Ain Shams University
Faculty of Engineering
Department of Computer and Artificial Intelligence Engineering

---

Object-Oriented Computer Programming
Course Code: CSE241

Submitted by:

Karim Samer Mostafa 23p0439

Youssed Atef Elnaggar 23p0341

Nourhan Hesham Elsayed 23p0442

Jana Tamer Maher 23p0173

Omar Gamil Anwar 23p0438

Instructor:

Dr. Mahmoud Ibrahim Khalil

Project GitHub Link : https://github.com/karimsamer100/OOPprj

Google Drive link(for the explanation video) :

https://drive.google.com/drive/u/0/folders/1B7pCeJQEhqRXcajQvPD3YDGTYL86btyz

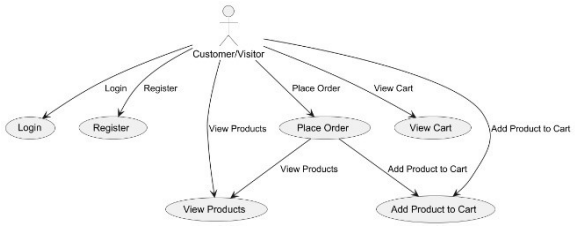
Contents

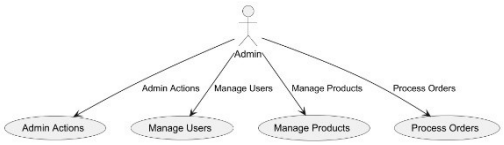
E-Commerce Backend System Implementation Using Object-Oriented Programming (OOP)

Introduction

E-commerce has quickly advanced, reshaping how businesses interact with customers. This has made it crucial to have efficient backend systems to handle tasks like user authentication, inventory, orders, and administrative duties. This report examines the development of an e-commerce backend using Object-Oriented Programming (OOP) principles, implemented in JavaFX. The system is designed with modular components to ensure scalability and ease of maintenance. Essential features, such as error handling and input validation, are included to create a reliable and user-friendly platform for both customers and administrators.

Initially, we will explore the structure of the e-commerce backend system, focusing on the design and implementation of essential components such as customers, admins, products, orders, and carts. We will also discuss the key features, the use of OOP principles, and the overall architecture that drives the system's functionality.





System Design and Features

The e-commerce backend system is built around several key entities, each represented as a class. These classes interact with each other to provide functionalities like customer management, product catalog management, order placement, and administrative control. Below is a detailed explanation of the classes and how they contribute to the system's overall structure and behavior.

Customer Class

The Customer class represents the consumer or user of the e-commerce platform. The attributes of this class are designed to model a real-world customer's profile.

1. **Attributes:**
    - **username:** A unique identifier for the customer, essential for login and authentication.
    - **password:** A secure password for the customer's account (best practice would be to hash this password for security reasons).
    - **Date Of Birth:** The customer's birth date. This information can be used for age-related discounts or restrictions.
    - **balance (store Credit):** The customer's store credit, which can be used to make purchases.

- **address:** The delivery address for orders placed by the customer.
- **gender:** An enumerated value representing the gender (either MALE or FEMALE).
- **interests:** A list of interests (like shopping preferences or categories) that can be used to recommend products to the customer.

2. **Methods:**
    - Display Info(): This method provides an abstraction for displaying a customer's details. It is overridden in the Customer class, allowing the system to print out the customer's details in a readable format.

3. **OOP Principles Applied:**
    - Encapsulation: The attributes of the customer are private to prevent direct manipulation, and getters/setters are used to manage access.
    - Polymorphism: The displayInfo() method can be invoked on different types of users (like Admin or Customer), and the specific version of the method is executed depending on the object type.

Admin Class

The Admin class represents administrative users who have privileged access to manage the e-commerce platform. This class provides functionality to manage users, products, and orders.

1. **Attributes:**
    - username: The admin's unique identifier.
    - password: Admin's secure password (again, ideally hashed).
    - dateOfBirth: Birth date of the admin.
    - role: The role of the admin (e.g., "Product Manager", "Order Manager"), which defines their specific duties.
    - workingHours: The number of hours the admin is scheduled to work, which can be useful for operational purposes.
2. **Methods:**
    - Display Info(): Displays detailed information about the admin's profile.
    - Show Users(List<Customer> customers): Displays all the customers in the system.
    - Show Products(List<Product> products): Displays the list of available products in the system.
    - Show Orders(List<Order> orders): Displays all the orders that have been placed.
    - Create Product(Scanner scanner): Allows the admin to add new products to the product catalog.

- Update Product(Scanner scanner): Enables the admin to modify the details of an existing product.
- Delete Product(Scanner scanner): Enables the admin to remove a product from the catalog.

3. **OOP Principles Applied:**
    - Encapsulation: The sensitive data like password is private, and the admin can only access their own information through controlled methods.
    - Abstraction: Admin actions such as managing users and products are abstracted into specific methods that hide the internal complexity.

Category and Product Classes

The Category and Product classes are used to organize and represent products in the system.

The Category class is a way of organizing products into logical groups, making it easier for customers to find what they are looking for. The Product class represents an individual product available for purchase.

1. **Attributes:**
    - id: A unique identifier for the category.
    - name: A name that represents the category, such as "Electronics", "Clothing", or "Books".

Product Class:

1. **Attributes**:
    - name: The name of the product.
    - price: The price of the product.
    - category: A Category object representing the product's category.
2. **Methods**:
    - CRUD operations (create, read, update, delete) allow the admin to manage products in the system
3. **OOP Principles Applied:**
    - Encapsulation: The product details are encapsulated within the class, preventing direct manipulation and ensuring proper validation during CRUD operations.

Cart and Order Classes

The Cart and Order classes are central to the process of managing customer purchases.

Cart Class: The Cart class models a shopping cart where products are temporarily stored before an order is placed.

1. **Attributes**:
   - products: A list of Product objects that the customer has added to their cart.
2. **Methods**:
   - addProduct(Product product): Adds a product to the cart.
   - viewCart(): Displays all the products currently in the cart.
   - calculateTotal(): Calculates the total cost of the products in the cart.
   - clearCart(): Clears all items from the cart once the order is placed.

Order Class: The Order class represents a finalized purchase.

1. **Attributes:**
   - details: A string that holds order information such as the products ordered, payment method, and customer.
   - timestamp: The time the order was placed.

2. **OOP Principles Applied:**
   - Abstraction: The cart and order systems abstract away the complexity of the purchase process. Customers interact with simple methods such as adding products or placing orders, while the internal workings (calculating totals, storing orders) are hidden.

Database Class: In-Memory Data Storage for E-Commerce System

The Database class in the e-commerce backend system is responsible for acting as the central repository of all the data required by the system. This includes storing information about customers, admins, products, categories, and orders. Since this system is designed for in-memory storage, the Database class uses static ArrayList objects to manage and hold the data for different entities throughout the runtime of the application. This ensures that all data related to the application is readily available to the system's various components, such as customer and admin operations.

Attributes of the Database Class

The Database class contains four main attributes, each serving a distinct purpose for the storage and management of the system's data:

1. **customers (Static ArrayList<Customer>):** The customers attribute is a static ArrayList that holds all the Customer objects who have registered with the e- commerce platform. Each Customer object contains personal details, including the customer's username, password, balance, address, gender, interests, and other relevant attributes. Since the customers list is static, it persists across different instances of the Database class, ensuring that the list remains consistent and accessible throughout the application's lifecycle.

2. **admins (Static ArrayList<Admin>):** Similar to the customers list, the admins attribute is a static ArrayList that stores all the Admin objects. Each Admin object contains administrative details such as username, password, role, and working hours. The static nature of this list allows admins to be consistently managed across the entire system, ensuring that access control remains secure.

3. **products (Static ArrayList<Product>):** The products attribute is a static ArrayList designed to hold all the available products that customers can purchase. Each Product object in the list contains relevant information such as the product's name, category, price, description, and quantity available in stock. The products list is a key part of the e-commerce system, enabling customers to browse and purchase products while allowing admins to manage the catalog.

4. **orders (Static ArrayList<Order>):** The orders attribute is a static ArrayList that stores all the completed orders placed by customers. Each Order object contains details about the products ordered, the payment method used, the order status, and the customer who made the order. This attribute provides the historical record of all transactions, allowing both customers and admins to view past orders and manage fulfillment.

Methods of the Database Class

The Database class includes a key method that is responsible for initializing and populating the database with sample data for testing and development purposes. The initializeDummyData() method is essential for simulating a real-world e-commerce platform before actual users interact with the system.

1. **Initialize Dummy Data():** This method populates the static ArrayList attributes with sample data. It creates a few dummy Customer, Admin, Product, and Order objects, adding them to their respective lists. For instance, it might create:Dummy Customers: A set of sample customers with different usernames, passwords, addresses, and interests.
    - **Dummy Admins:** A few admin users with roles like "SuperAdmin" or "ProductManager," each having specific working hours and credentials.

This dummy data helps developers and testers simulate real-world interactions with the system, test the functionalities, and ensure everything works correctly without the need for an actual database at the beginning of the project. It can be removed or replaced when the system is ready for production deployment with a full-fledged database.

OOP Principles Applied

The design of the Database class makes extensive use of key Object-Oriented Programming (OOP) principles, particularly Encapsulation.

1. **Encapsulation:** Encapsulation is the core OOP principle applied in the Database class. It ensures that the data is hidden and accessed through well- defined methods, promoting modularity, security, and maintainability. By encapsulating the data in static ArrayList attributes, the class ensures that customers, admins, products, and orders are managed internally, and their data is not directly exposed to other parts of the program. This prevents external manipulation and ensures that data can only be modified or accessed via specific methods that provide controlled interactions.
    - The customers, admins, products, and orders lists are kept private to ensure that no class outside of the Database class can directly modify them. Instead, operations on the data, such as adding, removing, or retrieving elements, are done through appropriate methods like initializeDummyData(), ensuring that the integrity of the data is preserved.
    - The methods that interact with these static lists ensure that the data is manipulated safely and efficiently, using defined rules and validation checks. For example, when an admin adds a new product, the Product object is added to the products list through a well-defined method that checks for valid input and ensures the product is correctly added.

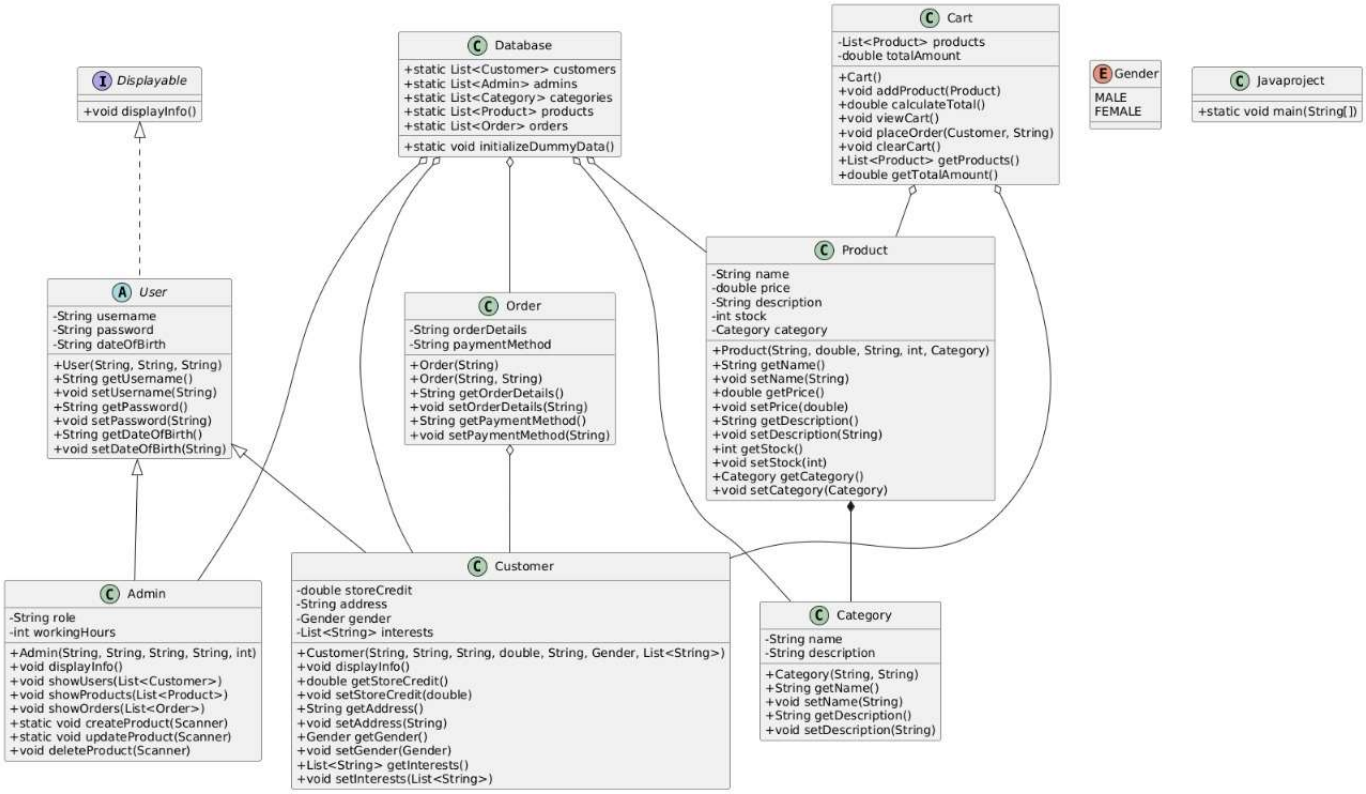
2. **Data Integrity and Security:** By encapsulating the data within the Database class and controlling how the data is accessed and modified, the class also enforces data integrity and security. Unauthorized access to or modification of critical data (such as customer orders or product inventory) is prevented by ensuring that only authorized methods within the Database class can modify the data.

For example, customers cannot directly manipulate the admins or orders lists. Admins, on the other hand, can manage products but cannot modify the data associated with customers' private details. This separation of concerns reduces the risk of bugs or security breaches and ensures that the application functions as intended.

3. **Modularity:** The Database class also demonstrates modularity, another key OOP principle. By grouping the management of all data entities (customers, admins, products, orders) into a single class, the system becomes easier to maintain and extend. If any part of the database functionality needs to change, such as switching from an in-memory implementation to an actual database, the changes can be made in one central location without affecting other parts of the system.
4. **Maintainability:** Using OOP principles like encapsulation and modularity enhances the maintainability of the code. With encapsulated methods controlling the interactions with the static lists, any future updates to the data model (for example, adding more attributes to Customer or Product) can be managed within the Database class. The rest of the system will remain unaffected as long as the database methods are updated to handle these changes.

Main Program (Java project Class)

The Javaproject class serves as the main entry point for the e-commerce backend system. It brings together all the components of the system—customers, admins, products, carts, orders, and the database—into a cohesive user interface that facilitates interaction with the system. This class operates as the bridge between the user and the backend, providing the interface to carry out various actions such as logging in, managing products, handling carts, placing orders, and more.



Key Functionalities in the Main Program

The main program in the Javaproject class is organized around several key functionalities that are essential for both customers and administrators to use the e-commerce platform effectively. Each of these functionalities is mapped to specific actions that the user can take, making it easy to navigate through the system.

1. Login and Registration

   One of the first steps when a user interacts with the system is logging in or registering. The system offers two types of users: customers and admins. Depending on the type of user, the system allows access to different functionalities.
   Login Process:

   The user is prompted to enter their credentials, including a username and password. The system verifies these credentials against the stored data in the database. There are several key steps in the login process:

- Prompting for Username and Password: The program prompts the user to enter their username and password.
- Checking Credentials: Once the credentials are entered, the program searches the customers or admins list (depending on the user type) to check if the credentials match any existing entries in the database.
- Successful Login: If the credentials are correct, the user is granted access to the system. Customers are directed to their main interface, while admins are given access to the admin controls.
- Failed Login: If the credentials are incorrect, an error message is displayed, and the user is prompted to try again.

Registration Process:

For new users (both customers and admins), the system offers a registration option. This process includes:

- Entering User Information: The user provides their personal details, such as username, password, email, date of birth, and any other relevant information.
- Storing Information in the Database: The newly registered user is added to the appropriate list in the database (either the customers or admins list).
- Confirmation Message: After successful registration, the system displays a confirmation message, and the user is redirected to the login page.

The login and registration system ensures that only authorized users can access sensitive features of the platform, such as managing products and orders.

2. Product Management

Product management is a crucial feature of the e-commerce backend system. For customers, the ability to browse and view products is a primary function. For admins, however, the system allows more advanced capabilities, including adding new products, updating existing ones, and deleting obsolete products from the catalog.

Customer Product Management:

For a customer, the primary interaction with the product catalog involves browsing available products. The customer can:

- View Product Details: The customer can view a list of all available products, which include essential information such as the product's name, category, price, and availability.
- Filter Products: The system may allow filtering of products based on categories to enhance the shopping experience.
- Search Functionality: The customer may also search for specific products by name, category, or price range to quickly find what they need.

Admin Product Management:

Admins have the ability to manage the entire product catalog. This functionality provides the admin with full control over the system's inventory, ensuring that products can be added, updated, or removed as necessary. Admins can:

- Add New Products: Admins can enter product details such as the name, price, category, and description. This process also includes specifying the quantity available for sale.
- Update Existing Products: If a product's details change (e.g., price update, quantity change, or description update), the admin can modify the relevant information to keep the catalog up-to-date.
- Delete Products: If a product is discontinued or no longer available for sale, the admin can remove it from the catalog to ensure the customers only see active items.

Error Handling in Product Management:

- Invalid Input: The program checks for invalid input during product addition or updating (e.g., non-numeric values for price, missing product details). If such input is detected, the system displays an error message and prompts the user to enter valid data.
- Product Not Found: When updating or deleting a product, the system checks whether the product exists in the database. If the product is not found, an error message is displayed.

3. Cart Management

The cart is a critical feature for customers in an e-commerce system, allowing them to temporarily store products before placing an order. Cart management involves adding products to the cart, viewing the cart, and calculating the total price of the items.

Customer Cart Features:

- Add Products to Cart: Customers can add products to their shopping cart with a simple command. The product is added to a list in the cart, and the cart is updated with the new product's details.
- View Cart: Customers can view all the products currently in their cart, along with details such as product name, quantity, price per item, and total price.
- Remove Products from Cart: If a customer changes their mind about a product, they can remove it from the cart.
- Proceed to Checkout: When the customer is ready to purchase, they can proceed to checkout, where they will select their preferred payment method and place the order.

Order Placement and Payment:

Once the customer is satisfied with their cart, they can proceed to place an order. The payment method can be selected from a variety of options, such as:

- Store Credit: The customer can pay using the balance in their store account.
- Credit Card: The customer can use a credit card to complete the purchase.
- PayPal: The customer can also use PayPal to process the payment.

Once the payment is successful, the system generates an order, deducts the corresponding balance from the customer's store credit (if applicable), and clears the cart.

4. Admin Controls

Admins have a comprehensive set of tools to manage the e-commerce platform. The admin interface provides the following functionalities:

- View All Users: Admins can view a list of all customers registered.
- View All Products: Admins can view a complete list of products in the catalog.
- View All Orders: Admins can view a list of all orders placed by customers, along with the payment status and any other relevant details.

Additionally, admins can take action on these records, such as modifying user details, updating product information, and addressing issues related to orders. This level of access ensures that admins can effectively manage the platform's operations.

Error Handling in Admin Controls:

Invalid Operations: Admin actions, such as deleting a product or modifying an order, are validated before execution. If an invalid operation is detected, such as trying to delete a non-existing product, the system displays an error message and asks the admin to re-enter the action.

5. Error Handling in the Main Program

Error handling is a critical aspect of any application, and the Javaproject class ensures that the system runs smoothly even in the event of unexpected errors. Several types of errors are anticipated and handled within the system, including:

Input Validation:

The system continuously validates user input to ensure that it is correct and appropriate. For example:

- Non-numeric Values for Price: When an admin tries to enter a product's price, the system checks if the input is a valid number.
- Invalid Username or Password: When a user logs in, the system verifies that the entered credentials match those stored in the database. If not, it prompts the user to try again.

Exception Handling:

The program uses try-catch blocks to handle exceptions that could arise during operations such as reading input, accessing the database, or performing calculations:

- Database Access Errors: If the program encounters any issues while accessing the database (e.g., trying to retrieve data from an empty list), an exception is thrown and caught, displaying a relevant error message to the user.
- Unexpected Errors: Other unforeseen errors, such as incorrect method calls or null reference errors, are caught by the catch block and handled gracefully, ensuring the program doesn't crash.

User Experience:

The error messages are designed to be user-friendly, helping the user understand the issue and providing clear instructions on how to proceed. For example, if a customer tries to add more products than are available in stock, the system will display an error message informing them of the issue.

User Class

The User class is a foundational element in the e-commerce system, designed to encapsulate the shared attributes and behaviors of all system users. It acts as a parent class for specific user types, such as Customer and Admin, promoting code reusability and maintainability.

The class includes private attributes for the user's username, password, and dateOfBirth, ensuring data security and encapsulation. These attributes are accessed and modified using public getter and setter methods. The class constructor initializes these attributes, allowing the creation of User objects with specific details.

Additionally, the displayInfo() method serves as a placeholder for displaying user information, which can be overridden by subclasses to provide more specific details about the user type. This method lays the groundwork for polymorphism, where derived classes can implement customized behaviors.

By encapsulating common properties and providing a base structure, the User class simplifies the management of user-related functionality across the system.

Functionalities Overview

Functionalities of the e-commerce system designed with JavaFX, where each feature includes an FXML file for the user interface and a controller file for its logic. All FXML files incorporate a title label centered at the top, displaying "E-Commerce" to ensure a consistent and user-friendly interface.

The e-commerce system offers the following functionalities:

1. Main Page
2. Login Choices
3. Register Choices
4. Customer Login
5. Admin Login
6. Customer Registration
7. Admin Registration
8. Add to Cart
9. View Cart
10. Place Order
11. Admin Actions
    - View Users
    - View Products
    - View Orders
    - Create Product
    - Update Product
    - Delete Product
    - Create Category

Main Page

The main page serves as the entry point for the system, providing users with navigation options such as login, registration, viewing products, managing the cart, placing orders, and accessing admin functionalities.

System Design

The Main Page provides users with several options:

1. Login
2. Register
3. View Products
4. Add to Cart
5. View Cart
6. Place Order
7. Admin Actions

Implementation of Main Page

The MainPageController class is responsible for handling all user interactions and switching between scenes based on the button clicked.

Key Functionalities

1. Login Button:
    - Description: Redirects the user to the Login page to choose between customer and admin login.
    - Implementation: The handleLoginClick() method is invoked when the user clicks the "Login" button. It loads the LoginChoices.fxml file and changes the scene.
2. Register Button:
    - Description: Redirects the user to the Registration page to choose between customer and admin registration.
    - Implementation: The handleRegisterClick() method loads the RegisterChoices.fxml and sets the scene.
3. View Products Button:
    - Description: Displays available products to the user.
    - Implementation: The handleViewproductClick() method loads the Viewproducts.fxml and sets the scene.

4. Add to Cart Button:
    - Description: Allows users to add products to their cart, but requires them to be logged in.
    - Implementation:
        - The handleADDproductrocartClick() method checks if a customer is logged in using Database.getLoggedInCustomer().
        - If no customer is logged in, it redirects to the customerlogin.fxml.
        - If the user is logged in, it loads the Addtocart.fxml scene.
5. View Cart Button:
    - Description: Displays the user's shopping cart.
    - Implementation:
        - The handleViewCartClick() method checks if a customer is logged in using Database.getLoggedInCustomer().
        - If no customer is logged in, it redirects to the customerlogin.fxml.
        - If logged in, it loads the viewcart.fxml.
6. Place Order Button:
    - Description: Allows the user to place an order for the products in their cart.
    - Implementation:
        - The handlePlaceOrderClick() method works similarly to the previous buttons by checking if a customer is logged in.
        - If logged in, it loads the placeorder.fxml scene.
7. Admin Actions Button:
    - Description: Provides admin functionalities but requires admin login.
    - Implementation:
        - The handleAdminActionClick() method checks if an admin is logged in using Database.getLoggedInAdmin().
        - If no admin is logged in, it redirects to the Adminlogin.fxml.
        - If logged in, it loads the adminactions.fxml scene.

Login Choices

Allows users to choose between logging in as a customer or as an admin. The interface includes two buttons that redirect users to their respective login screens.

System Design

The interface provides the user with two login options:

- Login as Customer
- Login as Admin

Upon selecting one of these options, the system will direct the user to the respective login screen.

The design consists of a label prompting the user to choose the login type and two buttons: one for customer login and one for admin login.

Implementation of Login Choices

The LoginChoicesController class handles the navigation for customer and admin logins. When the user clicks one of the buttons, the corresponding login screen is loaded.

Key Functionalities

1. Login as Customer
    - Description: Redirects the user to the Customer Login screen.
    - Implementation:
        - The handleCustomerLogin() method is triggered when the "Login as Customer" button is clicked.
        - The FXMLLoader loads the customerlogin.fxml file to display the Customer Login screen.
        - The scene is then switched to the Customer Login screen, and the stage title is set to "Customer Login."
2. Login as Admin
    - Description: Redirects the user to the Admin Login screen.
    - Implementation:
        - The handleAdminLogin() method is triggered when the "Login as Admin" button is clicked.
        - The FXMLLoader loads the Adminlogin.fxml file to display the Admin Login screen.
        - The scene is then switched to the Admin Login screen, and the stage title is set to "Admin Login.

Register Choices

Allows users to choose between registering as a customer or an admin. Two buttons navigate users to the appropriate registration forms.

System Design

The interface provides the user with two options for registering:

- Register as Customer
- Register as Admin

The design includes the following features:

- Two buttons for choosing the registration type: Customer or Admin.
- A title and a subtitle to guide the user.
- A clean, simple layout that centers the buttons and text to make it user-friendly.

The AnchorPane is used to position elements, with the buttons placed below the text labels for ease of navigation. The buttons are linked to actions that navigate the user to the respective registration forms for customer or admin.

Implementation of Register Choices

The RegisterChoicesController class manages the user interactions on the Register Choices page. It handles the actions triggered when the user selects either the "Register as Customer" or "Register as Admin" options. Each action loads the appropriate registration form based on the user's choice.

Key Functionalities

1. Customer Registration
    - Description: Allows the user to register as a customer.
    - Implementation:
        - The handleCustomerRegister() method is triggered when the "Register as Customer" button is clicked.
        - This method loads the customer registration page (customerregisteration.fxml) and switches the scene to the customer registration.
2. Admin Registration
    - Description: Allows the user to register as an admin.
    - Implementation:
        - The handleAdminRegister() method is triggered when the "Register as Admin" button is clicked.
        - This method loads the admin registration page (Adminregisteration.fxml) and switches the scene to the admin registration.

Customer Login

Enables customers to log in by providing their username and password. Incorrect credentials trigger an error message, while successful login redirects users to the main page.

System Design

The interface provides customers with the following actions:

- Input their username in a text field.
- Input their password in a password field.
- Login by clicking the "Login" button.
- Display an error message if login credentials are incorrect.

Implementation of Customer Login

The customerloginController class handles the login process. It authenticates the customer's credentials and navigates to the main page if the login is successful.

Key Functionalities

1. Customer Authentication
    - Description: The admin or customer can log into the system by entering their username and password.
    - Implementation:
        - The handleCustomerLogin() method is triggered when the login button is clicked.
        - It retrieves the username and password entered by the user.
        - The authenticateCustomer() method compares the provided credentials with those stored in Database.customers. If a match is found, the customer is authenticated.
        - If authentication is successful, the logged-in customer is set using Database.setLoggedInCustomer(customer), and the application navigates to the main page (mainpage.fxml).
2. Navigation to Main Page

    - Description: After successful login, the customer is redirected to the main page.
    - Implementation:
        - Once the customer is authenticated, the system loads the mainpage.fxml view.
        - The scene is updated to display the main page.

Admin Login

Enables admins to log in by validating their credentials against the database. Successful login grants access to the admin panel, while invalid credentials prompt an error message.**System Design**

The interface provides admins with the following actions:

- Input their username in a text field.
- Input their password in a password field.
- Login by clicking the "Login" button.
- Display an error message if login credentials are incorrect.

The design uses TextField and PasswordField for input and a Button for triggering the login process. A Label is used to display error messages when the login fails.

Implementation of Admin Login

The AdminloginController class manages the functionality of the admin login interface. It ensures secure login by validating the admin's credentials against the database and provides error handling for incorrect inputs. Successful login redirects the admin to the main admin actions interface, while invalid credentials trigger an error message.

Key Functionalities

- Admin Login
    - Description: Authenticates the admin using their username and password.
    - Implementation:
        - The handleAdminLogin() method retrieves the username and password from the input fields.
        - It validates the credentials using the Database.login() method, which checks for a matching admin user.
        - If the credentials are valid, the admin is redirected to the adminactions.fxml interface.
        - If the credentials are invalid, an error message is displayed in the errorLabel.
- Return to Main Page
    - Description: Allows the admin to navigate back to the main page.
    - Implementation:
        - The handleBackClick() method loads the mainpage.fxml file and switches the scene to the main page.

Customer Registration

Allows new customers to register by filling out their details, such as username, password, address, and interests. Upon successful registration, users are redirected to the main page.

System Design

The interface provides fields for the admin to input their details, including:

- Username (e.g., John William)
- Password
- Store Credit (e.g., 5000)
- Address (e.g., 42-jack st.)
- Interests (e.g., reading,cooking)
- Gender (MALE/FEMALE)

Additionally, a "Register" button is included to submit the entered data. The layout is intuitive and uses TextField components for inputs, paired with Label components for clarity.

Implementation of Customer Registration

The registration process begins when the user fills out the form and clicks the "Register" button. The system validates the input, creates a new Customer object, and adds it to the database.

Key Functionalities

1. User Input Collection and Validation
    - Description: The system collects various details from the user, such as username, password, date of birth, store credit, address, interests, and gender.
    - Implementation:
        - The input is retrieved from the respective text fields: usernameField, passwordField, dateOfBirthField, etc.
        - The input is validated, with the gender input specifically being checked to ensure it is either "MALE" or "FEMALE". If invalid, an error message is printed to the console.
2. Customer Object Creation
    - Description: After validation, a new Customer object is created using the collected details.
    - Implementation:
        - The customer's information is passed to the Customer constructor, which creates the object.
        - The new Customer is then added to the Database.customers list.
3. Navigation to Main Page
    - Description: After a successful registration, the user is redirected to the main page of the application.
    - Implementation:
        - The navigateToMainPage() method is called, which loads the mainpage.fxml view and switches the scene to the main page.

Admin Registration

Enables admins to register by providing details like username, password, role, and working hours. Once registration is complete, the user is redirected to the main page.

System Design

The interface provides fields for the admin to input their details, including:

- Username (e.g., John William)
- Password
- Date of Birth (YYYY-MM-DD)
- Role (e.g., Manager)
- Working Hours (e.g., 9)

Additionally, a "Register" button is included to submit the entered data. The layout is intuitive and uses TextField components for inputs, paired with Label components for clarity.

Implementation of Customer Registration

The AdminRegistrationController class manages the admin registration process. It collects user input, validates the data, and creates a new Admin object. The newly created admin is then added to the database. Upon successful registration, the user is redirected to the main page.

Key Functionalities

- Admin Registration
    - Description: Allows admins to register by providing the necessary details.
    - Implementation:
        - The handleRegister() method collects input from the form fields and validates the data.
        - An Admin object is created using the collected details, and it is added to the Database.admins list.
        - If registration is successful, the user is redirected to the main page.
        - Any errors during registration (e.g., invalid input or missing data) are handled gracefully with appropriate error messages.
- Navigate to Main Page
    - Description: Redirects the user to the main page after successful registration.
    - Implementation:
        - The navigateToMainPage() method uses FXMLLoader to load the mainpage.fxml file and switches the scene to the main page.

Add to Cart

Allows users to browse products, add them to the cart, and view cart contents. Stock availability is checked before adding products to the cart.

System Design

The interface provides users with the following actions for managing their cart:

- Display all available products in a table.
- Select a product to add to the cart.
- Add the selected product to the cart.
- View the cart to see added products.
- Return to the main page.

The design uses a TableView to display product details such as name, price, stock, and category. The interface also includes a TextField to show the selected product, a button to add it to the cart, and navigation buttons for cart viewing and returning to the main page.

Implementation of Customer Registration

The AddToCartController class handles all cart-related operations. It enables users to seamlessly browse and add products to their cart while ensuring proper stock management. It integrates user-friendly navigation and dynamic interaction with the database. Error handling is implemented with notifications to ensure a smooth user experience and prevent invalid actions.

Key Functionalities

- Display Available Products
  - Description: Shows a list of all available products with their details (name, price, stock, category) in a table.
  - Implementation:
    - The initialize() method sets up the TableView with columns for product attributes.
    - The loadProducts() method retrieves products from the database and populates the table.
    - A selection listener updates the selected product field and disables the "Add to Cart" button if the product is out of stock.
- Add Product to Cart
  - Description: Allows users to add the selected product to their cart and updates the product's stock in the database.
  - Implementation:
    - The handleAddToCart() method validates the selected product and checks stock availability.
    - The product is added to the cart, stock is decremented, and changes are reflected in the table.
- View Cart
  - Description: Navigates to the cart view where users can see all products they have added.
  - Implementation:
    - The handleViewCart() method loads the viewcart.fxml file and switches the scene.
- Return to Main Page
  - Description: Returns the user to the main page.

- Implementation:
  - The handleConfirmButtonClick() method loads the mainpage.fxml file and switches the scene.

View Cart

Displays the list of products in the cart, including their name, price, and description. Users can remove individual items, clear the cart, or proceed to checkout.

System Design

The interface is designed for users to view, manage, and proceed with items in their shopping cart. The design includes the following features:

- Display the list of products added to the cart in a TableView.
- Show the product name, price, and description in the table.
- A label displaying the total amount of the cart.
- Buttons for removing selected items, clearing the entire cart, proceeding to checkout, or returning to the main page.

The layout is based on an AnchorPane, which provides flexibility in positioning elements. The VBox and HBox containers are used for organizing the buttons and total price label. The TableView and TableColumn elements are used to display the cart items.

Implementation of View Cart

The ViewCartController class handles the cart's functionality, including displaying cart items, removing items, clearing the cart, and proceeding to checkout. It also manages enabling and disabling buttons based on the cart's state.

Key Functionalities

1. Load Cart Data
   - Description: Loads and displays the current cart's items in the TableView.
   - Implementation:
     - The initialize() method loads the cart's data into the TableView using ObservableList.
     - It updates the total price displayed in the totalPriceLabel.
     - The loadCartData() method updates the table with the products in the cart, ensuring the data is dynamically reflected.
2. Update Total Price
   - Description: Displays the total price of the products in the cart.
   - Implementation:
     - The updateTotalPrice() method calculates the total cost by calling the calculateTotal() method from the Cart class and updates the totalPriceLabel.
     - This ensures that the user is always informed of the total price as they modify the cart.

3. Remove Item from Cart
    - Description: Removes the selected item from the cart after confirmation.
    - Implementation:
        - The handleRemoveItem() method is triggered when the "Remove Selected" button is clicked.
        - It prompts the user with a confirmation dialog before removing the item from the cart.
        - After confirming, the selected product is removed, and the cart is updated.
4. Clear Cart
    - Description: Clears all items from the cart after user confirmation.
    - Implementation:
        - The handleClearCart() method is triggered when the "Clear Cart" button is clicked.
        - It asks the user for confirmation before clearing all items from the cart.
        - The method also restores the stock of all items and clears the cart.
5. Proceed to Checkout
    - Description: Proceeds with the checkout process if the user is logged in.
    - Implementation:
        - The handleCheckout() method is triggered when the "Proceed to Checkout" button is clicked.
        - It first checks if the user is logged in by calling Database.getLoggedInCustomer(). If the user is not logged in, a warning message is shown.
        - If the user is logged in, it proceeds to create a new order and adds it to the database, followed by clearing the cart.
6. Return to Main Page
    - Description: Allows the user to return to the main page without any changes.
    - Implementation:
        - The handleBackToMainPage() method is triggered when the "Back to Main Page" button is clicked.
        - This method loads the mainpage.fxml file and switches the scene to the main page.
7. Button State Management
    - Description: Updates the state of buttons based on the current status of the cart.
    - Implementation:
        - The updateButtonStates() method ensures that buttons are enabled or disabled appropriately:
            - The "Remove Selected" button is enabled only if an item is selected in the cart.
            - The "Clear Cart" and "Proceed to Checkout" buttons are enabled only if the cart is not empty.
            - If the cart is empty, all action buttons are disabled.

Place Order

Enables users to finalize purchases by selecting a payment method (e.g., store credit or PayPal). After placing an order, the cart is cleared, and a confirmation message is displayed.**System Design**

The interface allows a customer to place an order from their shopping cart. The design includes the following features:

- Select a payment method: Store Credit, Credit Card, or PayPal.
- View a summary of the items in the cart.
- Display the total amount for the order and available store credit.
- Place the order by confirming the payment method and clearing the cart.
- Navigate back to the main page or cancel the order.

The design uses a ListView to display cart items, several RadioButton options to choose the payment method, and Label elements to show the total amount and available store credit. It also includes buttons for placing the order, canceling, and navigating back to the main page. The TextArea is used to display order confirmation messages.

Implementation of Place Order

The PlaceOrderController class handles the order placement process. It updates the cart view, validates the selected payment method, and processes the order details. It ensures that the correct payment method is used, the store credit is sufficient, and that the cart is cleared once the order is placed. Error handling and user feedback are integrated to guide the customer through the process.

Key Functionalities

1. View Cart Summary
    - Description: Displays a list of all items in the cart with their names and prices, along with the total amount for the order.
    - Implementation:
        - The updateCartDisplay() method populates the cartItemsListView with the names and prices of products from the current cart.
        - The totalAmountLabel shows the total price of all the items in the cart.
2. Select Payment Method
    - Description: Allows the customer to choose a payment method (Store Credit, Credit Card, or PayPal).
    - Implementation:
        - The customer selects one of the three payment methods using the RadioButton options.
        - The creditCardRadio is set as the default selection.
3. Place the Order
    - Description: The customer can place an order after confirming the payment method and ensuring the cart is not empty.
    - Implementation:
        - The handlePlaceOrder() method is triggered when the "Place Order" button is clicked.
        - The method checks if the cart is empty. If so, an error message is shown.
        - The selected payment method is retrieved, and if the payment is via store credit, it is verified that the customer has enough credit.
        - If the payment method and store credit are valid, an order is created and saved to the Database.orders list.

        - The cart is cleared, and store credit is updated (5% of the order total is added as earned store credit).
        - A confirmation message is displayed in the orderConfirmationText area.
4. Cancel the Order
    - Description: The customer can cancel the order and close the window.
    - Implementation:
        - The handleCancel() method closes the order window without placing an order.
5. Navigate to Main Page
    - Description: Allows the customer to return to the main page without placing an order.
    - Implementation:

The goToMainPage() method loads the mainpage.fxml file and switches the scene to the main page, ensuring seamless navigation.

Admin Actions

Provides admins with a range of management functionalities.

System Design

The interface provides the admin with the following distinct actions:

- Show all users
- Show all products
- Show all orders
- Create a new product
- Update an existing product
- Delete a product
- Create a category

Implementation of Customer Registration

The AdminActionsController class handles all admin operations. It was designed to be user-friendly and efficient. Navigation between different views is seamless, achieved by using FXMLLoader to load FXML files dynamically. To ensure a smooth experience, error handling is implemented with try-catch blocks to prevent crashes during file loading or database operations.

Key Functionalities

- View Users
    - Description: Displays a list of all registered users.
    - Implementation:
        - The handleShowUsers() method loads the viewusers.fxml file.
        - Data is displayed in a tabular format using JavaFX TableView.
- View Products
    - Description: Displays all available products with their details (name, price, category, etc.).
    - Implementation:
        - The handleShowProducts() method loads the viewproducts.fxml file.
        - Data is fetched from the database and displayed.
- View Orders
    - Description: Displays all orders placed by customers.

    - Implementation:
        - The handleShowOrders() method loads the vieworders.fxml file.
        - Orders are retrieved from the database and displayed with details such as the customer, total amount, and status.
- Create Product
    - Description: Allows admins to add new products to the catalog.
    - Implementation:
        - The handleCreateProduct() method loads the createproduct.fxml file.
        - Inputs are validated before adding the product to the database.
- Update Product
    - Description: Modifies existing product details (e.g., price, stock).
    - Implementation:
        - The handleUpdateProduct() method loads the updateproduct.fxml file.
        - New details are validated before updating the product in the database.
- Delete Product
    - Description: Deletes a product from the catalog.
    - Implementation:
        - The handleDeleteProduct() method loads the deleteproduct.fxml file.
        - The product is identified by its name or ID before deletion.

View Users

Lists all registered users (customers and admins) with their details.

System Design

The interface is designed to provide the admin with the following actions:

- View all system users, divided into two categories**:** Customers and Administrators.
- Navigate back to the main admin panel.

Implementation of View Users

The ViewUsersController class handles the display of users. It organizes the users into two separate categories (customers and administrators) and presents them in a scrollable format.

Key Functionalities

1. Display Customers
    - Description: Displays a list of all customers, showing relevant information such as username, address, gender, and interests.

    - Implementation:
        - The initialize() method loops through the list of customers (Database.customers).
        - For each customer, an HBox is created to hold the customer details, which are displayed in Label elements.
        - The customer information (username, address, gender, interests) is displayed in a neatly formatted VBox within the HBox.
        - The customersVBox is populated with these HBox containers, creating a list of customers on the interface.
2. Display Administrators
    - Description: Displays a list of all administrators, showing details like username, role, and working hours.
    - Implementation:
        - The same initialize() method also loops through the list of administrators (Database.admins).
        - Each administrator is represented similarly with an HBox containing VBox and Label elements for username, role, and working hours.
        - These administrator boxes are added to the adminsVBox, allowing administrators to be listed separately from the customers.
3. Back to Main Panel
    - Description: Allows the admin to navigate back to the main admin panel.
    - Implementation:
        - The gotomain() method is triggered when the "Back to main" button is clicked.
        - This method uses FXMLLoader to load the mainpage.fxml file, which contains the main admin interface.
        - The scene is then switched to the main admin panel, providing a smooth navigation experience for the admin.

View Products

Displays all available products with their attributes.

System Design

The interface is designed to display a list of available products in a dynamic FlowPane, which adapts to the content. The design includes the following features:

- A label indicating the "Available Products" section.
- A FlowPane that dynamically displays the names and prices of the products available in the system.

Implementation of View Products

The ViewProductsController class is responsible for managing the display of products and handling the dynamic loading of products from the database.

Key Functionalities

1. Load Products
    - Description: Loads and displays the list of available products in the FlowPane.
    - Implementation:
        - The initialize() method is called when the scene is loaded.
        - In this method, the controller loops through the list of products retrieved from the Database.products list.
        - For each product, a Label is created to display the product's name and price.
        - Each label is styled with a font and width to ensure proper text wrapping and alignment.
        - The label is then added to the FlowPane, which dynamically arranges the labels based on the available space.

View Orders

Shows all orders placed by customers, including details like payment method.

System Design

The interface is designed to display a list of orders in a TableView, allowing users to view the details of each order and the corresponding payment method. The design includes the following features:

- A header section that displays the title "All Orders."
- A TableView to list all orders, showing two columns: Order Details and Payment Method.
- Buttons for navigating back to the previous page and refreshing the order list.

The layout is built with an AnchorPane, which provides flexibility in positioning the elements. The TableView element is used to display the orders, while TableColumn elements are used to define the columns for order details and payment methods. The action buttons (Back and Refresh) are placed at the bottom for navigation.

Implementation of View Orders

The ViewOrdersController class is responsible for managing the display of orders, refreshing the order list, and handling navigation. It handles the interaction with the database to load and update the list of orders.

Key Functionalities

1. Load Orders
    - Description: Loads and displays the list of all orders in the TableView.
    - Implementation:
        - The initialize() method is called when the scene is loaded. It sets up the columns for the order details and payment method using PropertyValueFactory.
        - The loadOrders() method populates the TableView with data by retrieving orders from the Database.orders list.
        - The ordersTable.setItems() method is used to display the list of orders in the table.

2. efresh Order List
    - Description: Refreshes the order list by reloading the data from the database.
    - Implementation:
        - The handleRefresh() method is triggered when the "Refresh" button is clicked.
        - It calls the loadOrders() method to update the TableView with the most recent data, ensuring that the order list reflects any changes made to the orders.
3. Navigate Back to Admin Actions
    - Description: Navigates back to the admin actions screen.
    - Implementation:
        - The handleBack() method is triggered when the "Back" button is clicked.
        - It loads the adminactions.fxml file and switches the scene to the admin actions page, allowing the user to return to the previous screen.
4. Table Column Setup
    - Description: Defines the columns for displaying order details and payment methods.
    - Implementation:
        - The orderDetailsColumn and paymentMethodColumn are set up to display the order details and payment method for each order using PropertyValueFactory. These properties are mapped to the fields in the Order class, ensuring the correct data is displayed in each column.

Create Product

Adds a new product by specifying attributes like name, price, and category.

System Design

The interface provides the admin with the following actions:

- Add a new product by entering the name, price, description, stock, and selecting a category.
- Navigate back to the admin actions.

The design includes fields for the product name, price, description, stock, and category selection. Additionally, there are two buttons: one for creating the new product and another to return to the admin panel.

Implementation of Create Product

The createproductController class handles the product creation process. It validates the inputs, adds the new product to the database, and provides feedback to the admin.

Key Functionalities

1. Add New Product
    - Description: Allows admins to create a new product by providing the necessary details such as name, price, description, stock, and category.
    - Implementation:
        - The initialize() method populates the categoryComboBox with the names of existing categories from Database.categories.
        - The handleCreate() method is triggered when the "Create Product" button is clicked.
        - The method checks if all fields are filled, and it validates that the price and stock are positive numbers.

        - If the input is valid, a new Product object is created and added to Database.products.
        - A success message is displayed to the admin.
2. Return to Admin Actions
    - Description: Navigates the admin back to the main admin actions panel.
    - Implementation:
        - The handleBack() method loads the adminactions.fxml file. It switches the scene back to the admin actions, ensuring smooth navigation.

Update Product

Allows modification of existing product details.

System Design

The interface allows the admin to update the details of an existing product. The design includes the following features:

- Select a product to update from a dropdown list.
- Edit the product name, price, description, stock, and category.
- A button to update the product.
- A button to return to the admin actions page.

The design uses a ComboBox to select the product, several TextField and TextArea elements to input new product details, and Button elements to trigger the update or return actions. The GridPane layout ensures that all input fields are neatly organized in rows and columns.

Implementation of Update Product

The UpdateProductController class handles the product updating process. It populates the fields with the existing product details when a product is selected from the ComboBox. The controller ensures that the input is valid, performs the update, and displays a confirmation message. It also handles navigation to the admin actions page.

Key Functionalities

1. Select Product
    - Description: Allows the admin to choose a product to update from the dropdown list.
    - Implementation:
        - The handleProductSelection() method is triggered when the admin selects a product from the productComboBox.
        - The selected product's details are populated into the input fields (nameField, priceField, descriptionField, stockField, and categoryComboBox).
        - Fields are enabled only after a valid product is selected.

2. Update Product
    - Description: Allows the admin to update the selected product's details.
    - Implementation:
        - The handleUpdate() method is triggered when the "Update Product" button is clicked.
        - The method validates the input fields and ensures that the new price and stock values are valid numbers.
        - If the validation passes, the selected product's details are updated, and a success message is shown.
        - The method then navigates back to the admin actions page.
3. Cancel Update and Return
    - Description: Allows the admin to return to the admin actions page without making changes.
    - Implementation:
        - The handleBack() method is triggered when the "Back" button is clicked.
        - This method loads the adminactions.fxml file and switches the scene to the admin actions page.
4. Field Validation
    - Description: Ensures that the input fields are properly filled out and that the new values are valid.
    - Implementation:
        - The method checks if any field is left empty, and if so, it shows an error alert.
        - It ensures that the price and stock are positive numbers and that the category is selected.

Delete Product

Enables deletion of a product from the catalog.

System Design

The interface provides the admin with the following actions:

- Select a product from a dropdown list of available products.
- View the product's details including name, price, description, stock, and category.
- Delete the selected product from the system.
- Navigate back to the admin panel.

The design uses a ComboBox to display the available products, a TextArea to show the product details, and two buttons: one for deleting the product and another to return to the admin panel.

Implementation of Delete Product

The DeleteProductController class handles the product deletion process. It populates the product list, displays the details of the selected product, and confirms the deletion before removing the product. Error handling and user feedback are integrated to ensure smooth interaction with the system.

Key Functionalities

1. Select Product to Delete
    - Description: Allows the admin to select a product from a dropdown list.
    - Implementation:
        - The initialize() method populates the productComboBox with product names from the Database.products list.

        - A listener is added to the productComboBox that updates the product details in the productDetailsArea when a product is selected.
2. View Product Details
    - Description: Displays the details of the selected product, including name, price, description, stock, and category.
    - Implementation:
        - The updateProductDetails() method retrieves the selected product from the Database.products list and formats its details in a readable format.
        - These details are then displayed in the productDetailsArea.
3. Delete Product
    - Description: Allows the admin to delete the selected product from the system after confirming the action.
    - Implementation:
        - The handleDelete() method is triggered when the "Delete Product" button is clicked.
        - It checks if a product is selected, and if not, it shows a warning alert.
        - The admin is prompted with a confirmation dialog to confirm the deletion.
        - Upon confirmation, the product is removed from the Database.products list.
        - A success message is displayed, and the combo box is refreshed.
4. Return to Admin Panel
    - Description: Navigates the admin back to the main admin panel.
    - Implementation:
        - The handleBack() method loads the adminactions.fxml file and switches the scene back to the admin panel.

Create Category

Facilitates the creation of new categories by providing a name and description.

System Design

The interface provides the admin with the following actions:

- View all existing categories in a list.
- Add a new category by entering the name and description.
- Navigate back to the admin panel.

The design uses a ListView to display existing categories, a TextField to enter the new category name, and a TextArea for entering a description. It also includes two buttons: one for creating the new category and another to return to the admin panel.

Implementation of Create Category

The CreateCategoryController class handles the category creation process. It validates user input and ensures that the newly created category is added to the list of categories. Error handling and user feedback are integrated to guide the admin through the process.

Key Functionalities

1. View Existing Categories
    - Description: Displays a list of all existing categories with their name and description.

    - Implementation:
        - The initialize() method calls the refreshCategoryList() function.
        - The refreshCategoryList() method populates the categoriesListView with the names and descriptions of all categories from the Database.categories list.
2. Create New Category
    - Description: Allows admins to create a new category by providing a name and description.
    - Implementation:
        - The handleCreate() method is triggered when the "Create Category" button is clicked.
        - The category name and description are retrieved from the categoryNameField and descriptionArea respectively.
        - Input validation ensures that the category name is not empty, and no duplicate category names exist.
        - If the input is valid, a new Category object is created and added to Database.categories.
        - A success message is displayed, and the form is cleared.
3. Return to Admin Panel
    - Description: Navigates the admin back to the main admin panel.
    - Implementation:
        - The handleBack() method loads the adminactions.fxml file.
        - It switches the scene back to the admin panel, ensuring seamless navigation.

Notifications and Error Handling

The system incorporates robust error handling and user notifications for a seamless user experience. Key examples include:

- Login Errors: Invalid credentials trigger error messages.
- Input Validation: Ensures valid data entry for registration and product management.
- Cart Management: Prevents invalid actions, such as adding out-of-stock items or placing orders with an empty cart.
- Admin Actions: Confirmation dialogs and success alerts guide admins through critical operations like deleting products or creating categories.

Conclusion

The e-commerce backend system combines core functionalities, such as user management, product handling, cart operations, and administrative tools, into a cohesive and efficient platform. By leveraging OOP principles like encapsulation and modularity, the system ensures scalability and maintainability, while robust error handling and input validation enhance stability and user experience. With its intuitive interface and extensible design, this system provides a solid foundation for managing e-commerce operations, offering both customers and administrators a seamless and reliable platform.