

**FACULTY OF COMPUTER SCIENCE AND ENGINEERING**

**FINAL PROJECT**

**SUBJECT: OBJECT ORIENTED PROGRAMMING(OOP**

**TOPIC:**

**Online Shop 2025**

**STUDENT:**

**1. TRAN CHI NGUYEN ( ID: 97482403501 )**

**2. NGUYEN HO HONG SON (ID: 97482403205)**

**LECTURER: DR. LE NGOC HIEU**

**ACADEMY YEAR: 2024-2025**

**Ho Chi Minh city, May 2025**

Table of Contents

[INTRODUCTION 4](#_Toc199287837)

[1. Reason for choosing the topic 4](#_Toc199287838)

[2. Research purpose and tasks 4](#_Toc199287839)

[3. Object and scope of research 5](#_Toc199287840)

[4. Research methods 5](#_Toc199287841)

[CONTENT SECTION 6](#_Toc199287842)

[Part 1 : Interface and Features 6](#_Toc199287843)

[1.1. Application Overview 6](#_Toc199287844)

[1.2. Application interface 6](#_Toc199287845)

[1.2.1 Login page 6](#_Toc199287846)

[1.2.2 Dashboard 7](#_Toc199287847)

[1.2.3 Cart and order history 7](#_Toc199287848)

[1.2.4 Product Category Distribution 9](#_Toc199287849)

[**1.3** **Detailed functions inside the system**: 9](#_Toc199287850)

[Part 2: Proposed Solution and Design Analysis 10](#_Toc199287851)

[2.1. Proposed solution 10](#_Toc199287852)

[2.2. MVC model 10](#_Toc199287853)

[2.3. User Interface (UI) 11](#_Toc199287854)

[2.4. Database Design 11](#_Toc199287855)

[2.4.1 User table 11](#_Toc199287856)

[2.4.2 Products table 11](#_Toc199287857)

[2.4.3 Orders table 12](#_Toc199287858)

[**3. System operation mechanism** 12](#_Toc199287859)

[3.1 Main class and database connection 13](#_Toc199287860)

[**3.2 Communication between Controller and View** 13](#_Toc199287861)

[**3.2.1 LoginController.java** 13](#_Toc199287862)

[**3.2.2** **DashboardController.java** 14](#_Toc199287863)

[**3.2.3 MyProfileController** 17](#_Toc199287864)

[**3.2.4 ShoppingCartManager** 18](#_Toc199287865)

[**3.2.5 OrderHistoryController**: 19](#_Toc199287866)

[**3.2.6 ProductDetailController**: 20](#_Toc199287867)

[**3.2.7 ProductListController:** 21](#_Toc199287868)

[**3.3 Model and data processing layer** 22](#_Toc199287869)

[**3.3.1 Product:** **Stores product information (ID, name, price, category, etc.)** 22](#_Toc199287870)

[**3.3.2 User:** 24](#_Toc199287871)

[**3.3.3 CartItem** Represents a product in the shopping cart (product + quantity) 26](#_Toc199287872)

[**3.3.4 OrderSummary**: 27](#_Toc199287873)

[**3.4 Support utilities** 28](#_Toc199287874)

[**3.4.1 ChartGenerator**: 28](#_Toc199287875)

[**3.4.2 UserSession**: 29](#_Toc199287876)

[4. Product Reviews 30](#_Toc199287877)

[4.1 Strengths 30](#_Toc199287878)

[4.2 Weaknesses 32](#_Toc199287886)

[SUMMARY 33](#_Toc199287893)

**Task Assignment Board**

|  |  |  |
| --- | --- | --- |
| **Student**  **Task** | **Tran Chi Nguyen** | **Nguyen Ho Hong Son** |
| **Data processing** | ✅ |  |
| **Code for cart, product, order history in pack Controller** |  | ✅ |
| **Code the dashboard, login, my profile section in the Controller pack, Main** |  | ✅ |
| **Code the file that connects to MySQL and code the tables in MySQL.** |  | ✅ |
| **Code JFreechart** |  | ✅ |
| **Code Model pack** | ✅ |  |
| **Make PowerPoint file, README, Record demo video** |  | ✅ |
| **Make report file** | ✅ |  |
| **Interface design, Git Hub, Git Responstory** |  | ✅ |

## INTRODUCTION

## 1. Reason for choosing the topic

In recent years, the trend of online shopping has grown strongly, especially after the pandemic, most people are used to ordering online instead of going directly to the store. Developing an online sales system is not only a trend but also an essential skill for modern programmers. The topic "Online Shopping 2025" was chosen with the goal of simulating the operating process of a real e-commerce platform, while comprehensively applying knowledge of object-oriented programming, databases, and software design thinking. Using JavaFX helps me practice the skills of building modern user interfaces, while MySQL supports systematic data storage. In addition, the integration of JFreeChart allows statistics to be displayed visually, helping administrators easily grasp the operating situation..

## 2. Research purpose and tasks

The topic “Online Shopping 2025” was chosen with the goal of building an application that simulates online shopping activities. The implementation of the topic not only aims to complete a software product with a complete interface and functions, but also focuses on properly implementing the technical requirements set out in the topic. Specifically, the topic aims at the following goals:

• Build a Java application with a friendly interface, using JavaFX to design a modern, easy-to-use UI for users who are administrators or system staff.

Fully apply the principles of object-oriented programming (OOP), including: using reasonable classes and objects, organizing inheritance - polymorphism - encapsulation - abstraction in a standard programming model.

• Organize the program according to the MVC design model, ensuring separation between data processing, interface and control logic, making the source code clear, easy to expand and maintain.

• Connect to MySQL database via JDBC, perform full CRUD operations (Create, Read, Update, Delete) with at least 5 data tables, including a user table storing encrypted passwords.

• Integrate Java Collections data structures (ArrayList, HashMap, TreeMap, etc.) for flexible data processing, serving search, filter, login session storage and classification statistics functions.

• Build a login/logout authentication system, with a secure session storage mechanism and a basic user personal information management interface.

• Create statistical charts using the JFreeChart library, visually displaying information such as revenue, number of orders, best-selling products, etc. to help administrators make decisions.

## 3. Object and scope of research

• Target: System users including admin and customers.

• Scope: The system includes functions such as: login, product management, ordering, order management, revenue statistics, not including actual payment or delivery functions.

## 4. Research methods

• Analyze system requirements and design functions.

• Apply MVC model in software development.

• Implement interface using JavaFX.

• Use JDBC to connect and process data on MySQL.

• Use JFreeChart to display statistical charts..

5. **Topic structure**

The topic is divided into 3 parts:

* Introduction
* Content section
  + Part 1: Interface and Features
  + Part 2: Proposed Solution and Design Analysis
* Summary

# CONTENT SECTION

# Part 1 : Interface and Features

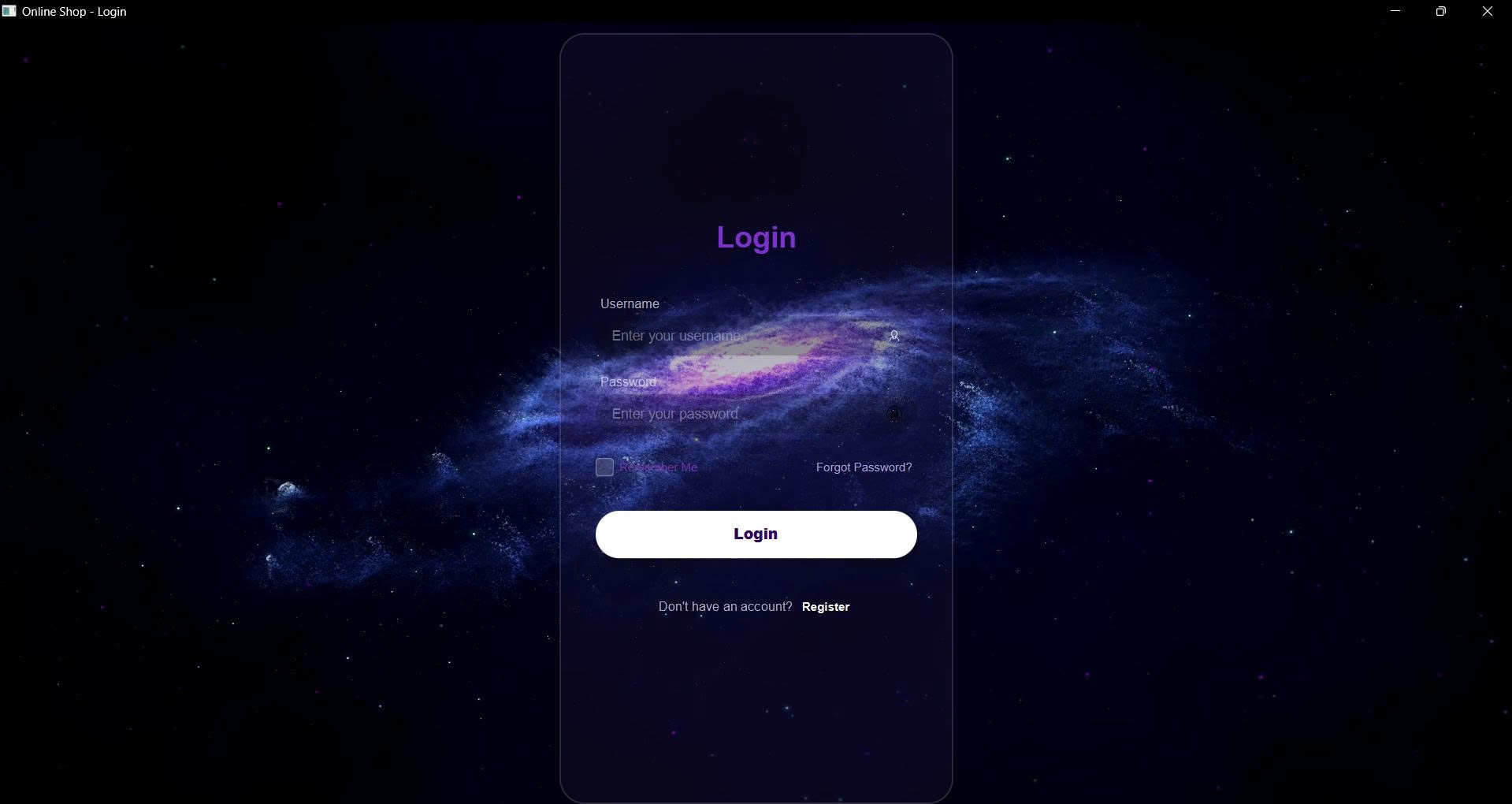
## 1.1. Application Overview

The "Online Shopping 2025" application is a basic online sales system built with JavaFX combined with MySQL and the JFreeChart library. The system includes two main user roles: Admin and Customer. Admin can manage products, orders, customers and sales statistics. Customers can log in, view products, place orders and view order history.

## 1.2. Application interface

## 1.2.1 Login page

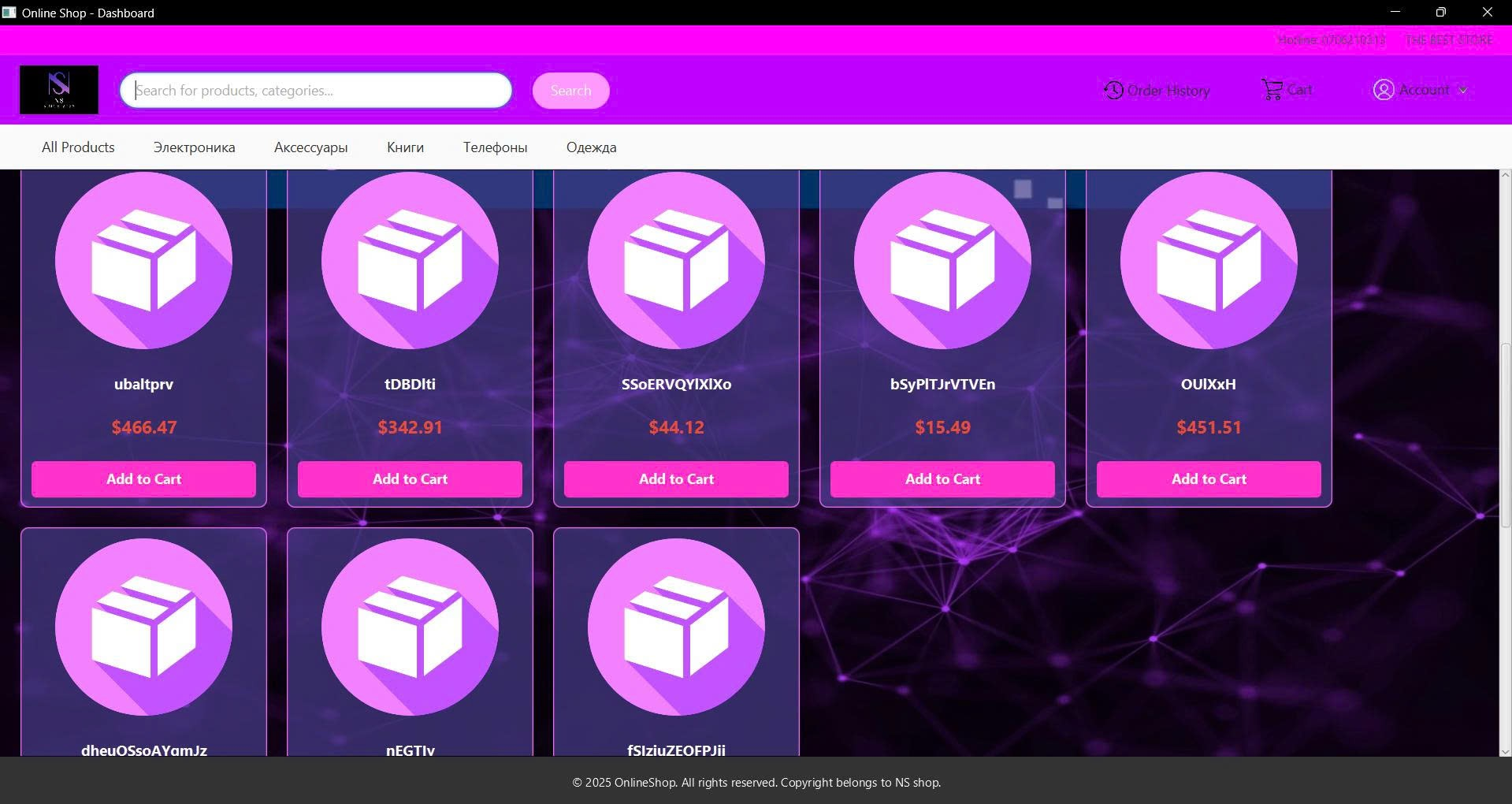
The login interface is the first screen when the user launches the application. With a fantasy space background design, the interface creates a modern and outstanding feeling. Users need to enter Username and Password to access the system. There are also additional functions such as "Remember login", "Forgot password" and a link to the registration page.



*Hình 1.1: Giao diện đăng nhập*

## 1.2.2 Dashboard

* The Dashboard interface is designed vividly with modern purple-pink tones. The main screen includes a search bar, product categories, a product list displayed in card form, and navigation buttons such as "Order History", "Cart", and "Account".
* Search bar: Allows quick search for products by name or category.
* Product categories: Displayed in tab form, supporting many languages ​​(including Russian).
* Product list: Each product is displayed in a purple circular card with the product name, price, and "Add to Cart" button.
* Top navigation bar: Displays the store logo, main function buttons, and system clock.

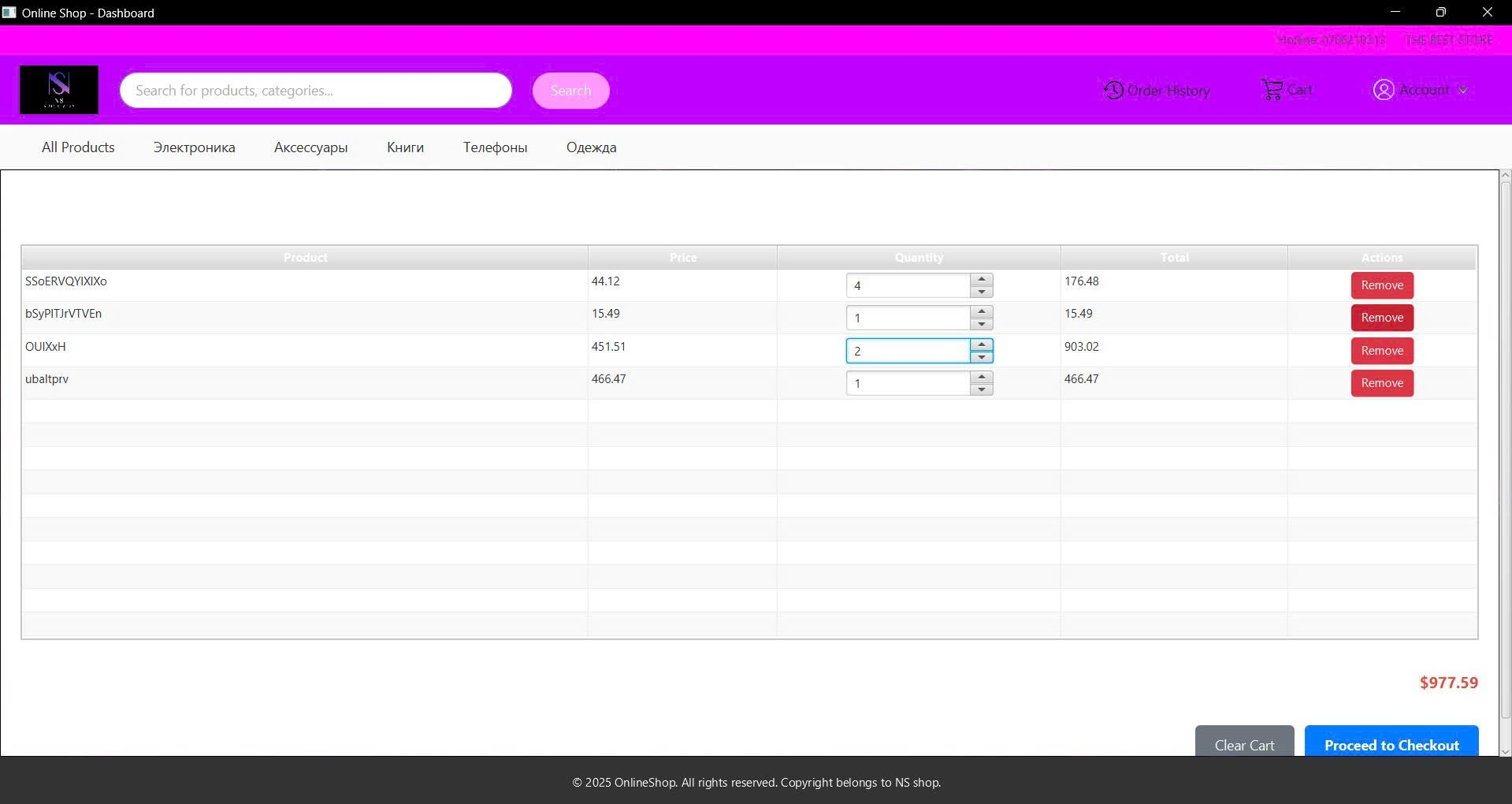


Picture 1.2: Dashboard

## 1.2.3 Cart and order history

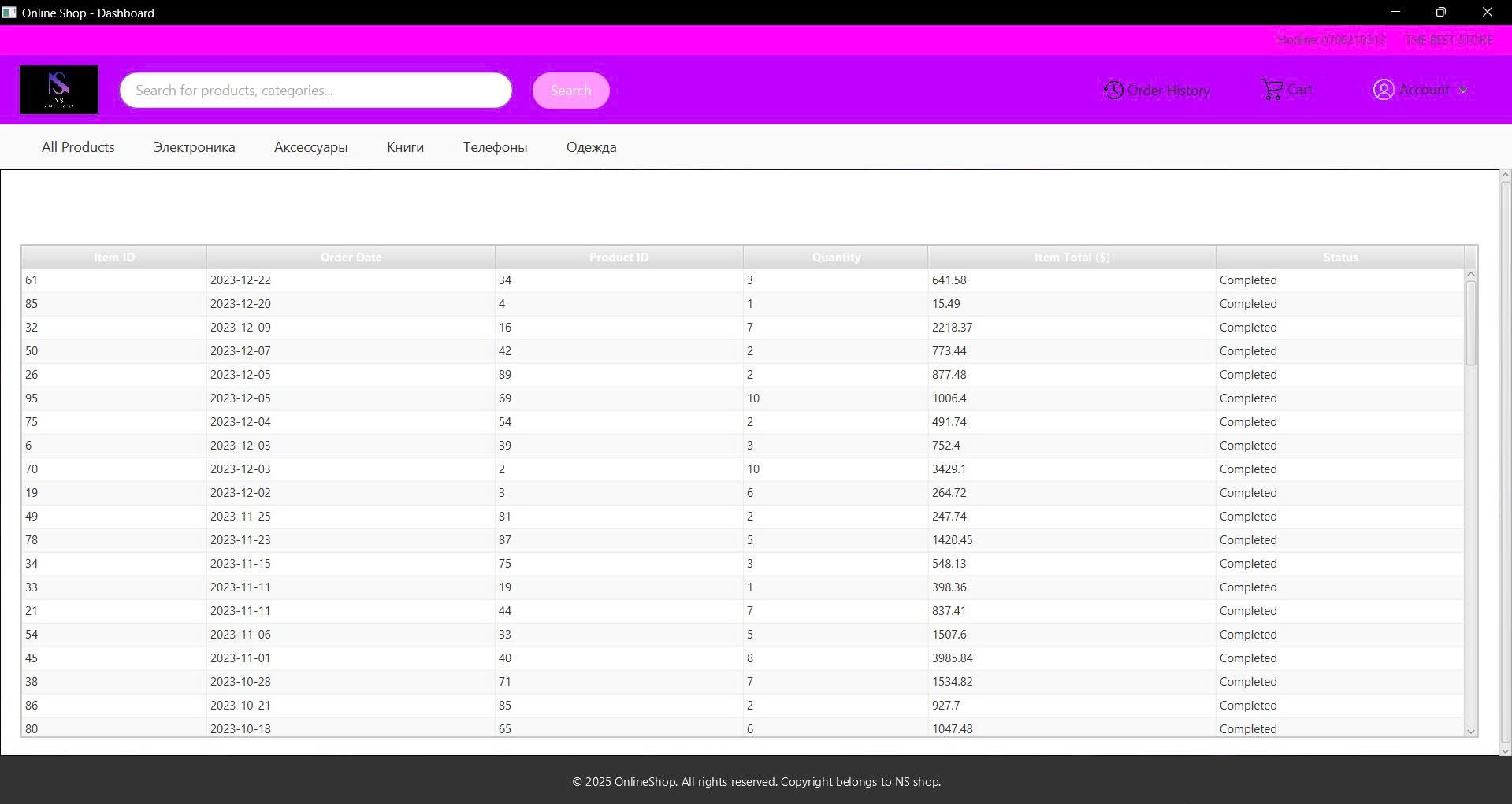
Shopping cart and order history functionality are an important part of the user experience.

* Shopping cart: Allows customers to add products to their cart, adjust quantities, or remove products before proceeding to checkout. Information such as product name, price, quantity, and total are updated automatically.



Hình 1.3: cart

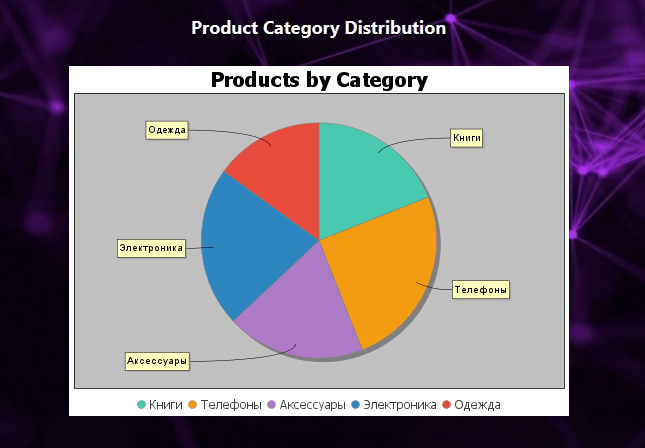
* **Order history:** Displays a list of orders placed including: Order code, order date, product code, quantity, total amount and order status. The data table interface helps users easily look up and track the progress of orders.



Hình 1.4: Oder History

## 1.2.4 Product Category Distribution

This function displays a statistical chart that visually represents the distribution of products. The chart is created using the JFreeChart library integrated with JavaFX to show the distribution of product types in the sales category.



**1.3** **Detailed functions inside the system**:

* Login/logout function: Authenticate users, save login information using UserSession.
* Product management: Use products table, add/edit/delete functions are performed via JavaFX interface.
* Order management: When users place an order, the system saves information in orders and order\_details tables.
* User management: User data is saved in users table with encrypted password.
* Search and filter: Use Java Collections functions to filter the list by name, product type, ...
* Statistics: Use ChartGenerator.java class to draw bar charts, pie charts from data in the database.

# Part 2: Proposed Solution and Design Analysis

## 2.1. Proposed solution

* Based on the requirements and objectives of the project, the Online Shopping 2025 system is built as a desktop application applying the MVC architectural model to clearly separate the interface, logic processing and data. The user interface is built using JavaFX, the database management system uses MySQL, connected via JDBC. The system has integrated JFreeChart to display visual statistical charts such as revenue, monthly orders, best-selling products, etc. to support administrators in monitoring the system's performance. In addition, the project is developed and managed using the Eclipse IDE tool combined with Maven to manage the library. The system is divided into clear modules:
* Controller: handles logic and control flow (LoginController, DashboardController, ShoppingCartController,...)
* Model: defines data entities such as Product, User, OrderSummary, CartItem...
* Util: supporting functions (statistical charts, session storage,...)

## 2.2. MVC model

* Model: Contains classes that define the system's main data such as:
* Product.java, CartItem.java, OrderSummary.java, User.java
* Designed according to the object-oriented principle, with full constructors, getters/setters.
* View: The user interface is built with JavaFX, including forms such as:
* Login interface, Dashboard main interface, account management, product list, order details, shopping cart.
* Views are organized in the view folder and linked to the Controller via FXML files.
* Controller: Responsible for navigating and handling logic between View and Model:
* Includes classes such as LoginController.java, DashboardController.java, ShoppingCartController.java, ProductListController.java, etc.
* Controller communicates with DAO or utility classes to perform operations such as login, add/edit products, and process shopping carts.
* Connect: The folder contains the DBConnection.java class, responsible for connecting to the MySQL database.
* Util: Supporting classes such as ChartGenerator.java (drawing charts), UserSession.java (saving session information).
* Model: Includes classes representing system data such as Product, User, Order, Category... Designed according to OOP standards (with getter/setter, constructor, etc.)
* View: JavaFX interfaces (for users and admins), including login form, home page, interface for viewing/editing/deleting/adding products.
* Controller: Processes business logic, interacts with DAO and updates View.

## 2.3. User Interface (UI)

* Friendly interface, clear layout: header, menu bar, content pane.
* Functional forms: login, admin dashboard, shopping cart, product list, search form.

## 2.4. Database Design

The system database is built using MySQL, including 3 main tables: users, products, and orders. The tables are designed simply but meet the business requirements of the online shopping system.

## 2.4.1 User table

Store user information, including:

• user\_id: primary key, auto-incrementing

• username: login name (unique)

• password\_hash: encrypted password

• email, full\_name, address, phone\_number: contact information

• registration\_date: account registration date (default is current date)

## 2.4.2 Products table

Save product information:

• product\_id: primary key

• product\_name: product name

• category: product type (Russian for data diversity)

• price: unit priceBảng này là trung tâm của chức năng hiển thị và quản lý sản phẩm.

## 2.4.3 Orders table

Save order information:

• order\_id: primary key

• order\_date: order date

• product\_id: link to products table

• quantity: quantity ordered

• total\_price: total price of order

The tables have a clear relationship:

• orders.product\_id → products.product\_id

• It is possible to extend user\_id to the orders table to track orders by user.

Sample data has been added to all 3 tables for testing and charting statistics in the system.

**3. System operation mechanism**

This section details how the components in the system work and coordinate with each other to ensure a smooth online sales process, from user login to order placement and statistics tracking.

## Main class and database connection

In the Main.java class, the system is started through the start(Stage primaryStage) function, which is the entry point of the JavaFX application.

“FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/Login.fxml"));

Scene scene = new Scene(loader.load());

scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());”

• This line loads the Login.fxml interface file and applies the application.css stylesheet, which helps shape the layout, colors, and effects of the login interface.

• The interface is named Online Shop - Login and is displayed immediately when the application is run.

“launch(args);”

• A call to Application.launch() starts the entire JavaFX interface flow.

Main.java acts as the single entry point to the system, initializing the interface and calling the corresponding controllers to start the application logic processing cycle.

• Main.java: The main launch point of the application. Here, the system initializes the first JavaFX interface, the login page.

• DBConnection.java: Contains the logic to connect to MySQL via JDBC. This class ensures efficient connection reuse and management, allowing Controllers to access stable data.

**3.2 Communication between Controller and View**

**3.2.1 LoginController.java**

LoginController is the logic class for the login interface. When the user launches the application, JavaFX will load Login.fxml, and LoginController is automatically linked to control the components on the interface. Some main functions:

• Interface effects: When the interface is launched, the initialize() function is called to play the FadeTransition and TranslateTransition effects to make the login interface smoother. In addition, the video background is also loaded using MediaPlayer with the .mp4 video format.

• User login: The handleLogin() function performs processing when the user clicks the login button. If the account matches the condition, the system:

Creates a User object simulating the user who is logging in.

Assigns it to UserSession to save the session.

Loads the Dashboard screen using FXMLLoader and displays it.

• Error notification: If the account is invalid or the FXML file is corrupted, the system will display an Alert to notify the user.

• Unfinished features: Buttons like “Forgot Password”, “Register” or “Google Login” are all defined events, but currently only display a “not supported yet” message.

• Video size handling: MediaView is dynamically sized bound to the rootPane to ensure full-screen login display.

The LoginController class not only handles login logic but also enhances the user experience with animations and modern video backgrounds.

* + 1. **DashboardController.java**

Main navigation in the application, managing the main admin interface (including menus, charts, administrative functions...)

• DashboardController controls the main interface of the application, integrating components such as product search, category display, shopping cart, order history and user profile management. This class uses JavaFX to build dynamic interfaces, combined with the database (via the DBConnection class) to load product data and display them as lists or product cards. In addition, it integrates multimedia features such as playing promotional videos and animated wallpapers, along with product category distribution charts.

Main functions

1. Initialize interface:

 The initialize method is called when the interface is loaded, responsible for initializing components such as video ads (initializeDashboardBannerVideo), animated backgrounds (initializeContentBackgroundVideo), and product category pie charts (ChartGenerator.createCategoryPieChart).

 Product data is loaded from the database via the loadProductsFromDatabase method, saved to the allProductsFromDB list for use in other functions.

 The main interface is displayed via showMainContentLayout, ensuring components such as video banners, featured product lists, and scroll bars work correctly.

2. Product Management:

 Load products from database (loadProductsFromDatabase): Use JDBC to query products table, get information such as product\_id, product\_name, category, price and save to allProductsFromDB list. Database error handling is done carefully, displaying error messages in the interface if necessary.

 Display products (displayFeaturedProducts, displayProductsByCategory, displaySearchedProducts): These methods filter and display products in FlowPane as product cards, supporting pagination (displayProductsPage) with the number of products per page is 8 (ITEMS\_PER\_PAGE). Product cards include image, name, price and "Add to Cart" button.

 Pagination (handlePreviousPage, handleNextPage): Provides navigation through product pages, updates pagination control interface (updatePaginationControlsUI) to reflect current page state.

3. Search and filter products:

 The handleSearch method handles the search event, filters products based on name or category, uses the Stream API to filter the allProductsFromDB list. The results are displayed in the featuredProductsPane, with appropriate messages if no products are found.

4. Media management:

 Promotional videos and backgrounds: The initializeDashboardBannerVideo and initializeContentBackgroundVideo methods load and play videos from internal resources (/images/dashboard\_banner\_video.mp4, /images/another\_background\_video.mp4). Videos are played in an infinite loop (INDEFINITE) and handle errors such as the file not existing or the MediaPlayer crashing.

 MediaPlayer lifecycle management: The stopAllMediaPlayers method ensures that MediaPlayer objects stop and release resources when the user logs out or switches interfaces, avoiding memory leaks.

5. Interacting with shopping cart and user profile:

 Shopping cart (handleViewCart, createProductCard): Integrates with ShoppingCartManager to add products to cart, display confirmation message via Alert.

 Order history (handleOrderHistory) and user profile (handleMyAccount): Loads corresponding interfaces (OrderHistory.fxml, MyProfile.fxml) into mainScrollPane using FXMLLoader.

 Logout (handleLogout): Clears user session (UserSession.clearSession), stops MediaPlayers and switches to login interface (Login.fxml).

6. Handling dynamic interfaces:

 The loadViewIntoScrollPane method loads other FXML interfaces into mainScrollPane, allowing flexible switching between views (products, shopping cart, profile, etc.).

 Handling of FXML loading errors is done by displaying error messages in the interface. ProductListController.java, ProductDetailController.java: Handles the display of product list and details.

**3.2.3 MyProfileController**

Manage current user account information. The interface includes fields such as login name, full name, email, address and phone number. Some main functions:

1. Load user information: The loadUserProfile() method queries the users table by user\_id, then displays data into the interface fields via the populateFields() method.

2. Edit and save changes: When the user edits fields such as full name, email, address or phone number and clicks “Save Changes”, the system will update the database via the UPDATE statement. If successful, a confirmation message will be displayed and the new data will be reloaded.

3. Change password: The “Change Password” button is declared but the function has not been implemented yet, currently only a notification is displayed.

4. Display errors and notifications: Use JavaFX Alert to respond to users when errors occur or operations are successful.

5. Prevent username modification: The usernameField field is set to be uneditable, ensuring that the identification data is not changed.

The MyProfileController class plays an important role in allowing users to manage personal information in an intuitive and secure way.

• ShoppingCartController.java: Manages all display logic and handles interactions with the shopping cart. This class is mainly responsible for the shopping cart section in the application. Some outstanding functions include:

Display a list of products in the cart: Use TableView<CartItem> to display products, quantity, unit price and total price of each item.

Dynamic quantity update: The quantity column uses Spinner<Integer> to allow users to change directly on the table. When changing, the system automatically updates the total amount by calling cartManager.updateQuantity().

Delete products: The actionsColumn column displays the "Remove" button to delete each product in the cart.

Calculate total amount (subtotal): After each change, the updateSubtotal() function is called to update the total amount of the order and activate/hide the payment button.

Dynamic UI effects: The interface displays a message if the cart is empty, automatically updating the table placeholder.

Handle Clear and Checkout buttons:

* handleClearCart(): Delete all products in the cart.
* handleCheckout(): Check if the cart is empty, if not, notify (function not fully implemented).

**3.2.4 ShoppingCartManager**

The **ShoppingCartManager** class is a central singleton responsible for managing all the shopping cart logic. It provides a consistent API for controllers to interact with the cart using an ObservableList. Key functionalities include:

* **Adding products to the cart:**
  + The addProduct(Product product, int quantity) method checks if the product already exists in the cart. If so, it increases the quantity; otherwise, it creates a new CartItem and adds it to the list.
* **Removing products from the cart:**
  + The removeProduct(CartItem item) method removes a product from the cart list.
* **Updating product quantity:**
  + The updateQuantity(CartItem item, int newQuantity) method updates the quantity. If the new quantity is less than or equal to 0, the item is removed from the cart.
* **Calculating total cart price:**
  + The getTotalCartPrice() method loops through the list and sums up item.getTotalPrice() for all items.
* **Clearing the entire cart:**
  + The clearCart() method empties the product list.
* **Counting items in the cart:**
  + The getItemCount() method returns the number of distinct (unique) items. To get the total number of units, you can loop through each item.getQuantity().

The ShoppingCartManager class acts as a service layer for controllers such as DashboardController and ShoppingCartController, centralizing the cart logic into one place, making it easier to maintain and extend in the future.

**3.2.5 OrderHistoryController**:

**Displays and manages the user’s order history.** Some of the main functionalities include:

* **Setting up the data table:**  
  Uses TableView<OrderSummary> with columns such as Order ID, Order Date, Product ID, Quantity, Total Price, and Status. These columns are bound to properties from the OrderSummary class.
* **Loading data from the database:**  
  The loadOrderHistory() method queries the orders table, sorts the results by order date, and loads the data into the orderList. The data is then mapped into the table using an ObservableList.
* **Handling empty data display:**  
  If the order list is empty, the table will show the placeholder text: *"You have no past orders."*
* **Database connection via DBConnection class:**  
  Ensures reusable and reliable access to the database whenever data needs to be retrieved.
* **Default user\_id = 1:**  
  In the current version, the system simulates the scenario with a hardcoded user ID of 1 to simplify the display of historical order data.

**3.2.6 ProductDetailController**:

The **ProductDetailController** handles the logic for the product detail screen. This is where users can view full information about a product before deciding to add it to their cart.

**Main functionalities:**

* **Displaying product details:**
  + The setProduct(Product product) method is used to pass the product data into the UI.
  + Information such as product name, price, category, and image is displayed in detail by updating the content of Label and ImageView components.
* **Handling product images:**
  + The product image is retrieved from the imagePath field in the Product object.
  + If the image cannot be found or the path is invalid, the system automatically replaces it with a default image at /images/product\_placeholder.png, preventing UI errors.
* **"Add to Cart" button:**
  + When users click the "Add to Cart" button (handleAddToCart), the system currently logs the action to the console via System.out.println(...) for confirmation.
  + This functionality can be extended to actually add the product to the cart using ShoppingCartManager.addProduct(...).

The ProductDetailController adds professionalism and depth to the user experience by providing full product details and acting as a bridge between the product list and the shopping cart.

**3.2.7 ProductListController:**

The **ProductListController** class is responsible for managing the user interface that displays the product list. It supports actions such as product searching, category filtering, or displaying all available products. This controller works with the main user interface where users browse product categories.

**Main functionalities:**

* **Controller initialization:**
  + In the initialize() method, the class logs a message to confirm successful initialization. This is also a good place to add extended logic in the future.
* **Searching for products (searchProducts()):**
  + This method takes a search keyword as a parameter.
  + It updates the categoryLabel to display: *"Search Results for: <keyword>"* to reflect the current user query.
  + Useful when integrated with the system's search function triggered from the DashboardController.
* **Filtering products by category (filterByCategory()):**
  + When the user selects a specific category, the method updates the label to: *"Category: <category name>"*.
  + Helps users clearly identify the current category being viewed.
* **Displaying all products (showAllProducts()):**
  + Used when users want to view all available products without any filters.
  + The label is updated to: *"All Products"*.

**Summary:**  
ProductListController.java enhances the user experience when browsing product lists. It acts as the bridge between the UI and the logic for product filtering and searching. Although currently basic, it’s highly extendable—future improvements could include sorting by price, filtering by rating, or paginating search results.

**3.3 Model and data processing layer**

The classes in the model directory represent the system's data and follow OOP principles:

**3.3.1 Product:**  
**Stores product information (ID, name, price, category, etc.)**

The Product class is a simple Java class following the Plain Old Java Object (POJO) model. It is designed to store and manage information about a product, including its ID, name, category, price, and image path. This class is used throughout the application to display products in the UI, manage shopping cart operations, and interact with the database.

Structure and Functionality

**Attributes:**

The class defines five private attributes:

* productId (int): A unique identifier for the product.
* productName (String): The name of the product.
* category (String): The product category (e.g., "Electronics", "Clothing").
* price (double): The price of the product.
* imagePath (String): The path to the product image file, used for visual display in the UI.

These attributes are declared as private to enforce encapsulation, ensuring data can only be accessed or modified through public methods.

**Constructor:**

The class provides a single constructor:

“public Product(int productId, String productName, String category, double price, String imagePath)”

This constructor accepts all necessary fields to fully initialize a Product object. The imagePath field allows specifying either a custom image or a default placeholder (e.g., "/images/product\_placeholder.png").

**Getter Methods:**

The class provides getter methods for all attributes:

* getProductId(), getProductName(), getCategory(), getPrice(), getImagePath()

These methods allow safe access to the attribute values. They are used in other classes (like DashboardController) to display product information or perform actions like search and filtering.

**Setter Methods:**

Setter methods are provided for most attributes, except for productId:

* setProductName(), setCategory(), setPrice(), setImagePath()

These setters allow product information to be updated after initialization, which is useful in cases such as price or image updates. The absence of a setter for productId ensures that the product ID remains immutable, maintaining its role as a unique identifier.

**toString Method:**

The toString method is overridden to provide a readable string representation of the Product object, including all attributes:

“@Override

public String toString() {

return "Product{" +

"productId=" + productId +

", productName='" + productName + '\'' +

", category='" + category + '\'' +

", price=" + price +

", imagePath='" + imagePath + '\'' +

'}';

}”

This method is useful for debugging or logging, offering a clear and complete view of the product’s data in string format.

**3.3.2 User:**

**Stores registered user information**

The User class in the model package is a simple Java class (POJO) used to represent user information in the online shopping application. Its core functionality includes:

Attributes:

* userId (int): A unique identifier for the user.
* username (String): The username used to log in.
* email (String): Email address.
* fullName (String): Full name of the user.
* address (String): Physical address.
* phoneNumber (String): Phone number.
* registrationDate (LocalDateTime): The time the account was registered.

All attributes are declared private, ensuring encapsulation and data integrity.

Constructors:

* **Full constructor:** Accepts all attributes (userId, username, email, fullName, address, phoneNumber, registrationDate) to create a complete User object.
* **Simplified constructor:** Accepts attributes excluding registrationDate, useful in contexts where the registration time is not needed.
* **Empty constructor:** Takes no parameters, used for temporary object creation (especially helpful for JavaFX FXML or data binding).

Getter Methods:

Provides access to attribute values via:

* getUserId(), getUsername(), getEmail(), getFullName(), getAddress(), getPhoneNumber(), getRegistrationDate()

Setter Methods:

Allows updates to attribute values via:

* setUserId(), setUsername(), setEmail(), setFullName(), setAddress(), setPhoneNumber(), setRegistrationDate()

How It Works

**Object Initialization:**

When the application needs to represent a user (e.g., after login or when retrieving data from the database), a User object is created using one of the constructors. For example:

“User user = new User(1, "john\_doe", "john@example.com", "John Doe", "123 Main St", "555-1234", LocalDateTime.now());”

An empty constructor (new User()) can also be used if the object needs to be instantiated first and populated later via setters.

**Accessing and Updating Data:**

* Getter methods are used to retrieve user data, such as displaying the username on the UI (user.getUsername()) or validating email in authentication logic.
* Setter methods are used to update user information, for example:

“user.setAddress("456 New St");

user.setPhoneNumber("555-5678");”

User data can then be displayed in the interface (e.g., MyProfile.fxml) or saved back to the database.

**Usage in the System:**

The User class is used in various parts of the system, such as DashboardController, to manage the user session (via UserSession.getInstance().getLoggedInUser()).

User information is passed into interfaces like the personal profile or order history views, and also used to verify login status. This class provides a clean structure for storing and managing user information, acting as a data model between the database and the UI, and is easily integrable with JavaFX and other application components.

**3.3.3 CartItem**  
**Represents a product in the shopping cart (product + quantity)**

The CartItem class, located in the model package, is a core component of the cart feature in the application. Each CartItem object represents a specific product along with the quantity added by the user.

Structure and Key Features

**Attributes:**

* product (SimpleObjectProperty<Product>): The associated product object.
* quantity (SimpleIntegerProperty): The number of items added to the cart.
* totalPrice (SimpleDoubleProperty): The total value (price × quantity), automatically updated whenever the quantity changes.

**Object Initialization:**

* The constructor accepts a Product object and an int quantity, initializing the totalPrice accordingly.
* A **listener mechanism** is implemented so that whenever the quantity changes, the totalPrice updates automatically—no need for manual recalculation.

**Getter and Setter Methods:**

* The class provides full getter and setter methods, as well as JavaFX property getters:  
  productProperty(), quantityProperty(), and totalPriceProperty()—perfect for data binding with the UI in JavaFX.
* Updating either the product or quantity will automatically adjust the totalPrice.

**Utility Methods:**

* getProductName() and getProductPrice() provide quick access to product info, often used in the cart table view.
* getTotalPrice() returns the total value for this cart item, usually used in the cart’s total payment calculation.

**Role in the System:**

CartItem serves as a bridge between the actual product (Product) and the shopping cart logic (ShoppingCartManager). Thanks to JavaFX Properties and binding mechanisms, the cart UI responds instantly to user interactions like changing product quantity. Total price updates happen automatically, optimizing user experience and simplifying controller logic.

**3.3.4 OrderSummary**:

**Aggregates order data for display or database storage**

The OrderSummary class, located in the model package, represents a row of data in the orders table. It is commonly used to display the user’s order history in a table format (TableView).

Structure and Functionality

**Attributes:**

* orderId: The order ID or order line ID (primary key).
* orderDate: The date the order was placed, stored as a String for easy display on the interface.
* productId: The product ID within the order, used to reference the products table.
* quantity: Number of units ordered.
* totalPrice: The total value for this order line (quantity × unit price).
* status: The status of the order (e.g., "Completed", "Pending").

**Object Initialization:**

* The constructor takes all 6 fields, making it easy to create an object from database results (ResultSet).
* All attributes are wrapped in JavaFX Property types, allowing seamless interaction with UI components (binding with TableView, Label, etc.).

**Getter and Property Getter Methods:**

* Includes standard getters:  
  getOrderId(), getOrderDate(), getProductId(), getQuantity(), getTotalPrice(), getStatus().
* Property getters like orderIdProperty(), orderDateProperty() are used for JavaFX UI binding.

**Role in the System:**

The OrderSummary class acts as a perfect intermediate structure between queried database data and the JavaFX user interface. Instead of displaying raw ResultSet data, the controller creates OrderSummary objects and inserts them into an ObservableList, which is then rendered automatically by the TableView.

In addition, this class helps manage extra display fields such as status, which may not be stored in the database but can be temporarily assigned in the controller to provide a clearer view to the user.

The class is fully equipped with a constructor and getter/setter methods, enabling easy data manipulation across different controllers.

**3.4 Support utilities**

**3.4.1 ChartGenerator**:

**Uses the JFreeChart library to create bar and pie charts for visualizing revenue, orders, and best-selling products.**

The ChartGenerator class, located in the util package, is responsible for generating pie charts from product data, helping visualize the distribution of product categories in the system.

Key Functionalities:

* **Generate product category chart (generateCategoryChart()):**
  + **Parameters:** A list of products (List<Product>) and the chart dimensions (width, height).
  + **Implementation:** Uses Java’s Stream API to group products by category and count each type using Collectors.groupingBy(...).
  + The grouped data is loaded into a DefaultPieDataset, which serves as the data source for the chart.
* **Create chart using JFreeChart:**
  + Uses ChartFactory.createPieChart() to generate a pie chart titled "Products by Category".
  + The PiePlot is customized as follows:
    - Assigns specific colors for each category (using RGB values).
    - Sets custom fonts, label spacing, and displays a message "No data available" when no data exists.
* **Convert to JavaFX image (toFXImage()):**
  + After creating a BufferedImage, the chart is converted into a JavaFX-compatible Image using SwingFXUtils.toFXImage(...), allowing it to be displayed in the interface.

Highlights:

* Utilizes **JFreeChart** to create professional-quality charts in JavaFX interfaces.
* Smoothly integrates with JavaFX via SwingFXUtils.
* Helps administrators quickly grasp the distribution of products by category.

Real-World Application:

Used in DashboardController.java to display product category distribution charts on the main admin dashboard.

**3.4.2 UserSession**:

**Stores the currently logged-in user's state and passes user information between screens without needing to pass parameters repeatedly.**

The UserSession class, located in the util package, acts as a singleton that stores the logged-in user's information throughout the session. Instead of passing a User object manually across multiple controllers, this class allows global access to user data from anywhere in the application.

Key Characteristics:

Singleton Pattern:

* UserSession follows the Singleton design pattern to ensure that only one instance exists and is used system-wide.
* The getInstance() method is used to access the single instance of the class.

Main Attribute:

* User loggedInUser: Stores the currently logged-in User object.

Core Methods:

* setLoggedInUser(User user): Sets the current user after a successful login.
* getLoggedInUser(): Returns the currently logged-in User object.
* getLoggedInUserId(): Returns the user\_id if a user is logged in.
* getLoggedInUsername(): Returns the username if available.
* isLoggedIn(): Checks whether a user is currently logged in.
* clearSession(): Clears the current user from the session (used during logout).

Application in the System:

* Used in classes such as DashboardController, OrderHistoryController, and MyProfileController to access user data without manual parameter passing.
* Helps reduce coupling between controllers while enhancing system security and scalability.

## 4. Product Reviews

## 4.1 Strengths

### 1. Correct and Complete Application of Object-Oriented Programming (OOP)

* Clearly separated classes: Product, User, CartItem, OrderSummary, etc.
* Proper use of **inheritance**, **encapsulation**, **abstraction**, and **polymorphism**.
* Easy to extend and maintain; fully aligns with modern OOP standards.

### 2. Standard MVC Architecture Design

* View: JavaFX FXML
* Controller: Handles logic
* Model: POJO data classes
* Clear modular structure: controller, model, util, connect
* Clean separation of UI and logic → easy to test, easy to reuse.

### 3. Friendly and Dynamic User Interface

* Built with JavaFX and styled using CSS.
* Background video, modern login UI, and clearly divided layout.
* Products displayed using card format, with representative images and user-friendly interactions.

### 4. Complete Feature Set

* Login/Logout with session tracking.
* Product, order, and personal profile management.
* Search, pagination, add/remove from cart.
* Detailed order history available.

### 5. Integrated Statistical Charts with JFreeChart

* Pie charts show product distribution by category.
* Helps administrators monitor and manage activities effectively.

### 6. Robust Authentication & Session Management

* UserSession.java stores the current logged-in user.
* Role-based access control (Admin / User).
* Temporary account lockout after multiple failed login attempts.

### 7. Thorough Functional Testing

* Clear and consistent test codes (e.g., KTC-LOGIN, KTC-PROFILE).
* Covered all scenarios: valid, invalid, locked, and restored accounts.
* Testing results meet all requirements in logic, UI design, and security.

## 4.2 Weaknesses

### 1. Lack of Real Payment Functionality

* No integration with bank cards, Momo, or other payment gateways.
* Orders are saved to the database, but there's no payment confirmation step.

### 2. Basic Order Management

* Each order is stored per product (order\_id, product\_id), rather than as a grouped shopping cart.
* Missing a "View Order Details" feature that shows multiple products in a single order.

### 3. No Network/Online Connectivity

* This is a JavaFX desktop application; it cannot be accessed via a web browser.
* Not suitable for real-world internet/cloud deployment in its current form.

### 4. Limited Access Control

* Although UserSession exists, the interface lacks fine-grained access restrictions.
* For example, a staff member could potentially access admin pages directly via FXML links.

### 5. No Actual Password Encryption

* While password hashing is mentioned, the implementation of SHA-256, BCrypt, or other encryption libraries is unclear.
* Storing plain-text passwords poses a significant security risk.

### 6. UI Supports Only One Language

* Some interface content is mixed in English, Russian, and Vietnamese.
* No language-switching feature, which may confuse users unfamiliar with certain languages.

### SUMMARY

The “Online Shopping 2025” project successfully developed a model application simulating the operations of a basic e-commerce platform using JavaFX. The system allows users to log in, browse and search for products, add items to the cart, view order history, and manage personal information. For administrators, the application supports product and user management, as well as data visualization through intuitive charts.

The application effectively applied key techniques and technologies, including:

* **Object-Oriented Programming (OOP)**: Designed logical classes with inheritance, polymorphism, and encapsulation to ensure a clear, extensible software structure.
* **MVC Model**: Separated interface, control logic, and data to enhance system maintainability.
* **JavaFX**: Built a modern graphical interface with efficient data binding.
* **MySQL + JDBC**: Enabled efficient data storage and retrieval through database connectivity.
* **JFreeChart**: Provided visual data representation via pie and bar charts.
* **Java Collections**: Supported search, filtering, cart, and order management functionalities.

Through the project, programming skills, software design thinking, and database manipulation abilities were enhanced, while providing deeper insight into the real-world operations of e-commerce systems.

**PHIẾU CHẤM**

Họ và tên SV1: Nguyễn Hồ Hồng Sơn…………………………… Lớp: 24CNTT……

Họ và tên SV2: Trần Chí Nguyện………………………………… Lớp: 24CNTT……

Tên Đồ án/Tiểu luận:

**Đánh giá**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TT** | **Tiêu chí** | **Thang điểm** | **Điểm chấm** | **Ghi chú** |
| 1 | ***Hình thức trình bày:***  - Trình bày đúng quy định hướng dẫn (font, số trang, mục lục, bảng biểu, danh mục tài liệu tham khảo …).  - Không lỗi chính tả, lỗi đánh máy, lỗi trích dẫn tài liệu tham khảo.  - Trình bày đẹp, văn phong sáng, không tối nghĩa. |  |  |  |
| 2 | ***Nội dung*** | | | |
| Chương 1 |  |  |  |
| Chương 2 |  |  |  |
| Chương 3 |  |  |  |
| Chương 4 |  |  |  |
| Chương 5 |  |  |  |
| 3 | **TỔNG ĐIỂM** | |  |  |

**Điểm chữ: ………………………………………… (Làm tròn đến 1 số thập phân)**

**Giảng viên**

**TS. Lê Ngọc Hiếu**