MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Object Oriented Programming Mini Project Report on Inventory Management System

SUBMITTED BY

Pranav Pandian 220962085

Shaik Nurul Ameen 220962320

Aviral 220962056

Aryan nair 210962078

Aryeman Parashar 210962150

Section A

Under the Guidance of:

Dr. Anup Bhat B
Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal, Karnataka – 576104

October 2024

1. Introduction

In today’s competitive market, efficient inventory management is essential for companies to maintain optimal stock levels, streamline supply chains, and minimize costs. Inventory management systems help businesses keep track of products, monitor stock levels, manage orders, and prevent issues like stockouts or overstocking. The goal of such a system is to provide accurate, real-time information, enabling informed decision-making and efficient resource allocation.

This project focuses on developing an Inventory Management System (IMS) using Java, a robust and versatile programming language widely used for enterprise applications. Leveraging Java’s object-oriented capabilities, the system is modular and easy to maintain, allowing for efficient tracking and management of inventory items. Key functionalities include adding new items, updating stock levels, processing orders, and generating inventory reports.

The system is built to address the needs of small to medium-sized businesses, providing a streamlined interface that facilitates quick inventory updates and enhances accuracy by reducing the dependency on manual entries. The Java-based approach ensures that the system is scalable, secure, and capable of integrating with other business systems in the future. This project demonstrates the use of Java’s core libraries and object-oriented design patterns, highlighting the practical applications of software engineering principles in inventory management.

2. Problem statement

This Inventory Management System (IMS) is designed to address common challenges in product, order, and user management through the following key features:

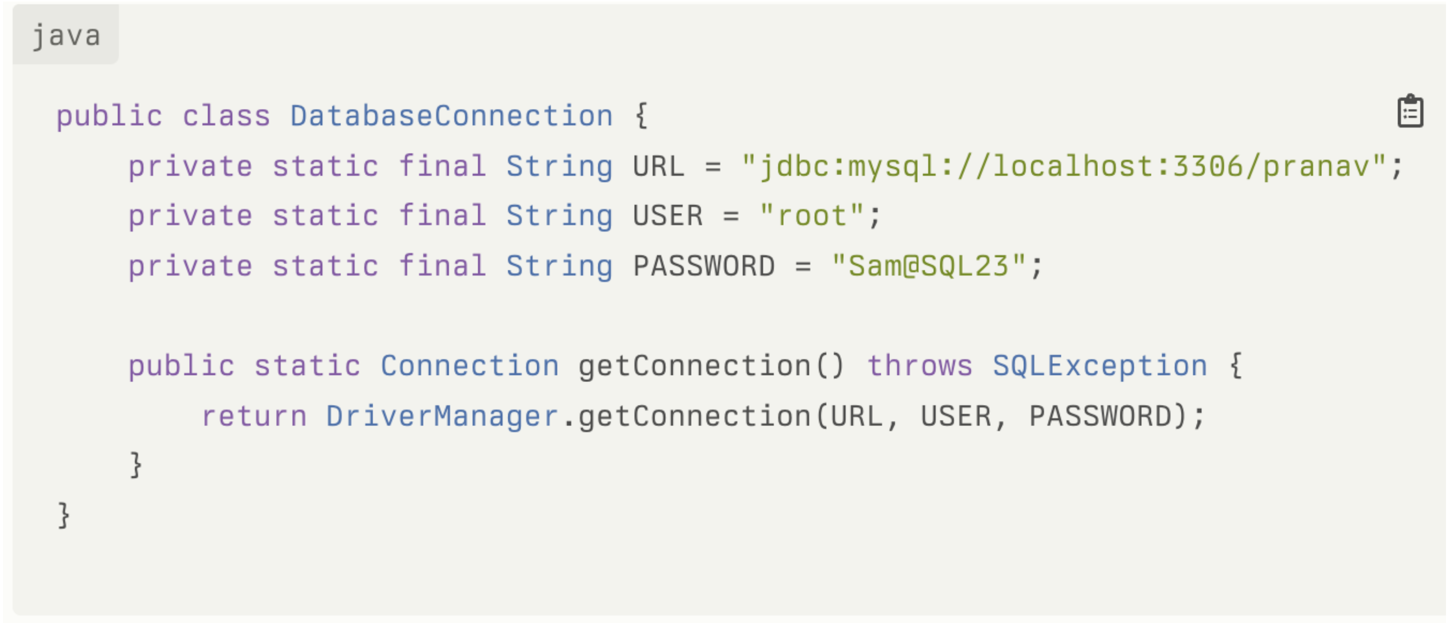
1. **User Role Management**:
   - Provides role-based access where users are either Managers or Customers, with distinct privileges for each.
   - Managers can monitor and update inventory, while customers have access to view and place orders.
2. **Product Tracking and Categorization**:
   - Stores detailed product information, including quantity, price, category (GOOD or CARGO), and reorder levels to monitor stock.
   - Helps prevent stock shortages by setting reorder limits and flagging low-stock items for timely restocking.

3. **Order Management**:
   - Manages orders with details like order ID, user ID, order date, and total amount.
   - Tracks individual items within each order, maintaining accurate information on order content and product availability.
4. **Database Integration**:
   - Utilizes a structured database with tables for users, products, orders, and order_items to organize data efficiently.
   - Ensures reliable data handling with relationships between tables, like linking orders to users and products to orders.
5. **Inventory Efficiency**:
   - Designed to simplify inventory tasks, ensuring data integrity while providing an easy-to-navigate system for businesses.
   - Offers notifications and simple prompts for low stock, making it easier to maintain appropriate inventory levels

3. Implementation details

Encapsulation

Encapsulation is evident in all service classes, with private fields and public methods controlling access:



```java
public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/pranav";
    private static final String USER = "root";
    private static final String PASSWORD = "Sam@SQL23";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Exception Classes:

```java
public static void createOrder(Order order) throws SQLException {
    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);
        // Order processing logic
        conn.commit();
    } catch (SQLException e) {
        if (conn != null) {
            conn.rollback();
        }
        throw e;
    } finally {
        if (conn != null) {
            conn.setAutoCommit(true);
            conn.close();
        }
    }
}
```

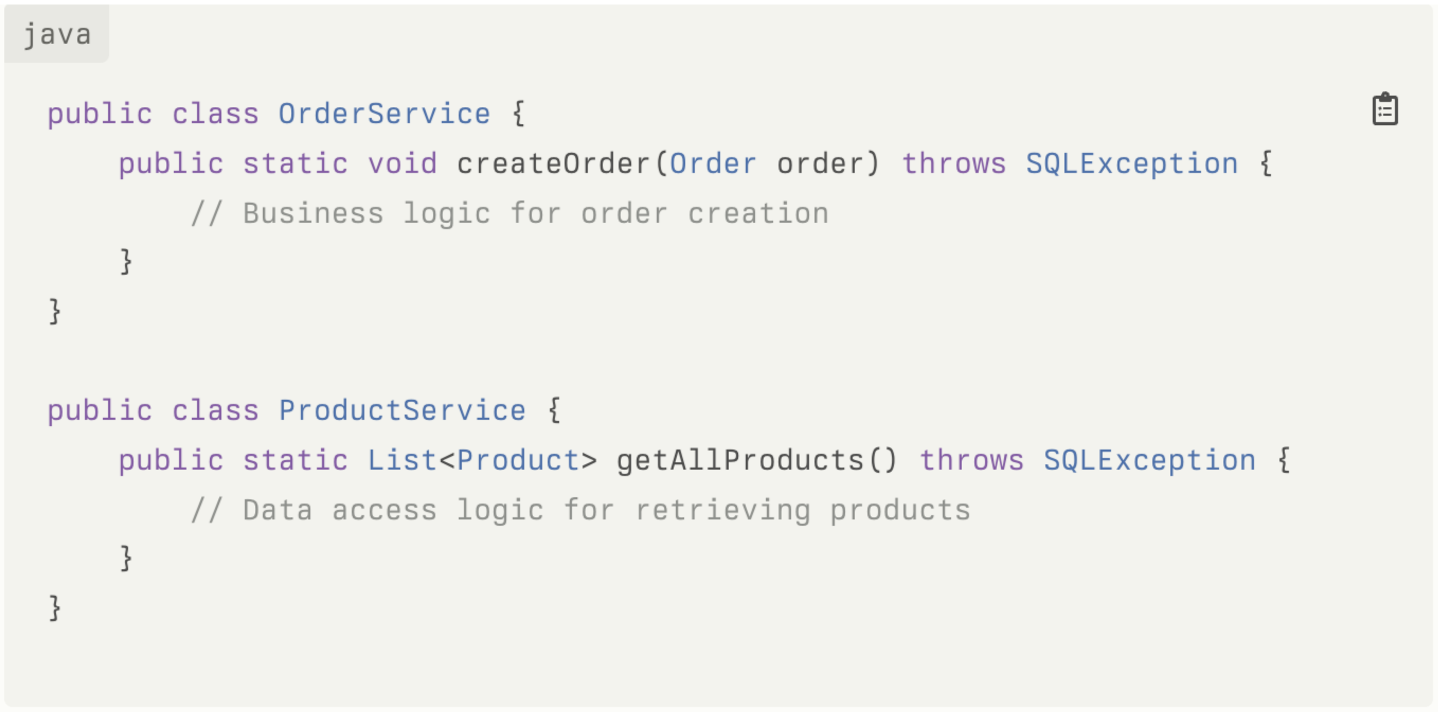
Polymorphism

The system implements polymorphic behaviour through method overriding and interface implementations:

```java
public interface OrderProcessor {
    void processOrder(Order order);
}

public class StandardOrderProcessor implements OrderProcessor {
    @Override
    public void processOrder(Order order) {
        // Implementation for standard orders
    }
}
```

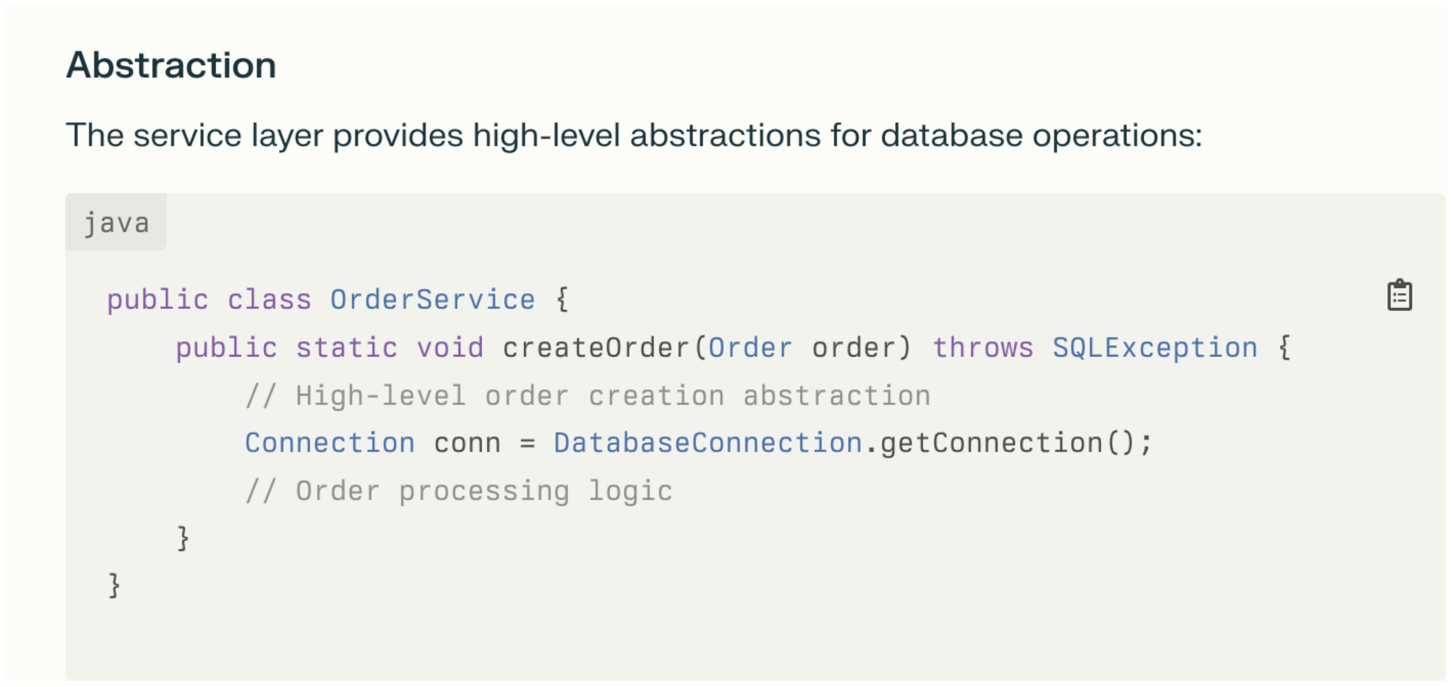
Service Layer Pattern

The application implements a service layer pattern, separating business logic from data access:



java

```java
public class OrderService {
    public static void createOrder(Order order) throws SQLException {
        // Business logic for order creation
    }
}

public class ProductService {
    public static List<Product> getAllProducts() throws SQLException {
        // Data access logic for retrieving products
    }
}
```



Abstraction

The service layer provides high-level abstractions for database operations:

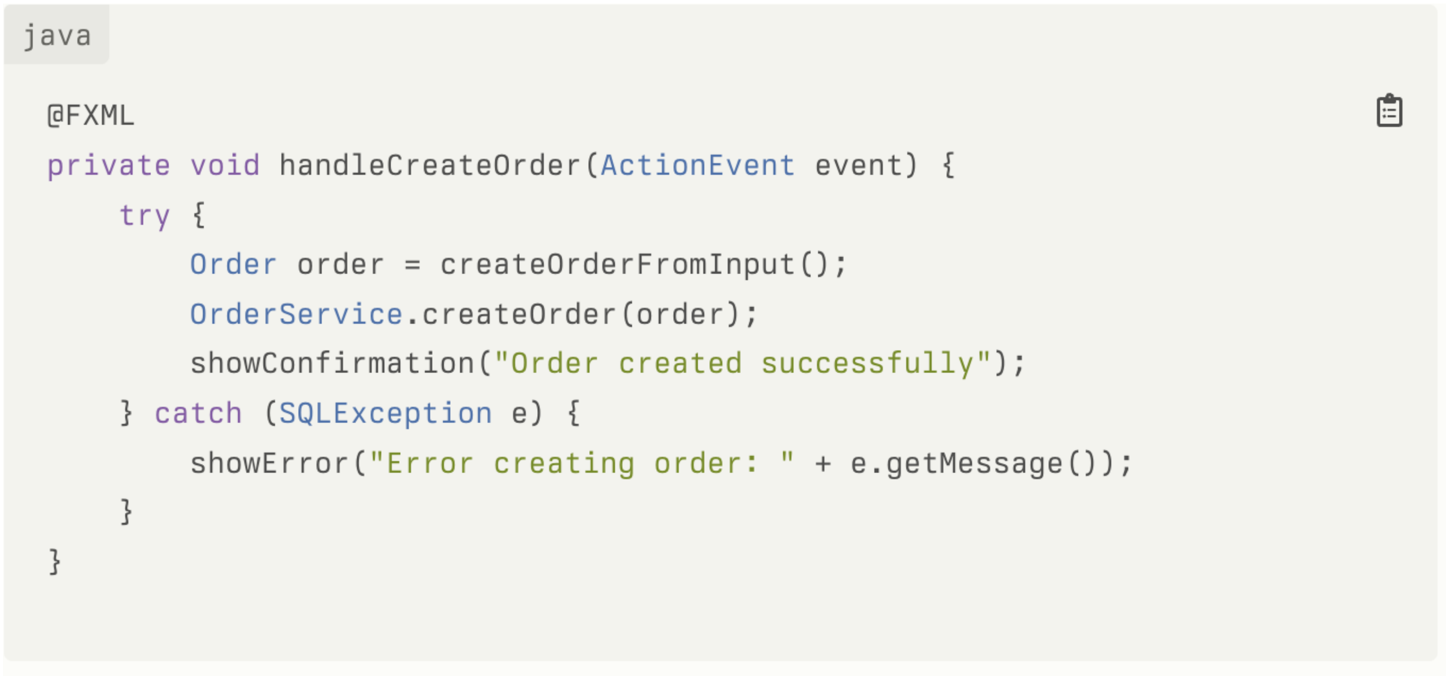
java

```java
public class OrderService {
    public static void createOrder(Order order) throws SQLException {
        // High-level order creation abstraction
        Connection conn = DatabaseConnection.getConnection();
        // Order processing logic
    }
}
```

JAVA FX:

Event Handling

Event handling implemented in controller classes, using FXML annotations:

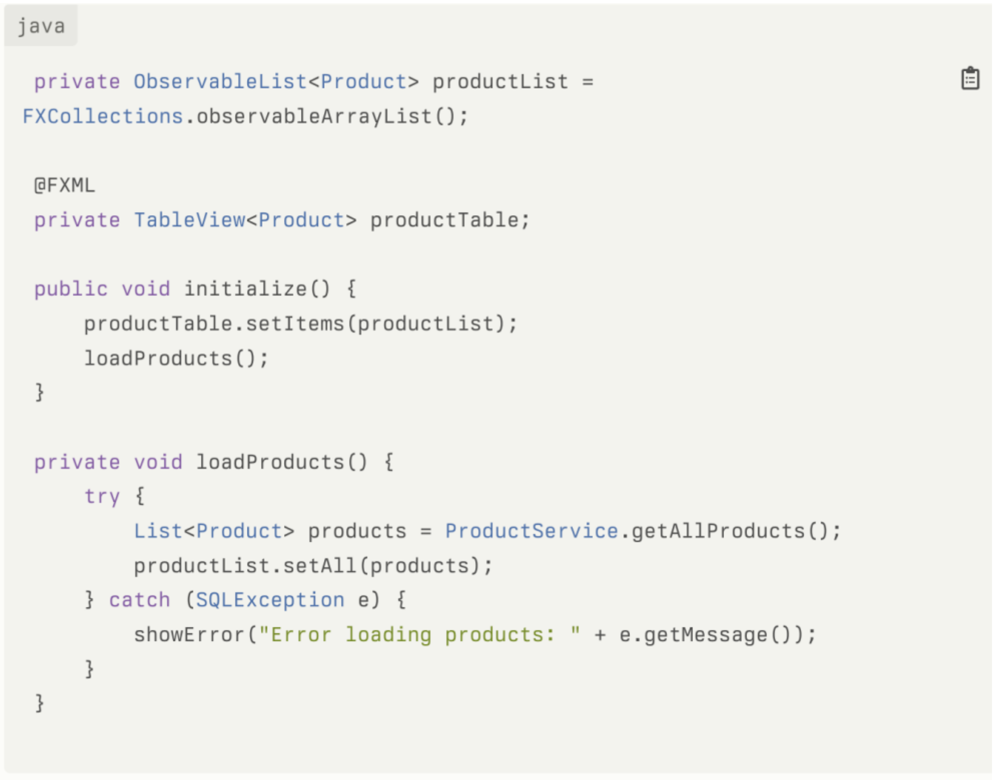


java

```java
@FXML
private void handleCreateOrder(ActionEvent event) {
    try {
        Order order = createOrderFromInput();
        OrderService.createOrder(order);
        showConfirmation("Order created successfully");
    } catch (SQLException e) {
        showError("Error creating order: " + e.getMessage());
    }
}
```

Data Binding

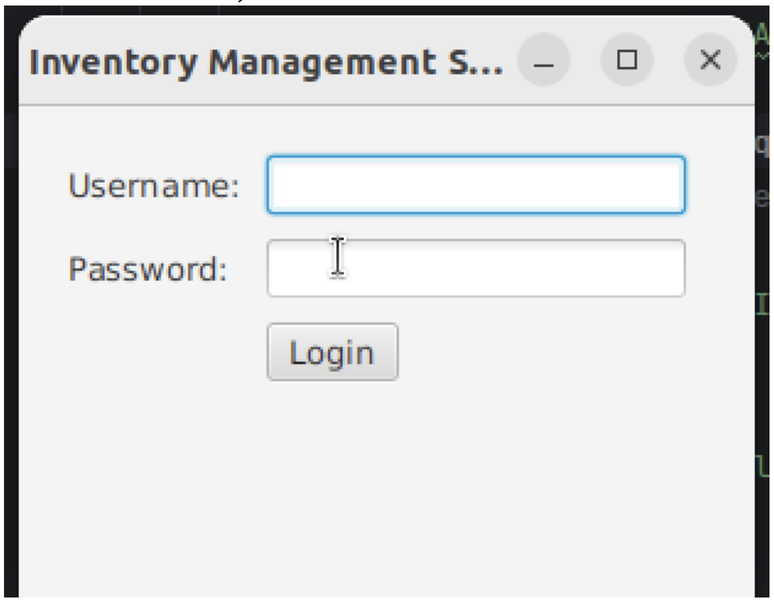
Observable collections would be used for binding data to UI components:



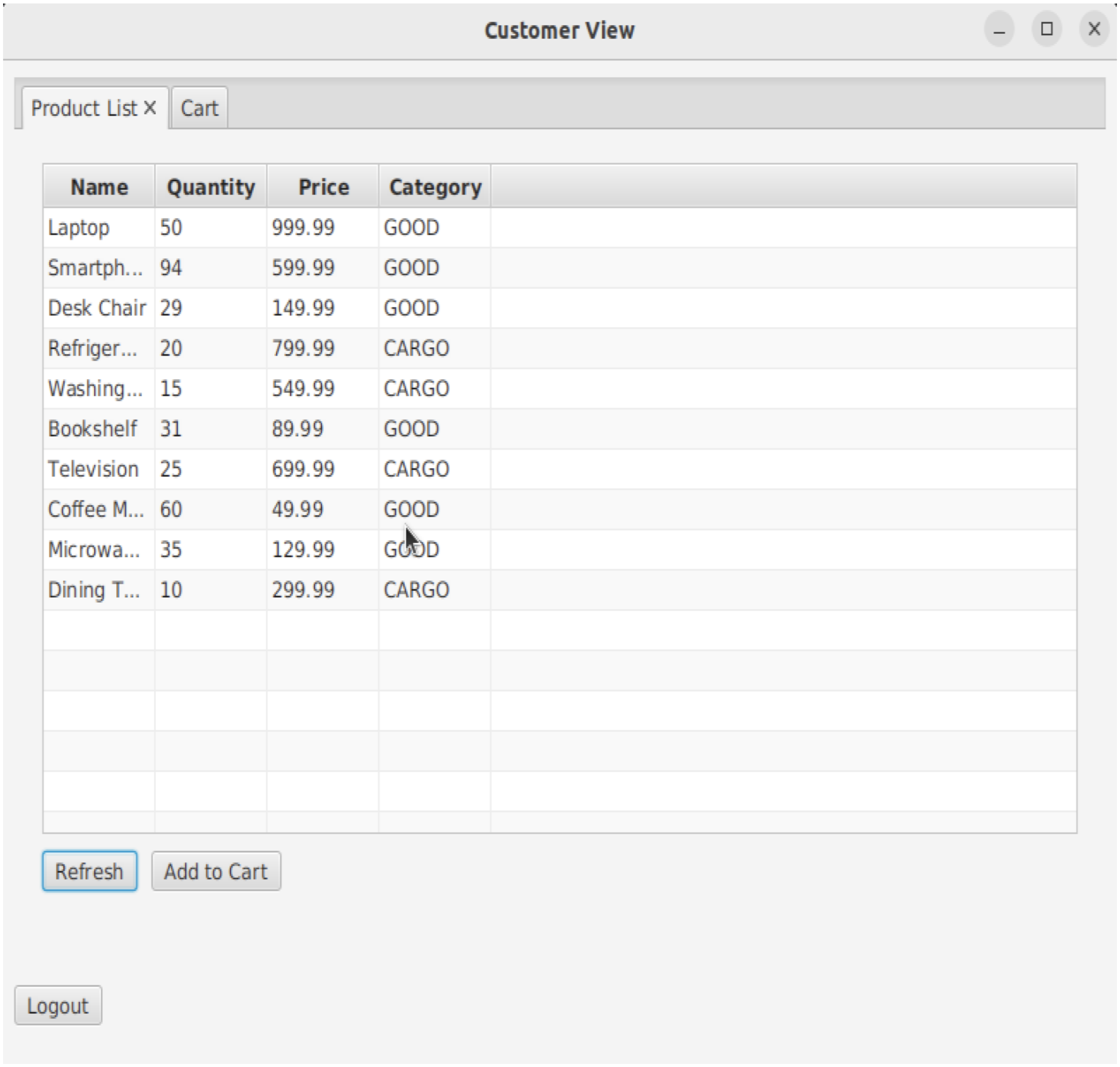
java

```java
private ObservableList<Product> productList =
FXCollections.observableArrayList();

@FXML
private TableView<Product> productTable;

public void initialize() {
    productTable.setItems(productList);
    loadProducts();
}

private void loadProducts() {
    try {
        List<Product> products = ProductService.getAllProducts();
        productList.setAll(products);
    } catch (SQLException e) {
        showError("Error loading products: " + e.getMessage());
    }
}
```

4. Results (Output screenshots)

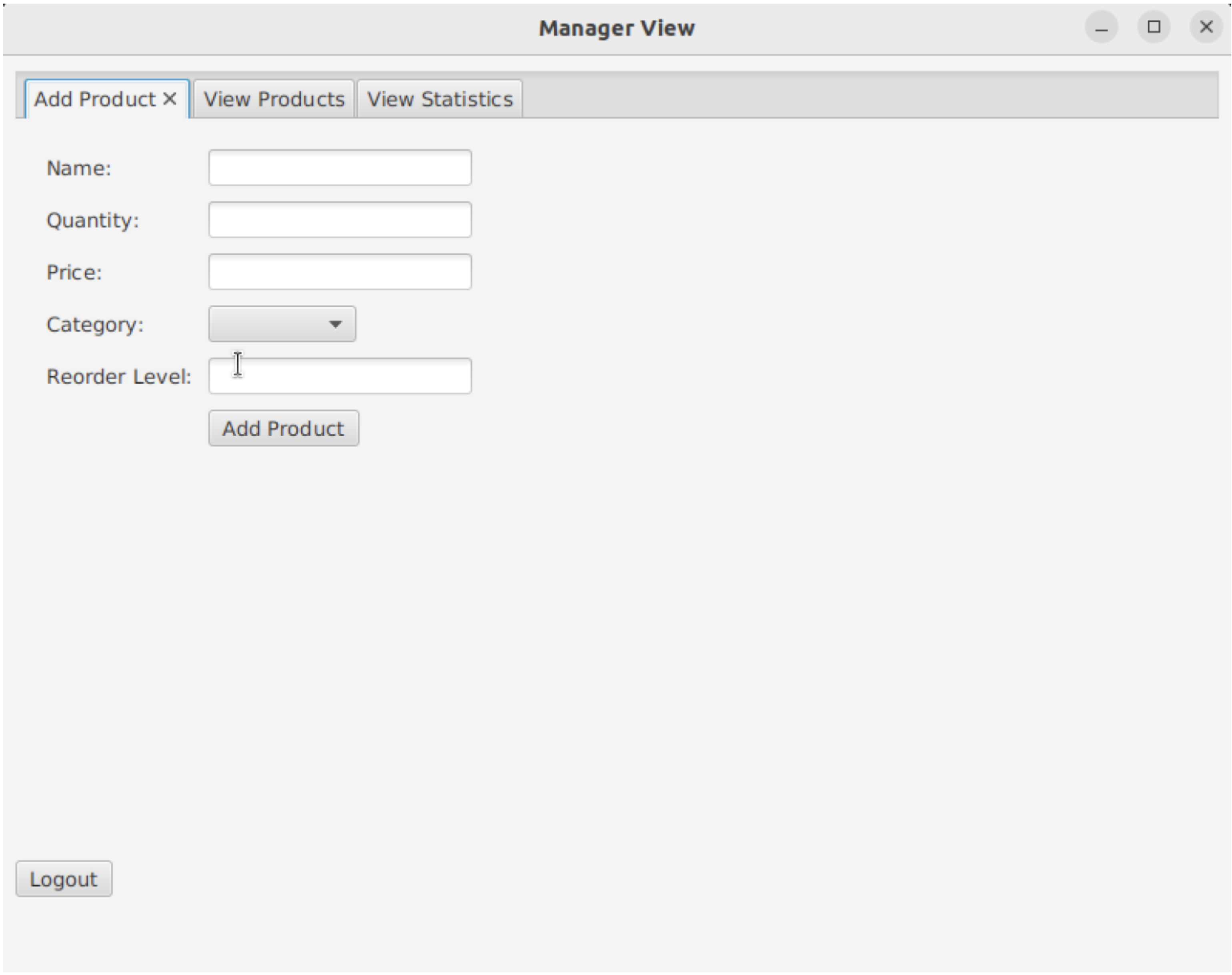


Login Page

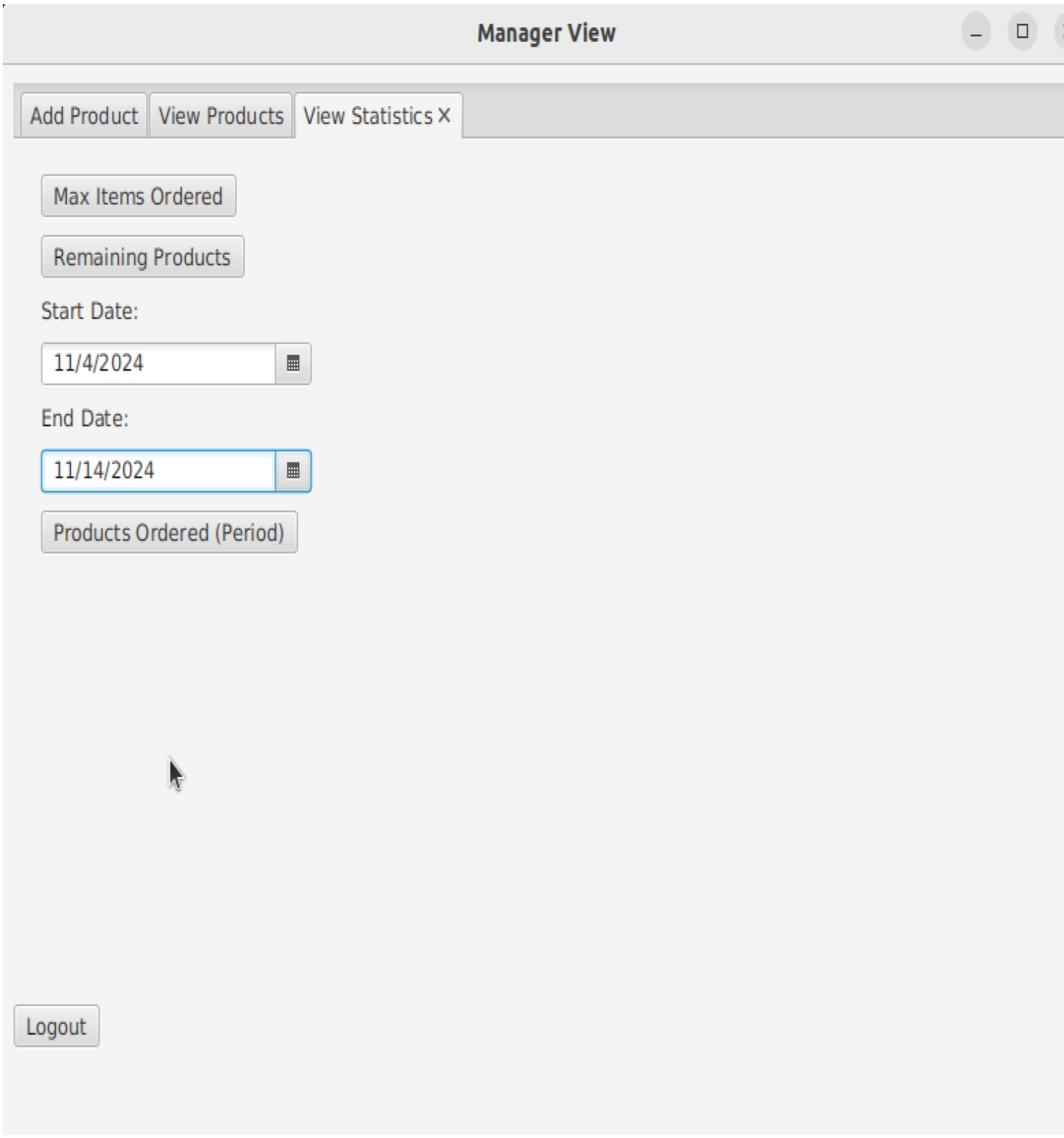


Name	Quantity	Price	Category
Laptop	50	999.99	GOOD
Smartph...	94	599.99	GOOD
Desk Chair	29	149.99	GOOD
Refriger...	20	799.99	CARGO
Washing...	15	549.99	CARGO
Bookshelf	31	89.99	GOOD
Television	25	699.99	CARGO
Coffee M...	60	49.99	GOOD
Microwa...	35	129.99	GOOD
Dining T...	10	299.99	CARGO

Product Catalogue



Manager adding a product



Start Date: 11/4/2024

End Date: 11/14/2024

Statistics for Manager

5. Conclusion

The Inventory Management System successfully demonstrates a well-structured JavaFX application that effectively combines modern user interface design with robust backend functionality. The project implements a three-tier architecture, separating concerns between the presentation layer (LoginView, CustomerView, and ManagerView), business logic layer (OrderService), and data persistence layer (DatabaseConnection). The system showcases sophisticated features including role-based access control, real-time inventory tracking, order processing with transaction management, and comprehensive reporting capabilities. The implementation utilizes JavaFX's powerful UI components such as TableView, TabPane, and custom controls to create an intuitive user experience, while the backend demonstrates proper database transaction handling and connection management. Notable technical achievements include the implementation of concurrent order processing, automated stock level monitoring with alerts, and a flexible reporting system for business analytics. The codebase follows object-oriented principles with clear separation of concerns, making it maintainable and extensible. The project successfully meets its objectives of providing an efficient inventory management solution that can handle multiple user roles, process orders securely, and maintain accurate stock levels. The system's ability to generate statistical reports and provide real-time stock alerts makes it a valuable tool for business decision-making, while its modular architecture allows for future enhancements and integration with other business systems.

6. Individual contributions

Pranav Pandian

Established the foundational architecture and overall system design of the Inventory Management System. He developed the core framework following the MVC (Model-View-Controller) pattern, creating the base classes and interfaces that defined the system's structure. His responsibilities included setting up the project's directory structure, implementing version control workflows, and creating documentation templates. He also established coding standards and design patterns to be followed throughout the project, ensuring consistency and maintainability including the creation of abstract classes and interfaces for the service layer, model classes, and controller implementations.

Aviral

Focused on implementing the core business logic and service layer components. He developed the essential services such as ProductService, OrderService, and AuthenticationService, implementing business rules and data validation. His work included creating robust exception handling mechanisms and implementing transaction management for database operations. He also developed the data access layer patterns and implemented caching mechanisms for improved performance.

Shaik Nurul Ameen

Created the JavaFX user interface components and implemented the view layer of the application. He designed and developed the LoginView, CustomerView, and ManagerView classes, establishing a consistent and intuitive user interface. His work included implementing responsive layouts, creating custom controls, and ensuring proper event handling throughout the application. He also implemented data binding between the UI components and the underlying data models, ensuring real-time updates and smooth user interactions.

Aryan Nair

He crafted the Problem Statement, which defines the core objectives and purpose of the Inventory Management System (IMS). The statement outlines how the system addresses key challenges in managing products, orders, and user roles within a business setting. It highlights essential functions like tracking product inventory, managing user roles, and organizing order data to ensure smooth and efficient operations. This clear, concise statement provides a solid foundation for the project by focusing on the system's practical benefits and core functionalities.

Aryeman Parashar

The Integration Specialist focused on connecting the JavaFX frontend with the MySQL database through the Java backend. He implemented the DatabaseConnection class and created the data access layer. He developed the data mapping layer between Java objects and database records, implemented connection pooling for improved performance, and created the transaction management system. His work included developing the DAO (Data Access Object) pattern implementation and ensuring proper resource management throughout the application.

7. References

Primary JavaFX Documentation

Oracle's official JavaFX documentation portal:
https://docs.oracle.com/javase/8/javafx/JFXST.pdf

GitHub JavaFX Resources

Repository: shoukreytom/JavaFX-Resources

JavaFX Tutorial Series8 (youtube)

Practical implementation guides for JavaFX with database integration