Online Store System (using JavaFX + Spring Boot)

Qi He, Shao Hang Li, Ritu Arora

Project Overview

Purpose: Simplify e-commerce operations for users and admins.

Problem Solved: Manual order tracking, no personalized search, and limited notification mechanisms.

Target Users: Online shoppers

Online Store System

A desktop-based e-commerce system with full product, cart, payment, and order management

Online Store System

Built with JavaFX (Frontend) & Spring Boot (Backend)

Frontend:

- JavaFX (FXML + CSS for UI)
- MVC architecture

Backend:

- Spring Boot (REST APIs)
- Spring Data JPA
- Azure SQL Server

Others:

- Maven (Build Tool)
- Microsoft SQL Server on Azure
- Singleton Pattern for Cart Manager

System Architecture

3-Tier Architecture

- **Presentation Layer**: JavaFX GUI (FXML Views + Controllers)
- **Business Logic Layer**: Spring Services (Product, Cart, Payment)
- **Data Access Layer**: JPA Repositories

MVC in JavaFX for clean separation of concerns

Key Features

- Product Browsing by Category and Price Filter
- Persistent Cart (In-memory + GUI Integration)
- Payment Module (Credit Card & PayPal simulation)
- Order & Payment History View
- Product Reviews Support
- Admin CRUD Operations (via REST)

GUI Overview

JavaFX Screens Implemented:

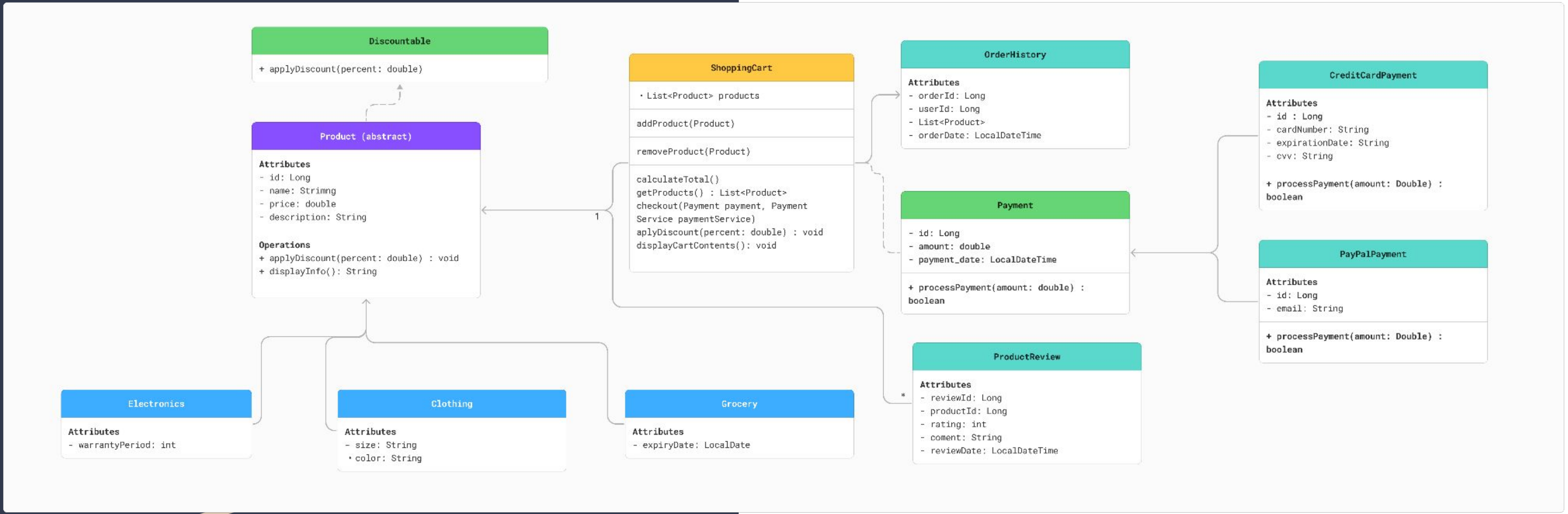
- Login Page
- Product Listing View
- Shopping Cart View
- Checkout & Payment Page
- Order History View
- Admin Dashboard (Optional)

Challenges & Learnings

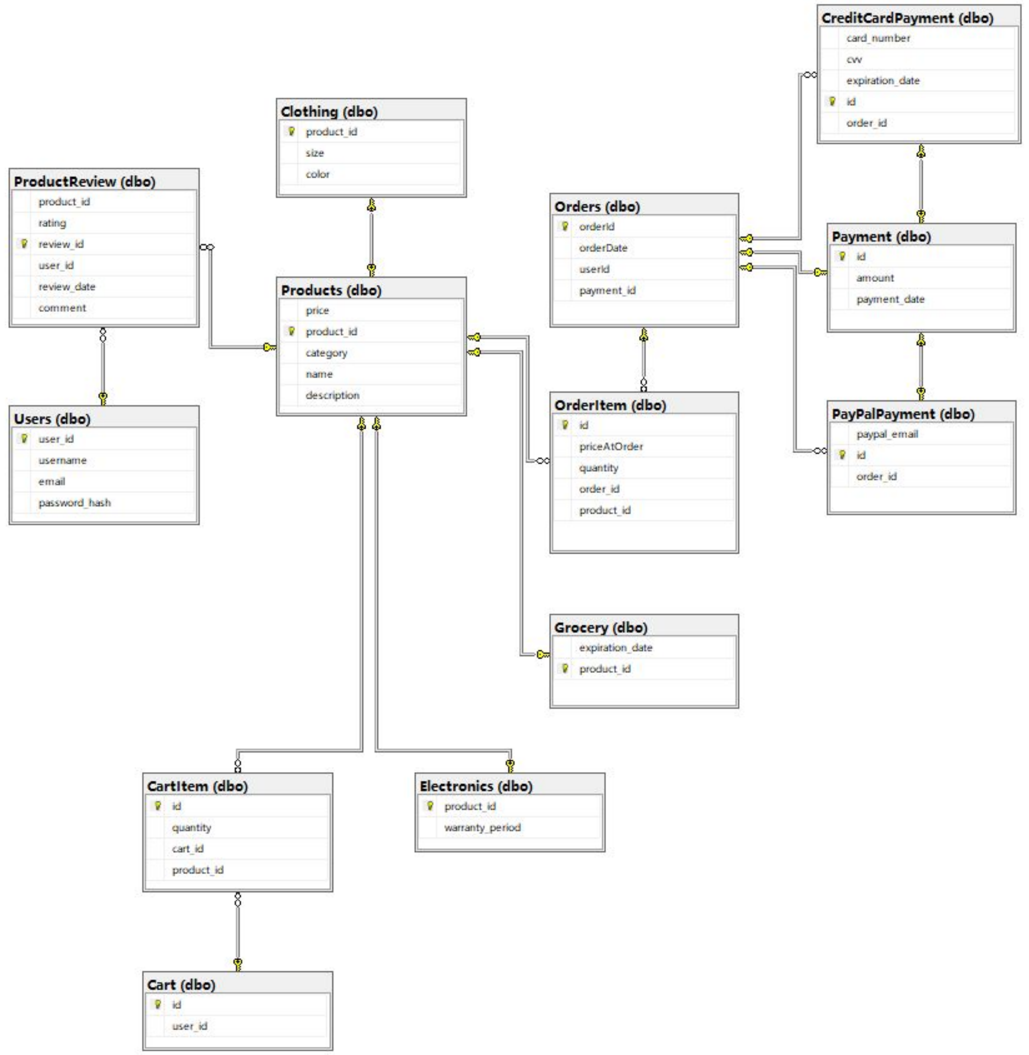
- Integrated JavaFX GUI with Spring Boot APIs
- Used Azure SQL for production-ready persistence
- Implemented filtering state preservation in UI
- Learned real-world use of Singleton, MVC, 3-tier, and REST principles

UML Class Diagram

UML diagram



Entity Relationship Diagram



In-Memory Shopping Cart

How data flows from UI to memory during a shopping session

What is an In-Memory Cart?

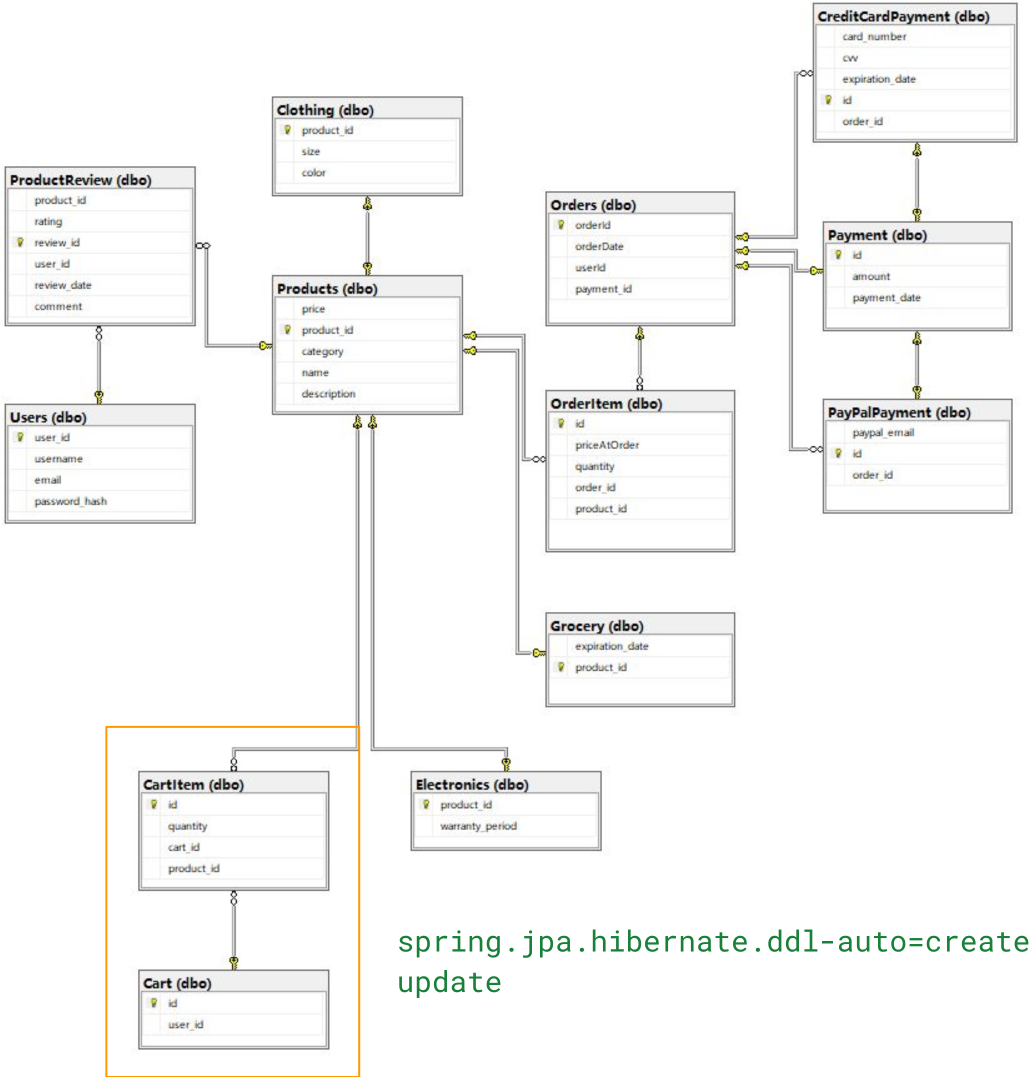
Stores cart data in RAM only - not saved to a database

Managed using a Singleton class (Cart Manager)

Cart items are kept as a list of CartItem objects

Reset on app close (non-persistent)

Entity Relationship Diagram



`spring.jpa.hibernate.ddl-auto=create` or `update`

Architecture & Flow

Spring Boot API → JavaFX ProductController

↓

CartManager (Singleton)

↓

CartController + CartView

Products loaded from backend

Added to cart via “Add to Cart” button

Displayed in Cart View using TableView<CartItem>

Understanding Singleton Pattern

- Ensures only **one instance** of CartManager exists throughout the app
- Provides **global access** to that single instance via getInstance()
- Prevents inconsistent cart states or duplicate cart objects
- Ideal for managing **shared state** in desktop applications (like cart items)

```
public class CartManager {
    private static CartManager instance = new CartManager();
    public static CartManager getInstance() {
        return instance;
    }
}
```

Code Highlights

Add to Cart:

```
CartManager.getInstance().addToCart(product);
```

CartManager Logic:

```
if (item.getProduct().getId() == product.getId()) {
    item.setQuantity(item.getQuantity() + 1);
} else {
cartItems.add(new CartItem(product, 1));
}
```

Cart Table in View:

```
cartTable.setItems(FXCollections.observableArrayList(CartManager.getInstance().getCartItems()));
```

Benefits of In-memory Cart & Future Improvements

Benefits:

- Simple, fast, no DB setup needed
- Keeps GUI code clean and testable

Future Enhancements:

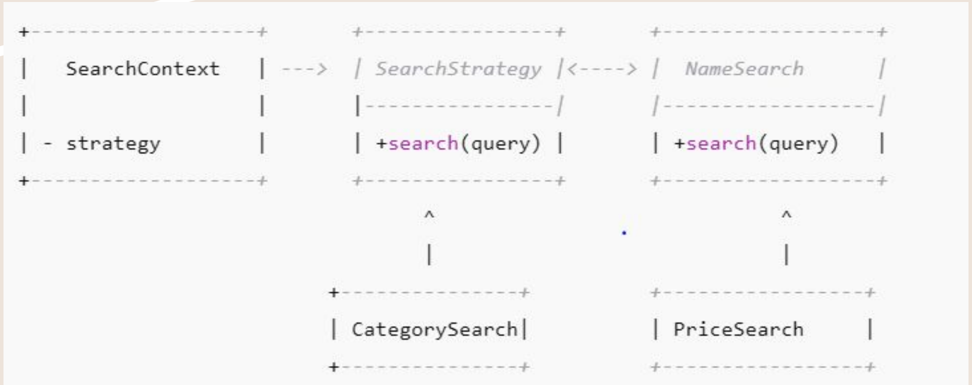
- Send cart to backend on checkout
- Persist cart to file or local DB
- Add session support for multiple users

Strategy Pattern – Concept Overview

What is the Strategy Pattern?
A behavioral design pattern that:

- Encapsulates interchangeable algorithms.
- Lets the algorithm vary independently from the clients that use it.

Why we used it:
To allow dynamic switching between **different product search methods**, e.g., by name or category.
Structure:



```
+-------------------+     +----------------+     +------------------+
|   SearchContext   | --> | SearchStrategy |<----> |  NameSearch     |
|                   |     |----------------|     |------------------|
| - strategy        |     | +search(query) |     | +search(query)   |
+-------------------+     +----------------+     +------------------+
                                  ^                       ^
                                  |                       |
                          +---------------+      +-----------------+
                          | CategorySearch|      | PriceSearch     |
                          +---------------+      +-----------------+
```

Strategy Pattern in Our Online Store

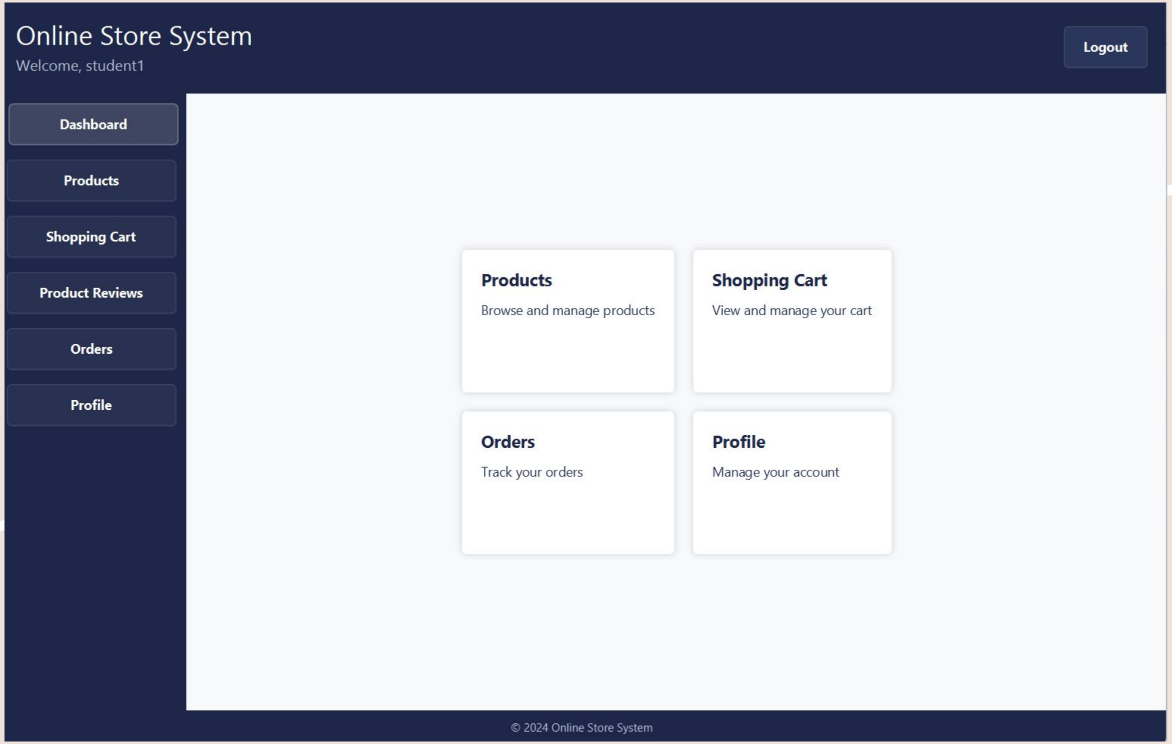
```
// Strategy Interface
public interface SearchStrategy {
    List<Product> search(String query);
}

// Concrete Strategy: Search by Name
public class NameSearch implements SearchStrategy {
    public List<Product> search(String query) {
        return productRepo.findByNameContainingIgnoreCase(query);
    }
}

// Context Class
public class ProductSearchContext {
    private SearchStrategy strategy;

    public void setStrategy(SearchStrategy strategy) {
        this.strategy = strategy;
    }

    public List<Product> executeSearch(String query) {
        return strategy.search(query);
    }
}
```

Dashboard

- Dynamic Load Fxml

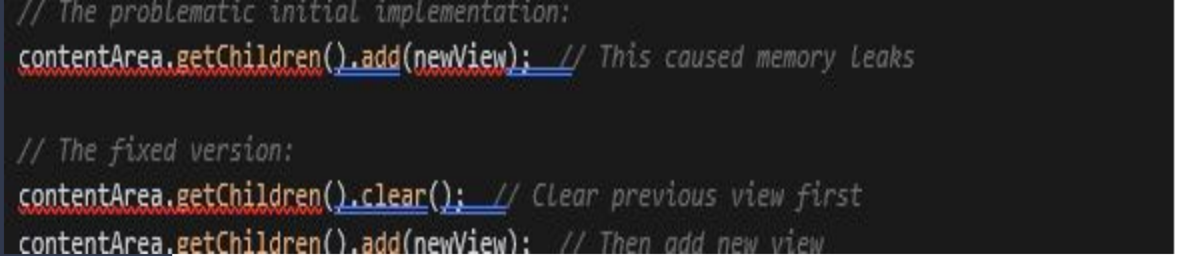
- Left Side Navigation
- Center (content)

- Top and bottom (Title and logout button)

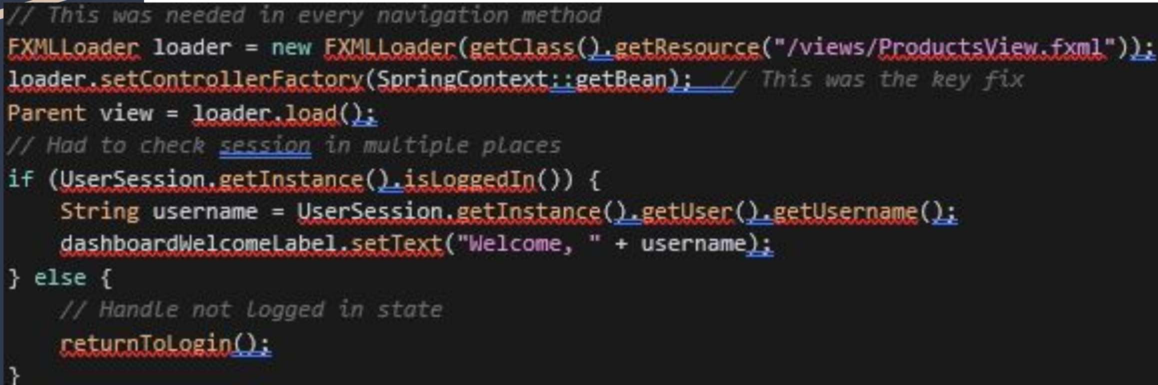


Problems

- Switching between views
- Controllers weren’t getting their @autowired services
- Reviews, Products and Cart services were null when navigating



```
// The problematic initial implementation:
contentArea.getChildren().add(newView);  // This caused memory leaks

// The fixed version:
contentArea.getChildren().clear();  // Clear previous view first
contentArea.getChildren().add(newView);  // Then add new view
```



```
// This was needed in every navigation method
FXMLLoader loader = new FXMLLoader(getClass().getResource("/views/ProductsView.fxml"));
loader.setControllerFactory(SpringContext::getBean);  // This was the key fix
Parent view = loader.load();
// Had to check session in multiple places
if (UserSession.getInstance().isLoggedIn()) {
    String username = UserSession.getInstance().getUser().getUsername();
    dashboardWelcomeLabel.setText("Welcome, " + username);
} else {
    // Handle not logged in state
    returnToLogin();
}
```

Future Enhancements

- AI-Based Product Recommendations
- Third-Party Payment Gateway
- Web-based Frontend with REST API

Summary

- Built with JavaFX (GUI) + Spring Boot (Backend)
- Follows MVC + 3-Tier Architecture
- Features: Product browsing, filtering, cart, payment, order history
- Uses Singleton-based in-memory cart management
- Connected to Azure SQL via Spring Data JPA
- Simulates real-world e-commerce workflows
- Clean separation between UI, logic, and data layers

Thank you :)