APU Kiosk System

Object Oriented Methods with UML

Mini Project

Julien CHAN PENG
Yael CHABLOZ
Adrien ASSOUAD

L3

2024

Table of Contents

Introduction

This project was carried out as part of our academic curriculum at EFREI Paris, where we are pursuing our engineering degree in computer science. Currently in our third year, we are completing our first semester abroad at the Asia Pacific University of Technology (APU) in Malaysia. This international experience allows us to deepen our technical skills while discovering new educational and cultural approaches.

The project is part of the course Object Oriented Methods with UML. The objective is to apply the theoretical concepts covered during the semester, such as object-oriented programming principles, UML diagrams, and design patterns, to a practical problem. Working in a team of three students, we collaborated to design and develop a functional system that meets the requirements described in the assignment, while adhering to best practices in software design.

Through this project, we were able to leverage our knowledge of UML to model and design an extensible and maintainable system.

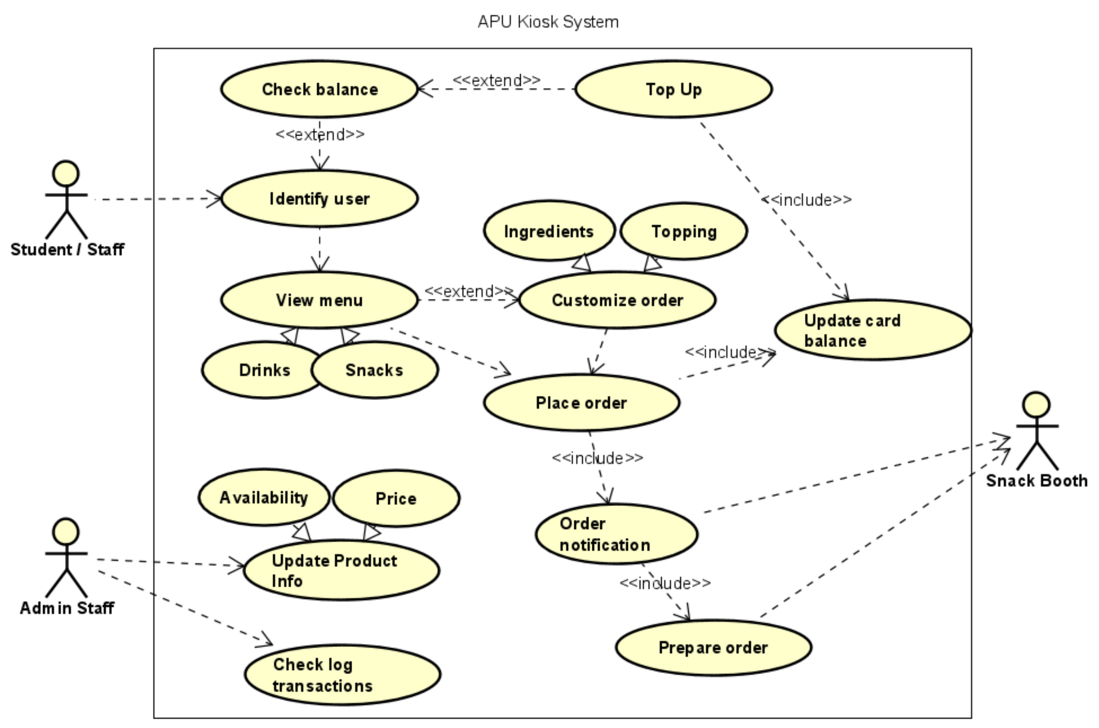
Overview of the scenario

Asia Pacific University aims to improve convenience for students and staff by introducing a snack and drink ordering kiosk system. The kiosks will streamline order management, reducing queues and providing an efficient way for users to browse, customize, and pay for snacks and drinks. Key features include:

- User Identification: Using AP cards for payment and user verification.
- Order Customization: Options for drinks (e.g., sugar level, size, toppings) and snacks (e.g., sandwich ingredients).
- Payment: Cashless deduction from the user's card balance.
- Central Management: Admin staff at the snack booth can update prices and product availability, reflecting changes across all kiosks in real time.

This system adheres to object-oriented principles for extensibility, maintainability, and scalability, ensuring it supports future requirements like additional kiosks or menu expansions.

Use Case Diagram



Description

The updated use case diagram illustrates the interactions between the following actors and elements of the system.

Actors and Use Cases

Student/Staff:

- **Identify User**: Users swipe their AP card to log in to the kiosk system.
    - *Check Balance*: Extends the identification process by displaying the card balance.
    - *Top Up*: Allows users to recharge their card.
- **View Menu**: Users can browse the menu, divided into **Drinks** and **Snacks** categories.
- **Customize Order**: Users can personalize their drinks or snacks by selecting **Ingredients** or **Toppings**.
- **Place Order**: Includes the following actions:
    - Deduction of balance (*Update Card Balance*).
    - Notifications sent to the central system (*Order Notification*).
    - Processing of the order for preparation (*Prepare Order*).

Admin Staff:

- **Update Product Info**: Admin staff can modify product **Price** and **Availability** across all kiosks.
- **Check Log Transactions**: Admin staff can monitor and log all kiosk transactions for auditing purposes.

Snack Booth System:

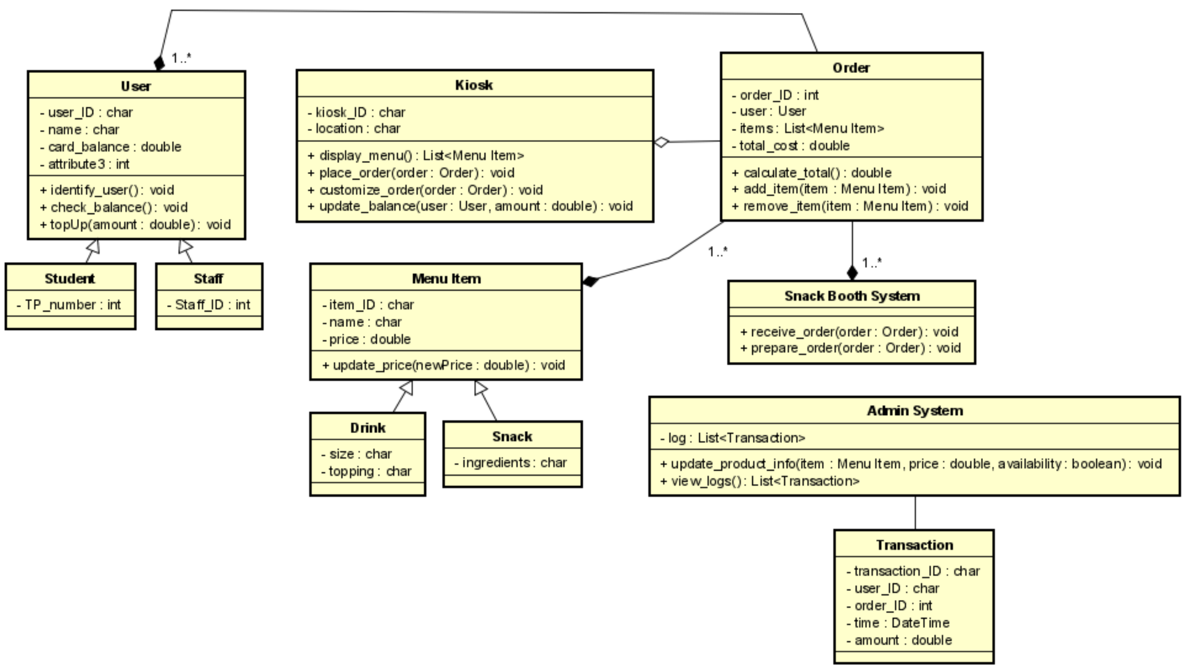
- **Order Notification**: Receives details of the order (item, user, customizations).
- **Prepare Order**: Processes the order in the snack booth for completion.

Structure and Flow

- The **Student/Staff** actor interacts with the kiosk to perform functions such as placing an order, checking balances, or customizing products.
- The **Admin Staff** manages the system centrally, updating information and monitoring transactions.
- The **Snack Booth System** ensures orders are received and processed centrally, demonstrating the connection between kiosks and the preparation area. The *Snack Booth* as a central system ensures clarity about where orders are processed.
- Explicit inclusion of *Update Card Balance* under *Place Order* highlights the integration of the cashless payment system.

This refined diagram accurately represents the interactions and flow in the kiosk system while ensuring modularity, extensibility, and maintainability.

Class Diagram


User
- user_ID : char
- name : char
- card_balance : double
- attribute3 : int
+ identify_user(): void
+ check_balance(): void
+ topUp(amount : double): void
Student
- TP_number : int
Staff
- Staff_ID : int
Kiosk
- kiosk_ID : char
- location : char
+ display_menu() : List<Menu Item>
+ place_order(order : Order): void
+ customize_order(order : Order) : void
+ update_balance(user : User, amount : double) : void
Order
- order_ID : int
- user : User
- items : List<Menu Item>
- total_cost : double
+ calculate_total() : double
+ add_item(item : Menu Item) : void
+ remove_item(item : Menu Item) : void
Menu Item
- item_ID : char
- name : char
- price : double
+ update_price(newPrice : double) : void
Drink
- size : char
- topping : char
Snack
- ingredients : char
Snack Booth System
+ receive_order(order : Order): void
+ prepare_order(order : Order) : void
Admin System
- log : List<Transaction>
+ update_product_info(item : Menu Item, price : double, availability : boolean): void
+ view_logs(): List<Transaction>
Transaction
- transaction_ID : char
- user_ID : char
- order_ID : int
- time : DateTime
- amount : double
1..*
1..*
1..*

Description

The class diagram outlines the structure of the APU Kiosk System and its main components, focusing on their attributes, methods, and interactions:

User Class:

- Represents general users (students and staff).
- Handles user identification, balance checking, and top-up functionality.
- Specialized into Student and Staff.

Kiosk Class:

- Manages user interaction, including displaying the menu, placing orders, and deducting user balance.

MenuItem, Drink, and Snack:

- MenuItem is the base class for all menu items.
- Drink allows customizations (e.g., toppings, size), while Snack tracks ingredients.

Order Class:

- Represents an individual order, containing a list of items and total cost calculation.

AdminSystem Class:

- Enables staff to update product information and view transaction logs.

Transaction Class:

- Records details of completed transactions, such as user ID, order ID, timestamp, and amount.

SnackBoothSystem Class:

- Handles backend processes, including order notifications and preparation.

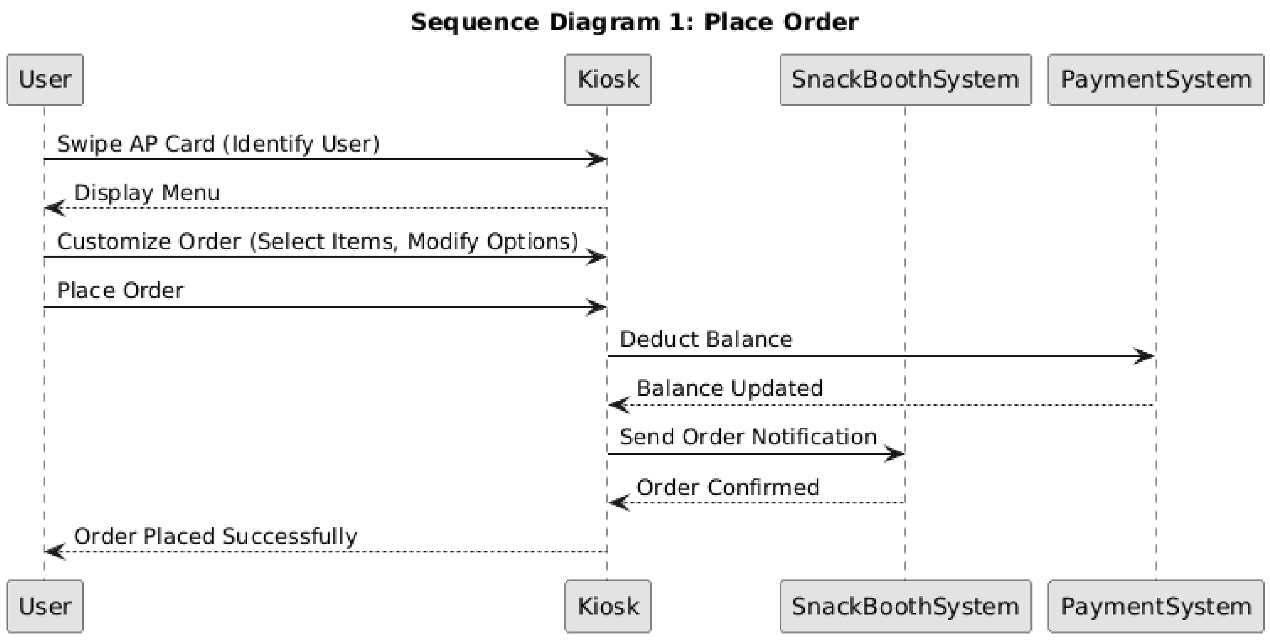
Key Relationships:

- Inheritance: Student and Staff inherit from User; Drink and Snack inherit from MenuItem.
- Associations: Users place Orders; Orders contain multiple MenuItems.
- Aggregation: Kiosk aggregates orders; AdminSystem aggregates Transaction logs.

This design ensures modularity, reusability, and future extensibility of the system.

Sequence Diagrams

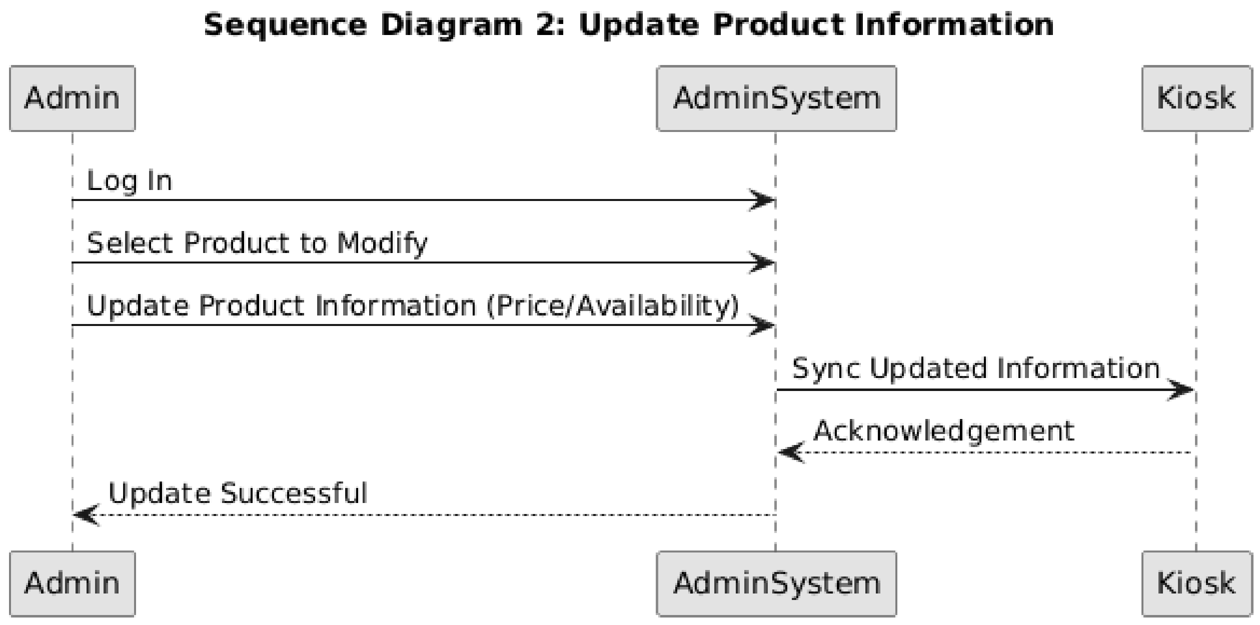
Sequence Diagram 1 : Place Order



This diagram represents the process of a user placing an order:

1. The user swipes their AP card to identify themselves.
2. The kiosk displays the menu, allowing the user to customize their order.
3. Once the order is placed, the kiosk communicates with the payment system to deduct the balance.
4. After payment is confirmed, the order notification is sent to the snack booth system.
5. The user receives confirmation that their order has been successfully placed.

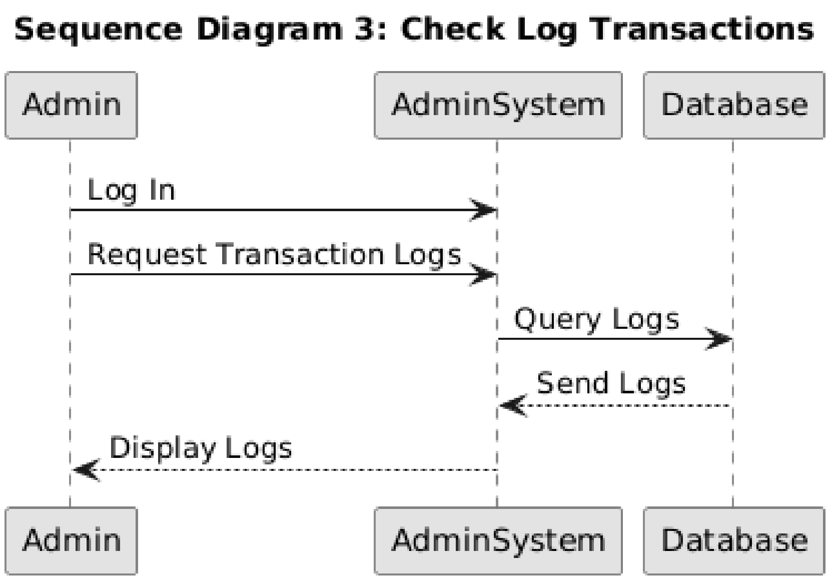
Sequence Diagram 2 : Update Product Information



This diagram showcases how an admin updates product information:

1. The admin logs into the admin system.
2. The admin selects a product and updates its price or availability.
3. The updated information is synced with all kiosks to ensure consistency.
4. The admin system acknowledges the update and confirms its success to the admin.

Sequence Diagram 3 : Check Log Transactions



This diagram illustrates the process of retrieving transaction logs:

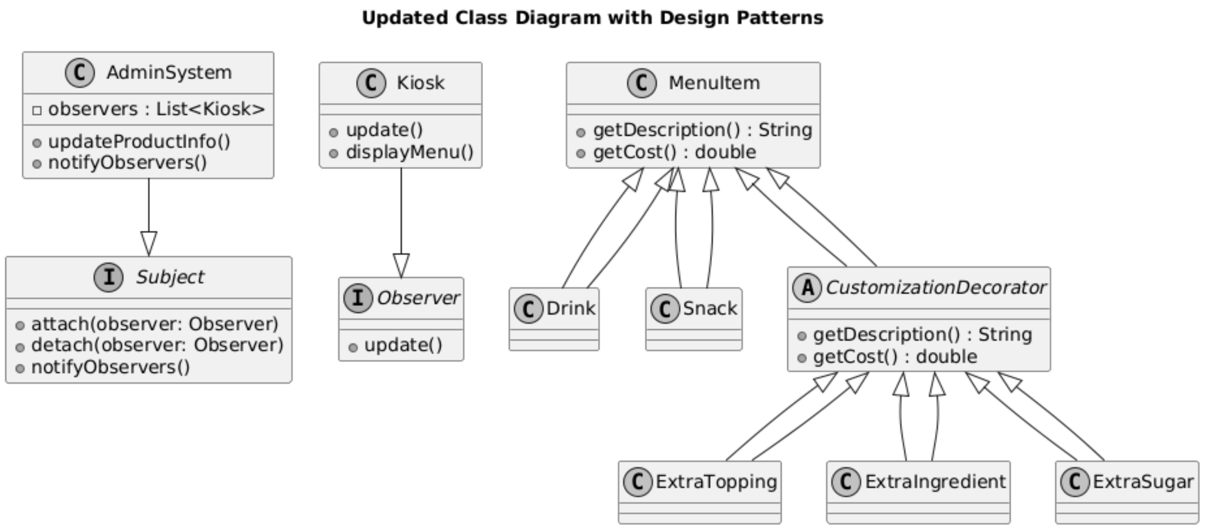
1. The admin logs into the admin system.
2. The admin requests transaction logs for review.
3. The admin system queries the database for the required logs.
4. The database sends the logs back to the admin system, which displays them for the admin.

Class Diagram with design patterns

To enhance the system's extensibility and maintainability, we will integrate two common design patterns into the **APU Kiosk System**:

- **Observer Pattern** - To ensure that all kiosks receive updates when product information (e.g., price, availability) is changed.
- **Decorator Pattern** - To support customization of drinks and snacks (e.g., adding ingredients, toppings).

Here is a brief overview of how the design patterns are incorporated:



Observer Pattern

- **Subject**: AdminSystem
  - Maintains a list of observers (kiosks) and notifies them of any changes.
- **Observer**: Kiosk
  - Updates its product menu and prices when notified by the AdminSystem.

Decorator Pattern

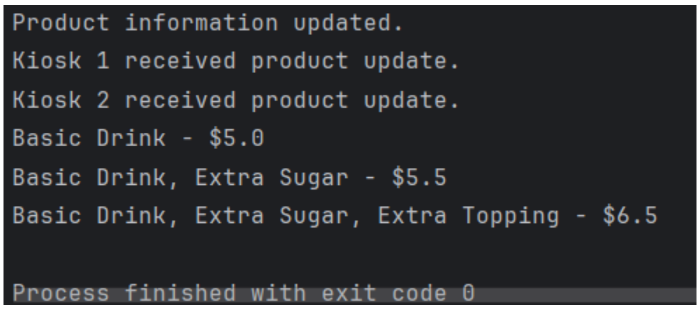
- **Component**: MenuItem (base class for drinks and snacks).
- **ConcreteComponent**: Drink, Snack.
- **Decorator**: CustomizationDecorator (abstract class for customizations).
- **ConcreteDecorator**: ExtraTopping, ExtraIngredient, ExtraSugar (concrete customizations).

Implement design patterns in Java

We decided to implement our design pattern in Java because we have not learned C++ programming. Also, we have stutied Java this semester so it was easier for us to use Java programming.

You can find the Java code for the implementatinon of our design patterns in the folder of this project (file name : KioskSystem.java).

Here is the output of the running code.



```
Product information updated.
Kiosk 1 received product update.
Kiosk 2 received product update.
Basic Drink - $5.0
Basic Drink, Extra Sugar - $5.5
Basic Drink, Extra Sugar, Extra Topping - $6.5

Process finished with exit code 0
```

Concrete applications in the project

- Kiosk Update: When the administrator updates a product (price change or availability), all connected kiosks receive the notification and adjust their displayed data accordingly.

- Order Customization: Users can customize their drinks and snacks (e.g. add sugar, remove ice cubes) at the time of ordering. These options are added dynamically and properly leveraged in the final price.

Evaluation of the solution and choice of patterns

Evaluation of Solution and Choice of Patterns

Overview of the Solution

The APU Kiosk System was designed and implemented to adhere to object-oriented principles such as modularity, encapsulation, and reusability. The system addresses core requirements, including product updates, user interactions, and order customization. By integrating design patterns, the solution is not only functional but also extensible and maintainable, meeting both current and future needs of the business.

Choice of Patterns

1. **Observer Pattern**

   - **Rationale**: The Observer Pattern was chosen to handle real-time updates of product information across all kiosks. This ensures consistency in product prices and availability while centralizing control in the admin system.

- **Advantages**:
    - Decouples the AdminSystem from the individual Kiosk instances.
    - Simplifies the process of adding or removing kiosks without modifying the core functionality.
    - Ensures all kiosks remain synchronized with minimal latency.
- **Suitability**: The Observer Pattern is highly suitable for systems requiring real-time updates and dynamic subscriber management. It adheres to the "open/closed" principle by allowing new observers to be added without changing existing code.

2. **Decorator Pattern**
    - **Rationale**: The Decorator Pattern was implemented to manage the customization of menu items, such as adding extra sugar, toppings, or ingredients to drinks and snacks.
    - **Advantages**:
        - Provides a flexible way to add new customizations without modifying existing classes.
        - Encourages code reuse by composing behavior dynamically rather than using extensive inheritance hierarchies.
        - Simplifies the management of multiple combinations of customizations.
    - **Suitability**: This pattern is ideal for scenarios where objects need additional behaviors dynamically. It avoids the complexity of a rigid inheritance structure while maintaining the modularity of the system.

Analysis of the Solution

1. **Strengths**:
    - The system leverages design patterns to ensure scalability and maintainability.
    - Real-time synchronization across kiosks enhances user experience and administrative control.
    - Customization features align with user expectations for flexible menu options.
2. **Limitations**:
    - The Observer Pattern may lead to increased memory usage if the number of kiosks scales significantly, requiring careful resource management.
    - The Decorator Pattern could become complex if there are too many layers of customization, potentially impacting performance.

3. **Future Enhancements**:
    - Implement caching mechanisms to optimize performance during frequent updates.
    - Introduce a Command Pattern to handle undo/redo operations for admin actions.
    - Enhance user authentication using an additional security layer like biometrics or a PIN system.

The solution effectively uses the Observer and Decorator Patterns to meet the system's requirements while ensuring extensibility and adherence to object-oriented principles. These patterns provide a robust foundation for future expansion, such as integrating new kiosks or introducing additional customizations, making the system both practical and forward-compatible.

Critical appraisal report

The APU Kiosk System project aimed to design and implement a robust, object-oriented solution for a campus-wide snack and drink ordering system. The focus was on addressing user needs, ensuring system extensibility, and maintaining administrative control through good software design practices. This project successfully demonstrated a well-thought-out object-oriented design that adheres to industry best practices. While the implementation met key requirements, there is room for refinement and expansion to ensure optimal performance, enhanced user experience, and robust scalability. The project provided valuable insights into designing extensible systems and highlighted areas for growth in both technical and collaborative aspects.