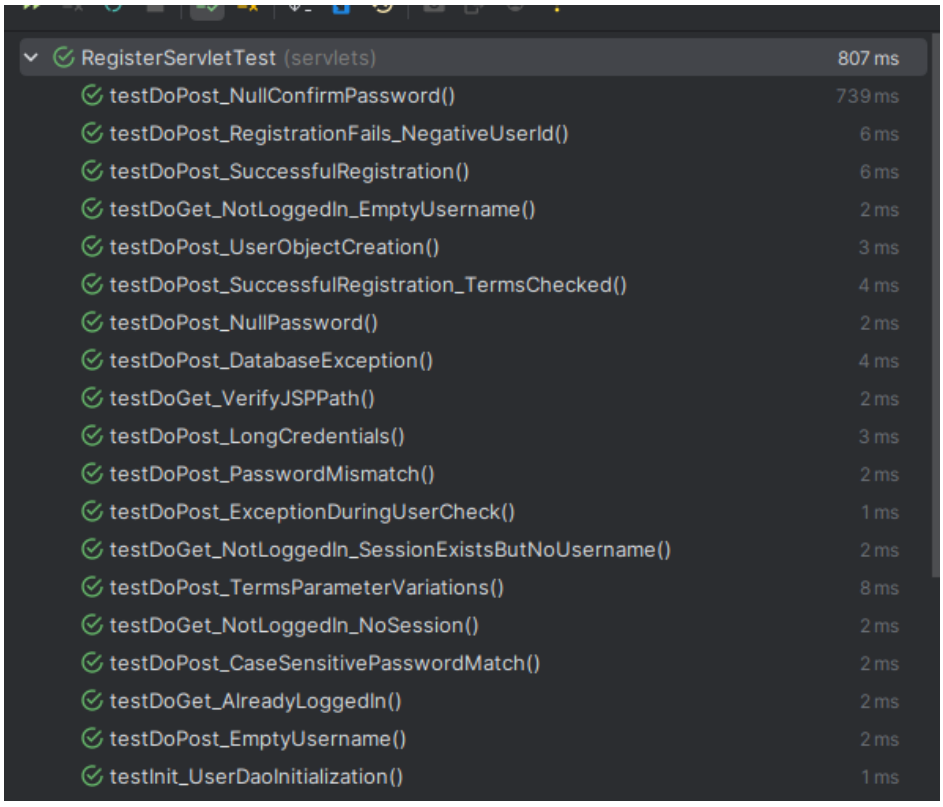


Figure 70 : RegisterServletTest

6. Critical Analysis and Discussion

6.1. Strengths in Clean Architecture Implementation

Passing from a console to a web application while remaining true to Clean Architecture is an excellent example of architectural discipline. Because controllers depend on services and services use repositories, the system is well suited for testing and maintenance. One advantage is that it neatly separates business logic from ways of displaying information which allows both parts to change independently.

this system lets you deploy services on numerous machines for greater scalability. If we strictly separate the presentation layer from the business and data areas, we will be able to change the architecture in the future without much difficulty.

6.2. Concurrency Architecture Analysis

Both the cleverness and the limits of server-side concurrency are well demonstrated in the implementation. Tomcat with a thread pool of 200 and an acceptCount of 100 and supported by Apache DBCP, is the right solution for companies with moderate-scale needs. However, sequential processing is very safe, but it can become very slow when there is a major increase in requests.

Critical Observation - Servlet-based concurrency limits the system's effectiveness in modern, high-traffic scenarios. I/O and event-driven approaches such as Spring WebFlux, have grown popular in today's systems to help use resources optimally.

6.3. User Experience and Interface Design

6.3.1. GUI Implementation Assessment

They made sure that all Admin, Cashier and Customer roles have complete dashboard implementations that optimize their roles and tasks. Because Tailwind CSS supports responsive design, the site will display well on all devices important in modern retail. Strengths Identified,

- Complete CRUD operations across all entities
- Real-time form validation with feedback
- Contextual help systems and notification mechanisms

- Mobile-responsive design patterns

Areas for Enhancement,

- The image of the interface indicates a common design which although practical, lacks use of up-to-date patterns like progressive disclosure or micro-interactions
- While the billing interface is rich with features, it looks complicated and might be easier to use if it was redesigned into a simpler workflow structure

6.4. Technical Implementation Critique

6.4.1. Database and Performance Considerations

It’s surprising that the connection pooling strategy set for an e-commerce platform is so low. Even though using launches this way saves resources, it can cause delays during the highest usage times. This wait time allows users to work well with your app, but it may hide performance problems.

Gaps - There is no discussion in the report about important database strategies such as indexing, query optimization or caching mechanisms. These issues must be handled carefully for platforms managing inventory changes at the same time.

6.4.2. Security and Data Integrity

The report discusses secure authentication and the use of prepared statements to stop SQL injections, but it does not do a thorough security analysis. Current e-commerce applications would need:

- HTTPS enforcement
- CSRF protection
- Input sanitization beyond SQL injection
- Session management security
- Password hashing and storage best practices

There is a critical gap in the analysis because information on security implementation is lacking.

6.5. Testing and Quality Assurance

JMeter Testing Analysis

Apache JMeter allows you to test multiple critical parts of your system for smooth performance. Since the test scenarios used GET and POST requests for many website paths, the system demonstrates good capability to handle many simultaneous tasks.

Testing Strengths,

- Protection for endpoints is completely comprehensive.
- Realistic simulation using more than one user at the same time
- Offering services for both authentication and transaction testing

7. Conclusion

The improved SYOS Point of Sale system proves that Clean Architecture principles work well in developing multi-tier distributed web applications. All the objectives of the assignment are met and a solid base for current e-commerce operations is established by the implementation. Using advanced web development techniques, the GUI is still simple for users to use. With careful thread management and well-optimized database connections, server-side concurrency mechanisms can deal with many clients using the server at one time. By following multi-tier architecture, Clean Architecture can be distributed and the scale of system components can be managed individually. A well-structured test case checks that the system can work with many visitors at the same time and keep all information accurate and reliable. The new system is a better version than the console app and will support growing demands for the business.