深圳大学考试答题纸

(以论文、报告等形式考核专用)  
二○二四～二○二五 学年度第 一 学期

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程编号 | 1500370003 | | 课程名称 | | 数据库系统 | | 主讲教师 | | Basker George | 评分 |  |
| 学号 | 2023040431 | | 姓名 | 刘嘉怡 | | 专业年级 | | 2023计算机科学与技术（创新班）02 | | | |
| 学号 | 2022080182 | | 姓名 | 曹博宇 | | 专业年级 | | 2022计算机科学与技术（创新班） | | | |
|  | | | | | | | | | | | |
| 教师评语：The mark obtained for different categories as per the grading policy are as follows   |  |  |  | | --- | --- | --- | | **序号** | **评分点 （分值）** | **Marks Obtained成绩** | | (1) | 第一部分 : Requirements Analysis（10分） |  | | (2) | 第二部分: ER Diagram/ Nomalization and Convert to Relations (20分） |  | | (3) | 第三部分: SQL DDL/ All the different kinds of DML query from PPT （30分） |  | | (4) | 第四部分: Application System Demonstration （20分） |  | | (5) | 第五部分: Report is completed as per the requirement and PPT（20分） |  | | 合计 100 | |  |     Basker George  10 January 2025 | | | | | | | | | | | |
| 题目： | | Development of Online Shopping System Based on Flet Framework | | | | | | | |  | |

**组员个人总分权重分配表：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **排序** | **姓名** | **学号** | **项目个人权重** | **个人最后得分** | **评分人** |
| **1** | 刘嘉怡 | 2023040431 | 100% |  |  |
| **2** | 曹博宇 | 2022080182 | 100% |  |  |

**任务分工及所完成的状况。**

**（2人总分为190% \*教师评分, 3人总分为270% \*教师评分）**

**个人总分权重分配表**

（1人总分为100%，2人为190%, 3人为270%）

|  |  |  |  |
| --- | --- | --- | --- |
| 排序 | 姓名 | 学号 | 项目个人权重 |
| 1 | 刘嘉怡 | 2023040431 | 100 % |
| 2 | 曹博宇 | 2022080182 | 100 % |

我组成员总共\_\_\_2\_\_\_\_名，权重总和为： 200%

# 1. Experiment Objectives

Design and develop a web-based application tailored to a real-world case of your choice. This application should demonstrate your understanding of web development concepts, database design, and user experience.

# 2. Experimental Environment

### 2.1. Hardware Environment

* **Processor**: Intel Core i5 or higher
* **Memory**: 8 GB RAM or more
* **Storage**: 500 GB HDD or SSD
* **Display**: 1920x1080 resolution

### 2.2. Software Environment

* **Operating System**: Windows 11
* Programming Language: Python 3.8+
  + **Dependencies**: Flet, pymysql
* **Frontend Framework**: Flet 1.0+
* **Database**: MySQL 8.0+
* Development Tools:
  + **IDE**: Visual Studio Code
  + **Database Management Tool**: MySQL Workbench

### 2.3. Network Environment

* Stable internet connection for downloading dependencies and remote database access (if applicable).

# 3. Experimental Tasks

### 3.1. Experiment Tasks

1. **Database Design and Implementation**:
   * Design the database table structures related to users, orders, and refund requests.
   * Create necessary indexes, stored procedures, and triggers to ensure data consistency and automation of business logic.
2. **Frontend User Interface Development**:
   * Develop user order viewing pages, refund application pages, and administrator refund management pages using the Flet framework.
   * Implement intuitive and user-friendly interfaces to enhance the experience for users and administrators.
3. **Backend Business Logic Implementation**:
   * Use Python to handle user requests, interact with the database, and execute business logic through stored procedures.
   * Implement permission management to ensure different roles (users and administrators) have appropriate access rights.
4. **Permission Control and Security**:
   * Implement user authentication and role-based access control to secure the application.
   * Encrypt sensitive data to protect user information.
5. **Testing and Validation**:
   * Conduct functional testing to ensure all modules operate correctly.
   * Perform permission testing to verify access controls.
   * Execute performance testing to assess system behavior under high load.

### 3.2. Experiment Requirements

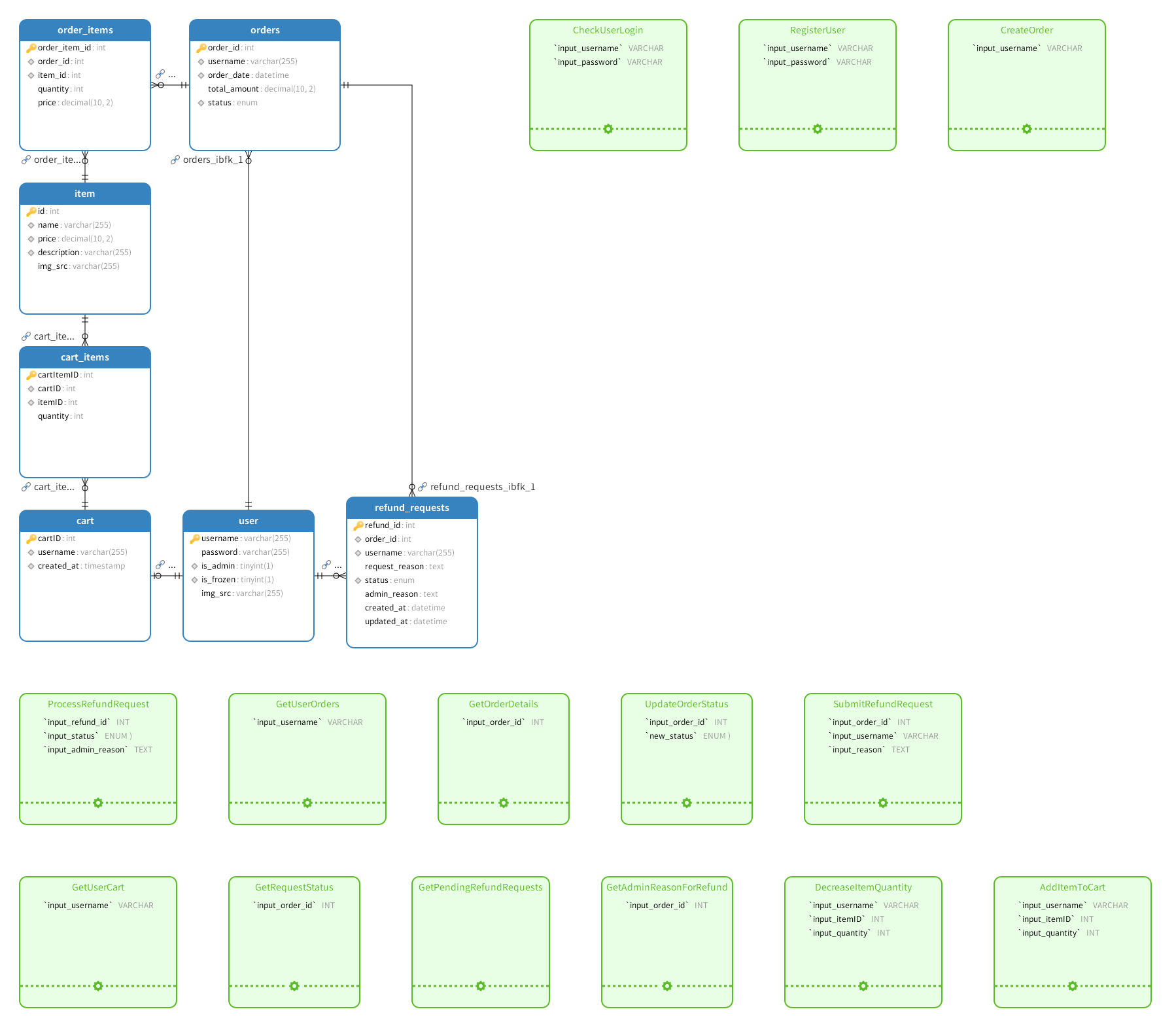
1. **Functional Completeness**: The system should encompass core functionalities such as user order management, refund request submission, and administrator refund request processing.
2. **Data Consistency**: Ensure data integrity and consistency through proper database design and the use of triggers and stored procedures.
3. **Permission Control**: Implement role-based access control to restrict functionalities based on user roles.
4. **User Experience**: Design a clean and intuitive user interface with smooth navigation and responsive feedback mechanisms.
5. **Security**: Protect sensitive data through encryption and safeguard the application against common security threats like SQL injection.
6. **Maintainability**: Ensure the codebase is modular, well-documented, and follows best practices to facilitate future maintenance and scalability.

### 3.3. Experiment Topic

Choose to develop an **Online Shopping System** based on the Flet framework. This project aims to integrate web development, database design, and user experience principles to create a comprehensive e-commerce platform.

# 4. Experimental Content

## a. Database Design



The database comprises:

* **7 Tables**
* **14 Stored Procedures**
* **2 Triggers**
* **Multiple Indexes**

### Key Components:

1. **Tables**:
   * user
   * cart
   * cart\_items
   * item
   * order
   * order\_item
   * refund\_requests
2. **Stored Procedures**:
   * GetPendingRefundRequests
   * GetAdminReasonForRefund
   * SubmitRefundRequest
   * ProcessRefundRequest
   * ... (additional procedures as needed)
3. **Triggers**:
   * after\_refund\_approval
   * after\_user\_insert
4. **Indexes**:
   * Created on frequently queried fields to optimize performance.

## b. Table Descriptions

### user Table

Stores user account information, including username, password, admin status, and the URL of the user's avatar (pointing to a location on a cloud-based image hosting service).

CREATE TABLE `user` (  
 `username` VARCHAR(255) PRIMARY KEY,  
 `email` VARCHAR(255) NOT NULL,  
 `password` VARCHAR(255) NOT NULL,  
 `is\_admin` BOOLEAN NOT NULL DEFAULT FALSE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### cart Table

Each user is associated with a unique cartID.

CREATE TABLE `cart` (  
 `cart\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `username` VARCHAR(255) NOT NULL,  
 FOREIGN KEY (`username`) REFERENCES `user`(`username`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### cart\_items Table

Records the items in a user's shopping cart using cartID and itemID.

CREATE TABLE `cart\_items` (  
 `cart\_item\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `cart\_id` INT NOT NULL,  
 `item\_id` INT NOT NULL,  
 `quantity` INT NOT NULL DEFAULT 1,  
 FOREIGN KEY (`cart\_id`) REFERENCES `cart`(`cart\_id`) ON DELETE CASCADE,  
 FOREIGN KEY (`item\_id`) REFERENCES `item`(`item\_id`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### item Table

Stores item details such as name, price, description, and the URL of the item's image (pointing to a location on a cloud-based image hosting service).

CREATE TABLE `item` (  
 `item\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `name` VARCHAR(255) NOT NULL,  
 `price` DECIMAL(10,2) NOT NULL,  
 `description` TEXT,  
 `image\_url` VARCHAR(500),  
 FOREIGN KEY (`image\_url`) REFERENCES `image\_hosting`(`url`) ON DELETE SET NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### order Table

Associates with the user table via the username field and records the creation time, total amount, and status of a user's order.

CREATE TABLE `order` (  
 `order\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `username` VARCHAR(255) NOT NULL,  
 `status` ENUM('Pending', 'Confirmed', 'Shipped', 'Delivered', 'Canceled') NOT NULL,  
 `total\_amount` DECIMAL(10,2) NOT NULL,  
 `created\_at` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 `canceled\_at` DATETIME DEFAULT NULL,  
 FOREIGN KEY (`username`) REFERENCES `user`(`username`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### order\_item Table

Associates with the order and item tables using orderID and itemID fields to record specific item information within an order.

CREATE TABLE `order\_item` (  
 `order\_item\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `order\_id` INT NOT NULL,  
 `item\_id` INT NOT NULL,  
 `quantity` INT NOT NULL DEFAULT 1,  
 `price\_at\_purchase` DECIMAL(10,2) NOT NULL,  
 FOREIGN KEY (`order\_id`) REFERENCES `order`(`order\_id`) ON DELETE CASCADE,  
 FOREIGN KEY (`item\_id`) REFERENCES `item`(`item\_id`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

### refund\_requests Table

Associates with the order and user tables using order\_id and username fields to record refund requests initiated by users for specific orders. It also records the status of the refund request, the reason provided by the user, and the reasons provided by the administrator for approval/rejection.

CREATE TABLE `refund\_requests` (  
 `refund\_id` INT AUTO\_INCREMENT PRIMARY KEY,  
 `order\_id` INT NOT NULL,  
 `username` VARCHAR(255) NOT NULL,  
 `request\_reason` TEXT NOT NULL,  
 `status` ENUM('Pending', 'Approved', 'Rejected', 'Canceled') NOT NULL DEFAULT 'Pending',  
 `admin\_reason` TEXT,  
 `created\_at` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 `updated\_at` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 FOREIGN KEY (`order\_id`) REFERENCES `order`(`order\_id`) ON DELETE CASCADE,  
 FOREIGN KEY (`username`) REFERENCES `user`(`username`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

## c. Trigger Descriptions

### after\_refund\_approval

* **Purpose**: Automatically updates the status of the associated order to "Canceled" when an administrator approves a refund request, provided the order has not been shipped. If the refund request is rejected, the order status reverts to "Pending".

DELIMITER $$  
  
CREATE TRIGGER `after\_refund\_approval`  
AFTER UPDATE ON `refund\_requests`  
FOR EACH ROW  
BEGIN  
 -- Check if the status has been updated from 'Pending' to 'Approved'  
 IF OLD.`status` = 'Pending' AND NEW.`status` = 'Approved' THEN  
 -- Retrieve the current status of the order  
 DECLARE v\_order\_status ENUM('Pending', 'Confirmed', 'Shipped', 'Delivered', 'Canceled');  
 SELECT `status` INTO v\_order\_status FROM `order` WHERE `order\_id` = NEW.`order\_id`;  
  
 -- Only cancel the order if it hasn't been shipped  
 IF v\_order\_status IN ('Pending', 'Confirmed') THEN  
 -- Update the order status to 'Canceled' and record the cancellation time  
 UPDATE `order`  
 SET `status` = 'Canceled',  
 `canceled\_at` = CURRENT\_TIMESTAMP  
 WHERE `order\_id` = NEW.`order\_id`;  
 ELSE  
 SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Order has been shipped and cannot be canceled.';  
 END IF;  
 ELSIF OLD.`status` = 'Pending' AND NEW.`status` = 'Rejected' THEN  
 -- Revert the order status to 'Pending' if the refund request is rejected  
 UPDATE `order`  
 SET `status` = 'Pending'  
 WHERE `order\_id` = NEW.`order\_id`;  
 END IF;  
END$$  
  
DELIMITER ;

### after\_user\_insert

* **Purpose**: Automatically creates a shopping cart for a new user upon their registration.

DELIMITER $$  
  
CREATE TRIGGER `after\_user\_insert`  
AFTER INSERT ON `user`  
FOR EACH ROW  
BEGIN  
 -- Insert a new cart entry for the newly registered user  
 INSERT INTO `cart` (`username`) VALUES (NEW.`username`);  
END$$  
  
DELIMITER ;

## d. Stored Procedures and Indexes

The project’s database includes numerous stored procedures and indexes to facilitate frontend application implementation and accelerate database query performance.

### Key Stored Procedures

1. **GetPendingRefundRequests**: Retrieves all pending refund requests, including order details.
2. **GetAdminReasonForRefund**: Fetches the administrator’s reason for a refund request if its status is "Canceled" or "Rejected".
3. **SubmitRefundRequest**: Allows users to submit a refund request for an order.
4. **ProcessRefundRequest**: Enables administrators to approve or reject a refund request.

### Index Creation

Indexes are created on frequently queried fields to optimize performance.

-- Creating indexes for refund\_requests table  
CREATE INDEX idx\_refund\_order\_id ON `refund\_requests` (`order\_id`);  
CREATE INDEX idx\_refund\_username ON `refund\_requests` (`username`);  
CREATE INDEX idx\_refund\_status ON `refund\_requests` (`status`);

*Additional indexes can be created similarly based on query patterns.*

## e. Frontend GUI Demonstration

### Home Page



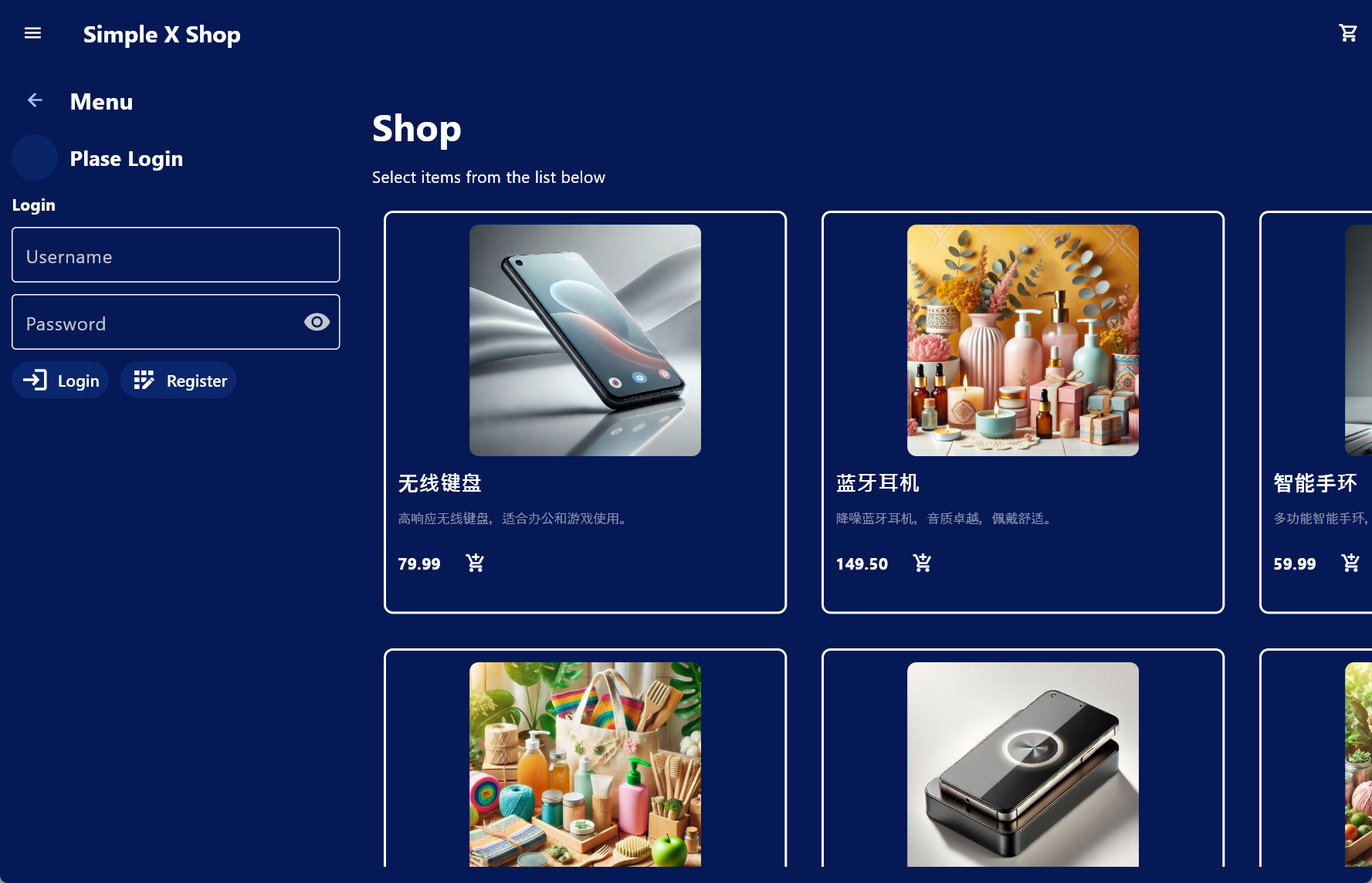
### Main Page

* **Description**: Displays all product data, including product names, descriptions, prices, and images, stored in a cloud database or cloud-based image hosting service.



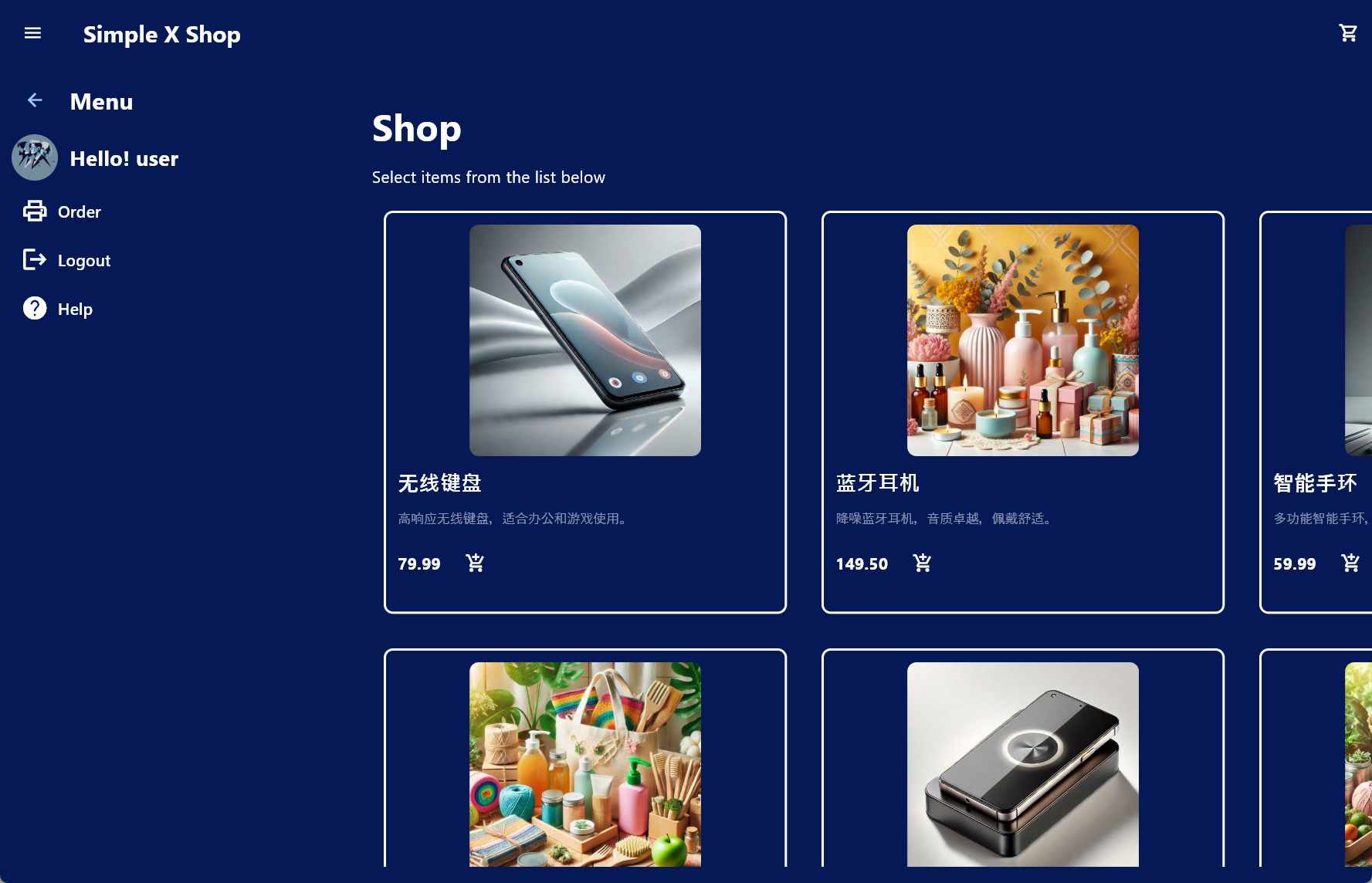
### Menu Interaction

* **Action**: Clicking the menu button at the top left corner.
  + **Result**: A login panel appears on the left side of the main interface.
  + **User Options**: Enter existing credentials or register a new account.
  + **Navigation**: Click the back button at the top left of the login panel to return to the previous page.



### Post-Login Interface

* **Description**: After logging in, the login panel updates to reflect the user’s status. The interface adapts based on the user’s permissions.



### Shopping Cart Interaction

* **Action**: After logging in and selecting products, clicking the shopping cart button at the top right corner.
  + Options
  + :
    - **Left Button**: Submit the cart contents to an order, which also clears the user’s cart.
    - **Right Button**: Navigate to the orders page.



### Orders Page

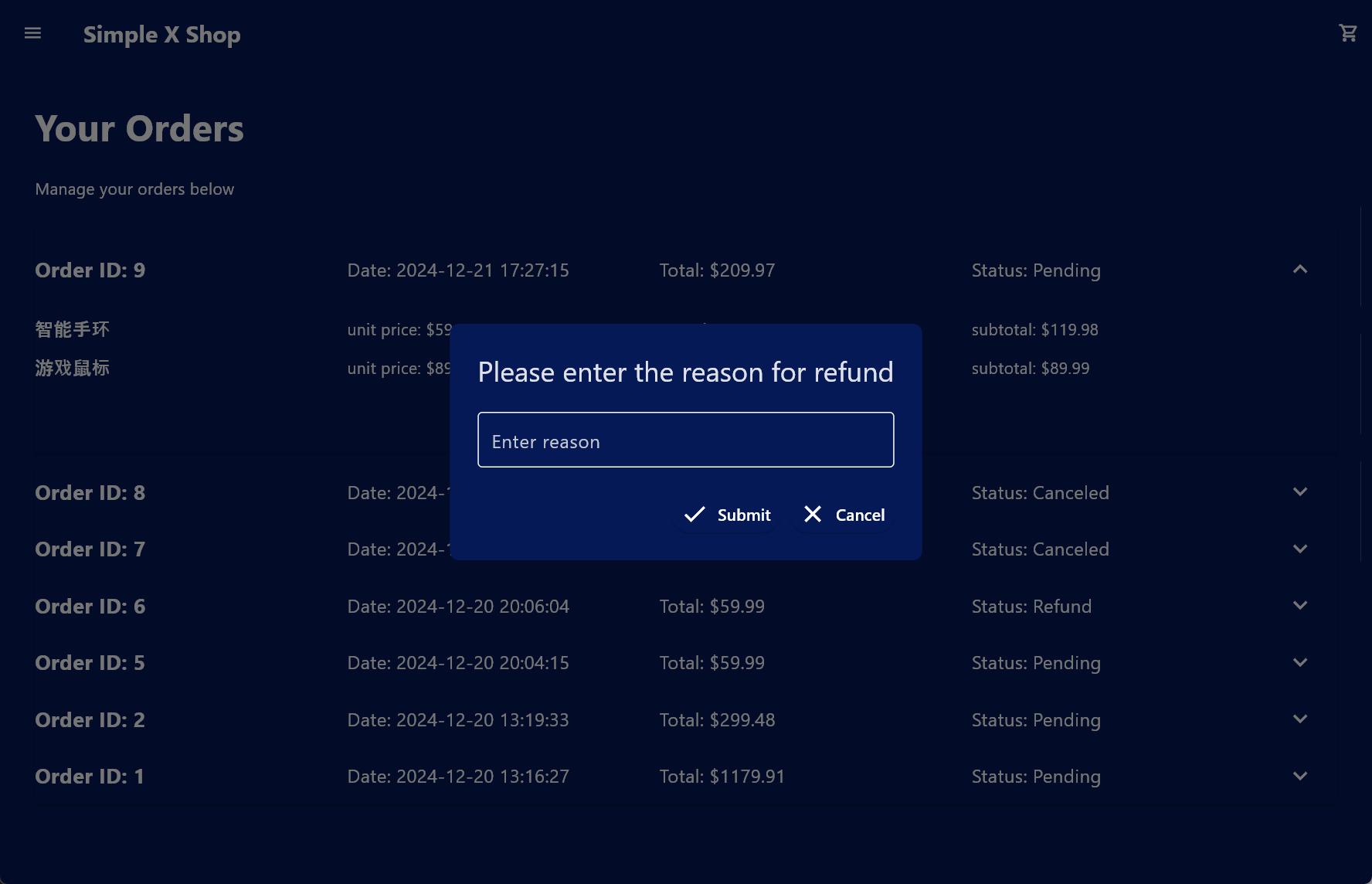
* **Description**: Users can view their order information and order details. They can apply for a refund by clicking the "Request Refund" button. After the refund request is processed by an administrator, users can view the administrator's reason for processing.





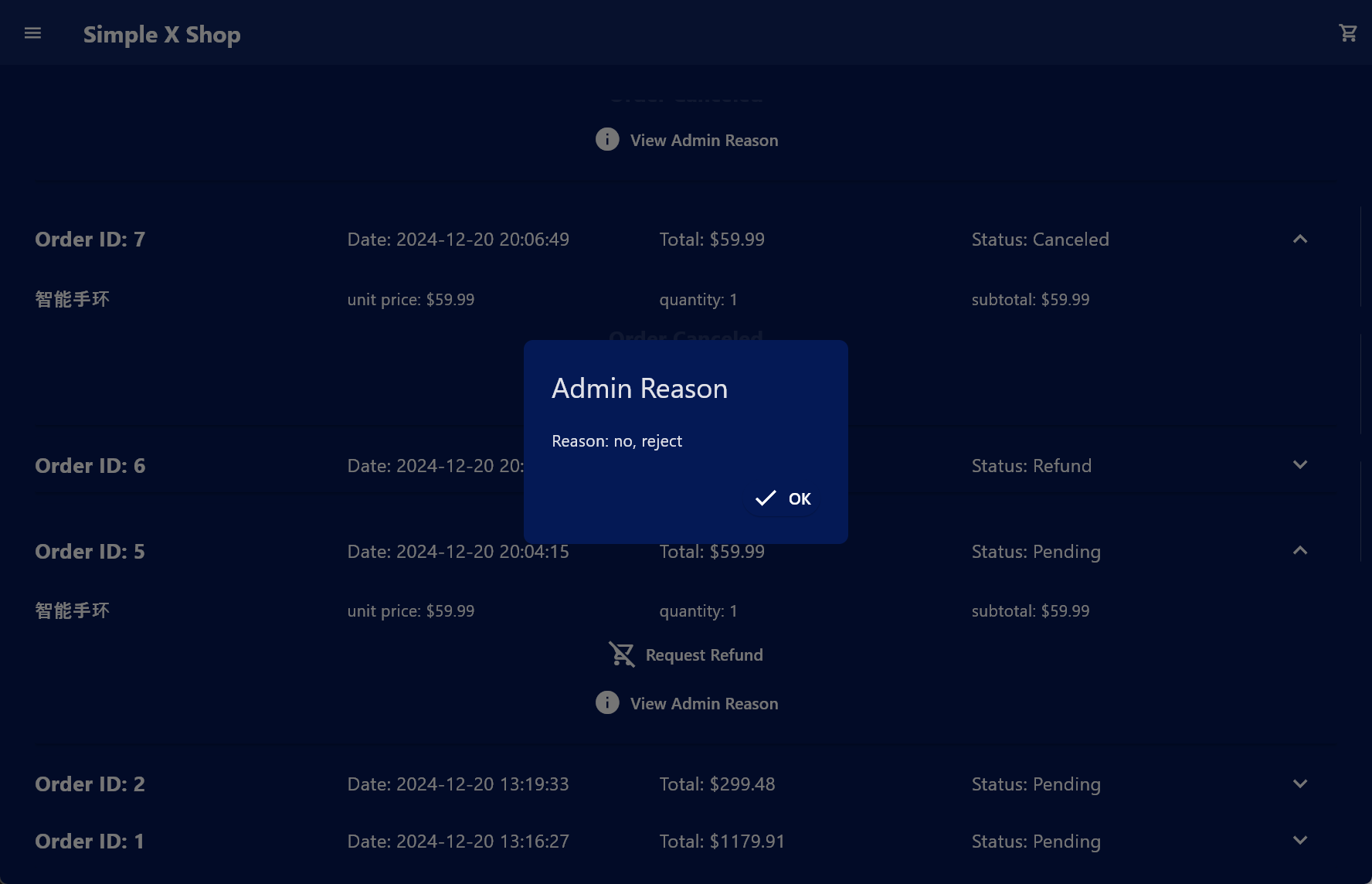
### Refund Request Interface

* **Action**: User initiates a refund request for a specific order.
  + **Requirement**: Users need to provide a reason for requesting a refund.



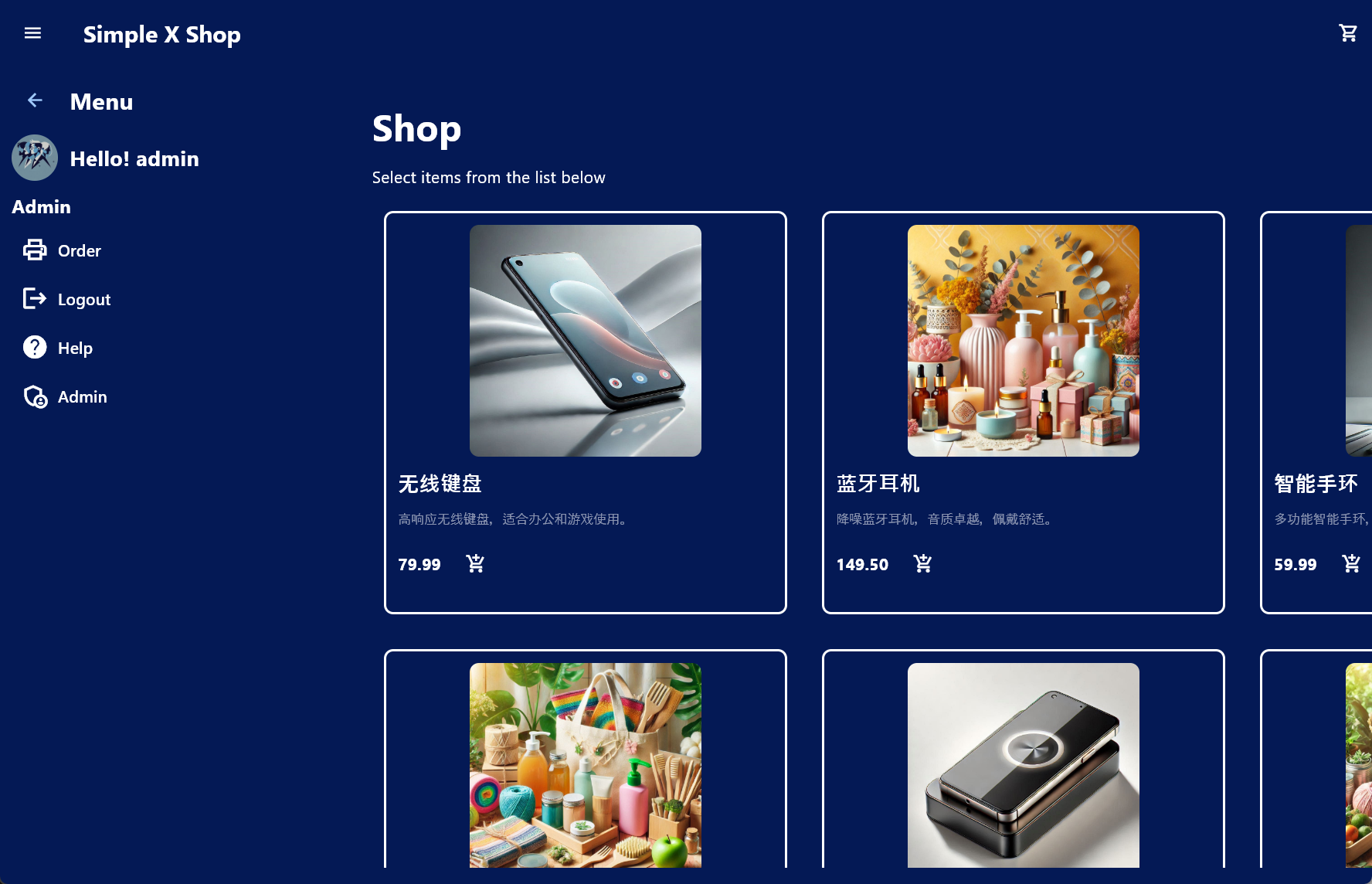
### Post-Refund Processing Interface

* **Description**: After an administrator processes a user's refund request, the user can view the administrator's reason for processing.



### Administrator Login Interface

* **Action**: When an administrator logs in.
  + **Result**: The "Admin" option appears at the bottom of the login panel, allowing the administrator to access the management page.



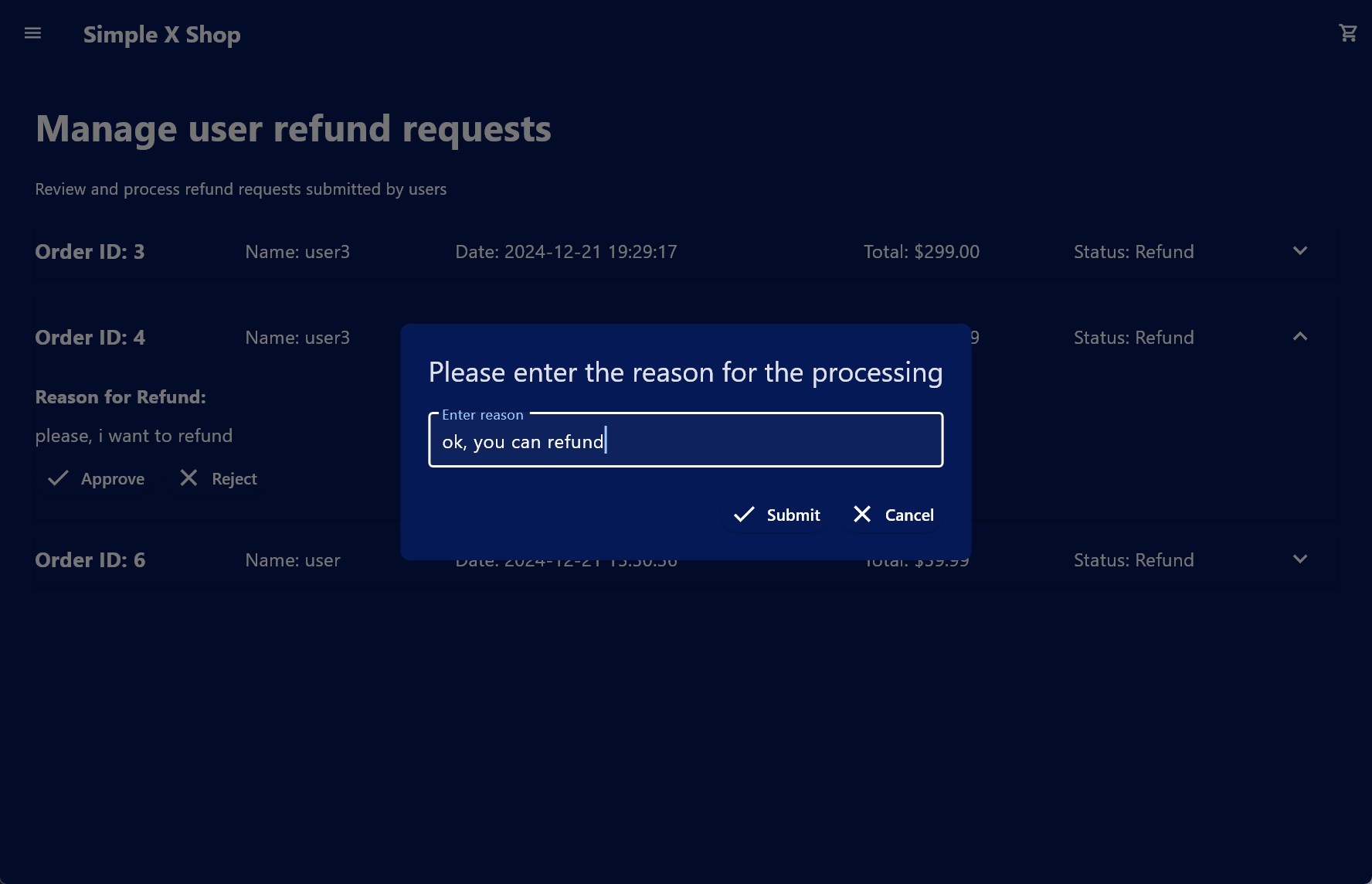
### Management Page

* **Description**: Administrators can view and manage users' refund requests here.



### Refund Processing Interface

* **Action**: When an administrator processes a refund request.
  + **Requirement**: Administrators need to provide a reason for processing.



## f. Code Introduction

The code spans thousands of lines and is not detailed here. Please refer to the source code files for detailed comments.

## 5. Experimental Conclusions and Insights

### 5.1. Experiment Conclusions

Through this experiment, successfully designed and implemented an e-commerce order and refund management system based on the Flet framework. The system offers the following advantages:

1. **Comprehensive Functionality**: Covers core features such as user order management, refund request submission, and refund request processing by administrators, meeting the needs of real e-commerce platforms.
2. **Data Consistency**: Ensures data consistency and integrity through proper database design, stored procedures, and triggers.
3. **User Experience**: Designed an intuitive and user-friendly interface, enhancing the operational experience for both users and administrators.
4. **Maintainability**: Modular code with clear comments and standardized database scripts facilitate future maintenance and expansion.

### 5.2. Personal Insights

1. **Enhanced Comprehensive Application Skills**:
   * Gained a deep understanding of the importance of front-end and back-end collaboration.
   * Mastered the full development process from database design to front-end development.
2. **Importance of Database Design**:
   * Good database design not only improves query performance but also provides a solid foundation for implementing business logic.
   * Proper indexing significantly enhances the system's performance, especially under high-load scenarios.
3. **Criticality of User Experience**:
   * An intuitive and user-friendly interface can significantly enhance system usability and reduce the complexity of user operations.
   * Implementing responsive design principles ensures the application is accessible and functional across various devices and screen sizes.
4. **Permission Control Complexity**:
   * Implementing both front-end and back-end permission controls is essential to prevent unauthorized access.
   * Ensuring that role-based access control is consistently enforced across all layers of the application is crucial for system security.
5. **Continuous Testing and Optimization**:
   * Regular functional, permission, and performance testing are vital to identify and resolve issues promptly.
   * Optimizing database queries and frontend interactions can lead to substantial improvements in system responsiveness and user satisfaction.
6. **Documentation and Code Maintainability**:
   * Maintaining comprehensive documentation and well-commented code enhances team collaboration and eases future maintenance efforts.
   * Adhering to best coding practices ensures that the codebase remains clean, organized, and scalable.

### 5.3. Future Improvement Directions

1. **Feature Expansion**:
   * **Pagination and Search**: Implement pagination and search functionalities for order and refund request lists to improve data navigation efficiency.
   * **Notification System**: Develop an email or SMS notification system to inform users and administrators about order statuses and refund request outcomes.
2. **Enhanced Security**:
   * **Data Encryption**: Encrypt sensitive data such as user emails and other personal information to enhance data security.
   * **Audit Logging**: Implement comprehensive audit logs to track all critical operations, enhancing system traceability and security.
3. **Performance Optimization**:
   * **Caching Mechanism**: Introduce caching strategies to reduce database query load and improve system response times.
   * **Load Balancing**: Deploy load balancing solutions to ensure high availability and reliability under heavy concurrent usage.
4. **User Interface Improvements**:
   * **Responsive Design**: Further optimize the interface layout to ensure excellent display and usability across different devices and screen sizes.
   * **User Feedback Integration**: Collect and incorporate feedback from users and administrators to continuously refine the system's usability and functionality.
5. **Comprehensive Testing Coverage**:
   * **Automated Testing**: Integrate automated testing tools to cover more extensive functional and regression testing, ensuring system stability.
6. **Internationalization and Localization**:
   * **Multi-language Support**: Add multi-language support to cater to a diverse user base.
   * **Localization Settings**: Adjust the system according to regional laws, regulations, and business requirements for better market adaptability.

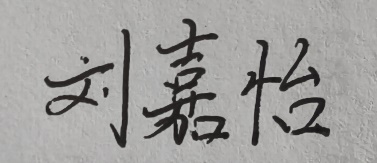
**二、诚信承诺：**

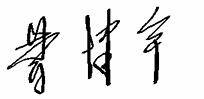
**本人郑重承诺在该课程论文完成的过程中不发生任何不诚信现象，一切不诚信所导致的后果均由本人承担。**

**Integrity Commitment:**

**I/We solemnly promise that I/we have not plagiarized during the completion of this course paper, and all the consequences caused by plagiarism will be borne by me/us.**

**签名拍照附件:**





**Name and Digital Signature of all students in the group**

**Date:**