CS396 Software Design Principles and Practices

Project: Milestone 1 Report

People

Xuyang Li, Irving Peng, Jenny Zhou

Option

Option 1

Link to Project GitHub Repository

(Instruction: Please provide the HTTP URL of the repository.)

https://github.com/nu-cs-sw-design/project-20252601-396_project

Project Planning

(Instruction: Please break down the activities needed to complete this project and a timeline.)

Week1 (11/10- 11/16):

Items to Complete:

- Github Set Up
- Project Planning
- Complete overall project planning.
- Produce initial analysis of use cases UC1–UC15 and derive major system components.
- Draft system architecture (based on MVC and three layers architecture).
- Produce a frontend/ backend planning based on user requirements
- Set up multiple roles in our projects as the main users/ classes

MileStone 1: 11/16

- Project Planning and Initial Analysis Report

Week 2 (11/17-11/23):

Objectives: Begin development; build prototype UI and backend skeleton.

Items to Complete

Frontend:

- Design and build prototype screens for:
    - Welcome screen(UC1)
    - Main Menu (UC1-UC2)
    - Category browsing (UC2)
    - Item details + customization view (UC3)
    - Order summary panel (UC4–UC5)

Backend:

- Set up project skeleton (routing, controllers, database connections).
- Implement early service stubs for Orders, Menu, Payments.
- Implement basic data models (MenuItem, Order, OrderItem, Payment).

Use cases to complete :

- UC1- UC7
- Project Frontend Prototype

MileStone 2: 11/23

- Project Progress Report

Week 3 (11/24 - 11/30)**:**

Objectives: Complete remaining use cases, integrate backend logic, and finalize core functionality.

Frontend:

- Screens for counter payment flow, kitchen queue view, manager dashboard, menu management, and reporting.
- Polish navigation flow and validation logic.

Backend:

- Implement business logic for payment-at-counter workflow (UC8–UC9).
- Implement kitchen order management and status updates (UC10–UC12).
- Implement menu management CRUD operations for manager (UC13–UC14).
- Implement reports generator (daily sales summary) for UC15.

Use Cases to Complete

- UC8-UC15

Architectural Expectation:

- Workable Frontend
- Runnable Backend system

Progress report: 11/23

Final Week (11/31 - 12/8):
Objectives: Finalize, polish, test, debug, and prepare deliverables.

Items to complete

- Full system integration testing (front/back/database).
- User testing with sample scenarios.
- Fix bugs and make adjustments for usability.
- Performance checks (menu load times, payment flow responsiveness).
- Prepare deployment instructions.

Milestone 3: 12/8 (Final Deliverable)

- Software Requirement Document
- Software Design document
    - a functional and flexible design of the product
    - an analysis of how the design complies with the design principles
    - A list of at least 3 anticipated software requirement changes
- Software Requirements Document (expanded with all 15 use cases).
- Functional and flexible final implementation of fast food ordering system
- Analysis of design compliance with design principles (MVC, Three Layers)

Initial Analysis

(Instructions:

For Option 1: Design Your Own Product

Please write down the purpose of the product, a brief description of the product features, and the target users. Also, write the initial list of use cases as much as you can. You may change or complete the use cases later if you see fit.

- **Purpose of the Product:** The purpose of our Fast Food Ordering System is to streamline the ordering process and fulfillment process in a fast-food restaurant. The system aims to reduce waiting time for customers, minimize order mistakes, and provide the restaurant employees with a clear, efficient workflow for handling high volumes of orders.
- **Brief Description of the Product Features:** The Fast Food Ordering System supports the complete end-to-end flow of placing and fulfilling orders in a fast-food restaurant. Customers interact directly with a self-service interface to browse the menu, customize items, place orders, and complete payments without assistance from restaurant staff. After an order is submitted, the system generates an order number for the customer to pick up. Cashiers can assist customers when needed by creating, modifying, or view order status, handling special payment situations, or managing refunds. Kitchen staff use the system to view orders queue, update order status. Managers maintain the menu, adjust pricing, and view basic sales summaries.
- **Target Users:** Customers who want to order foods, and restaurant employees(Cashier, Kitchen staff, Manager).

Use Cases:

- **UC1: Start the System**
  - **Actor:** Customer
  - **Precondition:** The system is powered on and the system is connected to the ordering backend.
  - **Basic Flow:**
    1. The customer taps the screen to start the system.
    2. The system displays a welcome message.
    3. The system displays the Main Menu screen with the “Start Order” button.
- **UC2: Browse Menu**
  - **Actor:** Customer

  - **Precondition:** The system is on the Main Menu screen.
  - **Basic Flow:**
    1. The customer selects “Start Order”.
    2. The system displays the menu organized by categories (e.g., Burgers, Sides, Drinks).
    3. The customer selects a category.
    4. The system displays the list of items in that category, including name, price, and a item image.
- **UC3: View Item Details and Customize Item**
  - **Actor:** Customer
  - **Precondition:** The menu for a specific category is displayed.
  - **Basic Flow:**
    1. The customer selects an item from the list.
    2. The system displays the item details, including description, base price, and available customization options.
    3. The customer chooses customization options (e.g., size, add/remove toppings, combo choice).
    4. The system updates the item price based on the selected options.
- **UC4: Add Item to Order**
  - **Actor:** Customer
  - **Precondition:** The item detail screen is displayed with selected customization options.
  - **Basic Flow:**
    1. The customer sets the desired quantity of the item.
    2. The customer selects “Add to Order”.
    3. The system adds the item to the current order.
    4. The system displays the updated order summary with subtotal.
- **UC5: Review and Edit Current Order**
  - **Actor:** Customer
  - **Precondition:** The customer has at least one item in the current order.
  - **Basic Flow:**
    1. The customer selects “Review Order”.
    2. The system displays the list of items in the order, including quantity and price.
    3. The customer selects an item to edit.
    4. The system allows the customer to change the quantity or remove the item.
    5. The system updates the order summary after the edits.

- **UC6: Confirm Order and Proceed to Payment**
  - **Actor:** Customer
  - **Precondition:** The order summary is displayed with at least one item.
  - **Basic Flow:**
    1. The customer reviews the items and total amount.
    2. The customer selects “Proceed to Payment”.
    3. The system calculates taxes and displays the final total.
    4. The system prompts the customer to choose a payment method, such as:
       a. “Pay at System (Cards)”
       b. “Pay at Counter (Cash or Card)”.
    5. The customer selects a payment method.
    6. The system branches to the corresponding payment use case.
- **UC7: Complete Payment at System**
  - **Actor:** Customer
  - **Precondition:**
    - The system is displaying the payment screen with the final total.
    - The customer has selected “Pay at System (Cards)” as the payment method.
  - **Basic Flow:**
    1. The customer follows the on-screen instructions to insert, tap, or swipe a payment card.
    2. The system sends the payment request to the payment processor.
    3. The system receives confirmation that the payment is approved.
    4. The system records the payment and finalizes the order as “Paid”.
    5. The system displays a confirmation screen with the order number.
    6. The system sends the order to the kitchen queue with status “Pending”.
- **UC8: Complete Payment at Counter**
  - **Actor:** Customer
  - **Precondition:**
    - The system is displaying the payment screen with the final total.
    - The customer has selected “Pay at Counter (Cash or Card)” on the system.
  - **Basic Flow:**
    1. The system records the payment and finalizes the order as “Pending Payment at Counter”.
    2. The system displays a confirmation screen with the order number.
- **UC9: Complete Payment at Counter**
  - **Actor:** Cashier
  - **Precondition:**

        - The customer has selected “Pay at Counter (Cash or Card)” on the system.
        - The system has created an order with status “Pending Payment at Counter”.
        - The customer receives the order number.
        - **Basic Flow:**
            1. The customer approaches the counter and provides the order number to the cashier.
            2. The cashier searches for the order by order number in the system.
            3. The system displays the order details and the total amount due.
            4. The cashier receives payment from the customer (cash or card).
            5. The system records the payment and updates the order status to “Paid”.
            6. The system sends the order to the kitchen queue with status “Pending”.
            7. The system optionally prints or displays a receipt for the customer.

- **UC10: View Pending Orders in Kitchen Queue**
    - **Actor:** Kitchen Staff
    - **Precondition:** At least one order has been submitted and paid.
    - **Basic Flow:**
        1. The kitchen staff logs in or opens the kitchen display screen.
        2. The system displays the list of pending orders sorted by creation time.
        3. The staff selects an order from the list to view details.
- **UC11: Update Order to “In Preparation”**
    - **Actor:** Kitchen Staff
    - **Precondition:** The kitchen staff is viewing the details of a pending order.
    - **Basic Flow:**
        1. The kitchen staff starts preparing the items in the order.
        2. The staff selects the option to mark the order as “In Preparation”.
        3. The system updates the order status to “In Preparation”.
        4. The order status screen at the pickup counter must display the order to be “In Preparation”.
- **UC12: Mark Order as Ready for Pickup**
    - **Actor:** Kitchen Staff
    - **Precondition:** The items in the order have been fully prepared.
    - **Basic Flow:**
        1. The kitchen staff verifies that the order is complete.
        2. The staff selects the option to mark the order as “Ready”.
        3. The system updates the order status to “Ready”.

    4. The order status screen at the pickup counter must display the order to be "Ready."

- **UC13: Add New Menu Item**
    - **Actor:** Manager
    - **Precondition:** The manager is authenticated in the management interface.
    - **Basic Flow:**
        1. The manager selects "Menu Management" from the management dashboard.
        2. The system displays the current list of menu items.
        3. The manager selects "Add New Item".
        4. The system prompts the manager to enter the item name, price, category, and optional description.
        5. The manager confirms the creation of the new item.
        6. The system saves the new item and updates the menu.
- **UC14: Update Existing Menu Item**
    - **Actor:** Manager
    - **Precondition:** At least one menu item exists in the system; the manager is authenticated.
    - **Basic Flow:**
        1. The manager opens the "Menu Management" screen.
        2. The system displays the list of menu items.
        3. The manager selects an item to update.
        4. The system displays the current details of the selected item.
        5. The manager edits fields such as price and description.
        6. The manager saves the changes.
        7. The system updates the item in the menu.
- **UC15: View Daily Sales Summary**
    - **Actor:** Manager
    - **Precondition:** The system has recorded orders and payments for at least one day.
    - **Basic Flow:**
        1. The manager selects "Reports" from the management dashboard.
        2. The system displays a list of available report types.
        3. The manager selects "Daily Sales Summary" and chooses a date.
        4. The system calculates total revenue, number of orders, and top-selling items for that date.
        5. The system presents the summary report on the screen.

Software Architectural Design

(Instructions: If you picked Option 1 or 2, please also attach the Software Architectural Design analysis.)

Quality Attributes Scenarios

- **QA1:** The system must support 40 or more concurrent users, including customers and staff, without performance degradation.
- **QA2:** The system must prevent any customer from viewing, editing, or deleting orders belonging to other customers.
- **QA3:** Paid customer orders must appear on the kitchen display screens within 30 seconds of successful payment processing.
- **QA4:** The price or description of any existing menu item must be editable through the management interface without requiring code changes or backend redeployment.
- **QA5:** Updated menu item information, including price or description changes, must propagate to all customer kiosks and staff screens within 10 seconds of submission.
- **QA6:** Customer kiosks must automatically log out and return to the welcome screen after 30 seconds of inactivity to prevent unauthorized actions.
- **QA7:** All menu categories must load on customer kiosks within 2 seconds during peak operating hours.
- **QA8:** Adding a new menu category (e.g., Chicken Nuggets) must be supported through the management interface without requiring code changes or backend deployment.
- **QA9:** Newly added menu categories must propagate to all customer-facing and staff-facing screens within 10 seconds of creation.
- **QA10:** Cashiers and customers must be prevented from accessing manager-level features or configuration screens.
- **QA11:** Customers must not be able to access kitchen staff features or manager features under any circumstances.
- **QA12:** A newly hired kitchen staff member must be able to configure their user account and begin operating the system within 1 hour of training.
- **QA13:** The system must prevent duplicate orders from being created if a customer attempts multiple payment submissions during a timeout or retry scenario.
- **QA14:** Adding a new customer kiosk or ordering device must not require restarting any backend services.
- **QA15:** The system must support up to 50 menu items per category without causing noticeable delays in category or item loading times.
- **QA16:** The manager interface must automatically log out after 2 minutes of inactivity to prevent unauthorized use
- **QA17:** The system must disallow SQL injections in any fillable textbox.
- **QA18:** Order status change by kitchen staff must propagate to the pickup display screen within 30 seconds of change.
- **QA19:** The display (background and theme) of user screens are configurable.

- **QA20:** Menu item appearance is customizable (color, font, highlights)

<u>Ranking of Quality Attributes</u>

High importance (also ordered within the list as well by importance, though not as different relatively):

- QA2: Preventing customers from accessing others' orders is security critical, protecting other customers from having their orders tampered with. This is necessary in keeping the integrity of the restaurant and preventing blame to the restaurant for order mistake misunderstandings.
- QA1: Slow response times will frustrate both customers and staff, and during peak hours, will result in significant bottlenecking and slowdown for staff who will likely already be very busy
- QA3, QA7, QA18: Delays will affect order fulfillment and customer satisfaction.
- QA10, QA11: Prevents unauthorized changes of menu items for those who should not have access, and in turn avoids malevolent action.
- QA13: Avoids financial loss and confusion for the customers.
- QA17: Disallow malevolent action that could result in major loss to the system.

Medium importance:

- QA4, QA8: Menu changes likely come with a variety of updates that are day-wise. Thus, they are usually not immediately critical
- QA5, QA9: Fast propagation will ensure that correct information will be shown to all users, preventing customers from possibly sending orders for items with slightly varied descriptions, but nevertheless this is not completely critical for a restaurant to function.
- QA12: Onboarding is done once per person, thus longer learning times due to complexity can be allowed.
- QA14: Scaling the number of devices that the system is displayed on is not immediately critical, as users can usually handle working around less devices if need be. This can be done in downtime.
- QA16: This will be assisting for managers, but with proper training, this feature is not critical as long as managers have a habit of logging out or putting devices where they cannot be easily accessed.
- QA6: Customer auto-logout will be nice to prevent customers from taking over other customer's orders or seeing what other customers are ordering, but since completion of payment will also close the customer's interaction with their screen, there will not be as much as a true security concern.

Low importance:

- QA19, QA20: Both features are purely cosmetic and do not change the usability of the system.

<u>Architectural Design Patterns</u>

- Three Layered Architecture:
    - The three layered architecture fits the system well, because our system's responsibilities separate into user interfaces, business workflows, and data concerns.
        - Presentation Layer – This layer should include Customer UI, Cashier UI, Kitchen UI, Manager UI. Those interfaces handle user inputs and display information.
            - Rendering screens, menus, kitchen queue, etc.
            - Sending user requests to applications layer.
            - Displaying error messages, confirmation messages, etc.
        - Application Layer – This layer implements the core business workflows and system behavior. It coordinates how the system responds to each user case.
            - Validating user actions.
            - Enforcing rules like order status transitions and manager permissions.
            - Navigates the order lifecycle.
        - Domain Layer – This layer encapsulates the data models, and data access logic.
            - Database access.
            - Persistent data models for orders, order items, menu items, roles, and daily summary records.
            - Maintaining data consistency.
- MVC Architecture:
    - The MVC pattern could also apply to our system. Multiple views in the system need to be updated by some same underlying information. MVC's observer pattern fits well for propagating updates to multiple areas.
        - Model – The Model represents the data that each UI needs to display or update. It is responsible for:
            - Holding local UI state.
            - Reflecting data retrieved from the backend.
            - Notifying views when the data changes.
        - View – The View includes all UI screens for the system. It focus on:
            - Visual Layout

            - Rendering menu, buttons, item details, etc.
            - Displaying status changes and error messages
        - Controller – The Controller handles interactions from users and translates UI actions into system behaviors. Such as:
            - Interprets UI events
            - Updates the Model based on response
- Client-server architecture
    - This pattern can be used as there are multiple devices for this system over the same data. As such, we could possibly use a browser based client and move the rest of the system outside the display into an API.
- Microservice Architecture
    - Microservice architecture could possibly apply, as we have different stages that allow us to cleanly map the lifecycle of a customer's order (generating order, order processors, etc), alongside needs of authentication, payment, traffic loading etc. But, as a one-restaurant system that is not expected to be immensely complex, the overhead of the extra complexity may very well be overkill, and the system could be much easier maintained with a more centralized backend that has unified testing.
- Other design patterns that would work well for a much larger system but would likely be overkill/too complex or unneeded for our system
    - Circuit breaker: Requires sophisticated testing and infrastructure management. This system is not expected to be easily accessible on the internet, thus this will likely be wholly unnecessary.
    - Sharding: Will not have enough data for this to be cost-efficient
    - Throttling; Will not have enough traffic for this to be cost efficient