Phase II – Design Report

Team J – Sri Suvetha Meenaa Subramanian, Sanskriti Khadka, Shreyosee Chowdhury, Damilola Babajide

1. Introduction

- Overall picture of the system
- Collaboration class diagram

2. All Use Cases

- For each use case: Normal scenario and Exceptional scenarios
- For each use case:
    - Sequence or collaboration diagram
    - Petri net diagram

3. ER Diagram

- Full system ER diagram
- Attributes + primary keys

4. Detailed Design

- Pseudocode for every method
- Inputs, outputs, functionality

5. System Screens

- GUI mockups or screenshots
- Explain one prototype functionality

6. Group Memos

- Meeting notes
- Team concerns

7. Git Repository Link

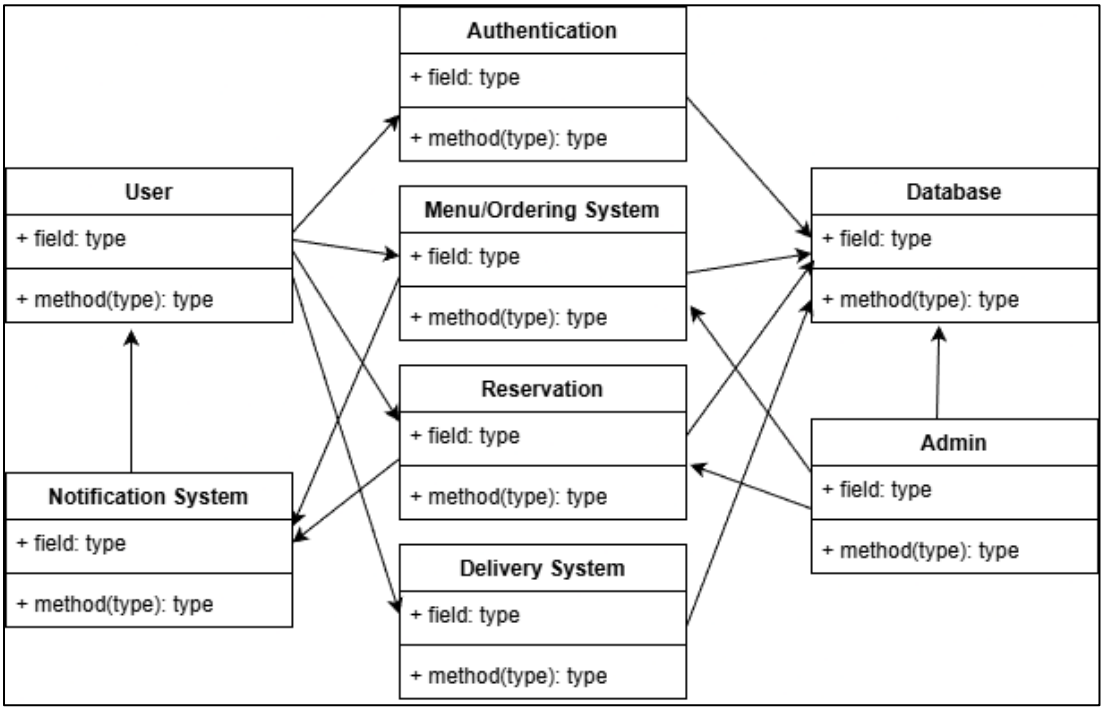
- https://github.com/shreychow/SD_Foods_csc322

1. Introduction

This project designs an AI-enabled online restaurant ordering and delivery system that connects customers, employees, and visitors through an integrated platform. Customers can register, browse menus with images, place orders, deposit money, rate food and delivery quality, participate in discussions, and upgrade to VIP status based on spending and behavior. Employees including chefs, delivery staff, and the manager, handle menu creation, food preparation, delivery, complaint resolution, HR decisions, and system moderation. Visitors may explore the menu, ask questions through an AI-assisted chat box, and apply for registration.

The system incorporates key operational features such as AI-based customer service, a reputation management and warning system, automatic financial checks, competitive delivery bidding, and manager-controlled HR actions including hiring, firing, promotions, and demotions. VIP customers receive discounts, special dishes, and weighted feedback, while misconduct may result in warnings or removal from the system. This design report outlines the system architecture through collaboration class diagrams, detailed use cases, the ER model, method pseudocode, and user-interface mockups, forming a complete blueprint for implementation.

The collaboration class diagram below shows the major system components and how they interact at a high level, showing the flow of information between users, services, and the database.



2. All Use Cases

This section explains the main use cases of our system. For each use case, we describe the normal flow and what happens if something goes wrong (exception cases). We also include one diagram per use case to show how the system behaves. For three of the more complex use cases (placing an order, handling complaints, and payment), we use Petri-net diagrams. For the remaining use cases, we use simple sequence/collaboration diagrams.

UC-01: Register / Login

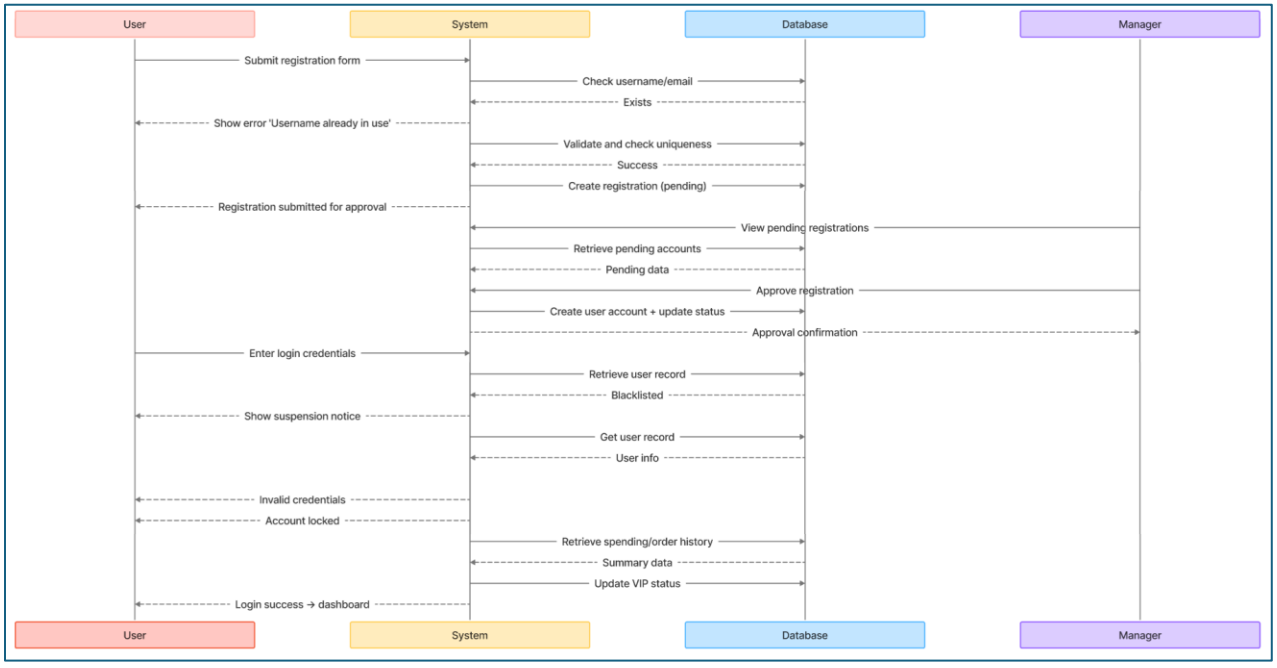
Purpose: This use case handles the creation and approval of user accounts and access authentication.

Normal Scenario

1. Visitor submits registration form (username, password, email, phone)
2. System validates input and creates registration request with status = "pending"
3. Manager reviews and approves registration
4. System creates user account (role = customer, warnings = 0, spending = $0, vip = false)
5. User logs in with credentials; system verifies password
6. System checks VIP eligibility (spending ≥ $100 OR 3 orders without complaints) and upgrades if qualified
7. System redirects to role-based dashboard with personalized content
8. System displays any warnings prominently

Exceptional Scenarios

- E1 - Username Exists: System rejects registration, displays error
- E2 - Weak Password: System requires 8+ chars with letters and numbers
- E3 - Manager Rejects: Registration deleted, applicant notified
- E4 - Wrong Credentials: System shows generic error, locks account after 5 attempts
- E5 - Blacklisted User: System blocks login, displays suspension message



UC-02: Browse Menu and Place Order

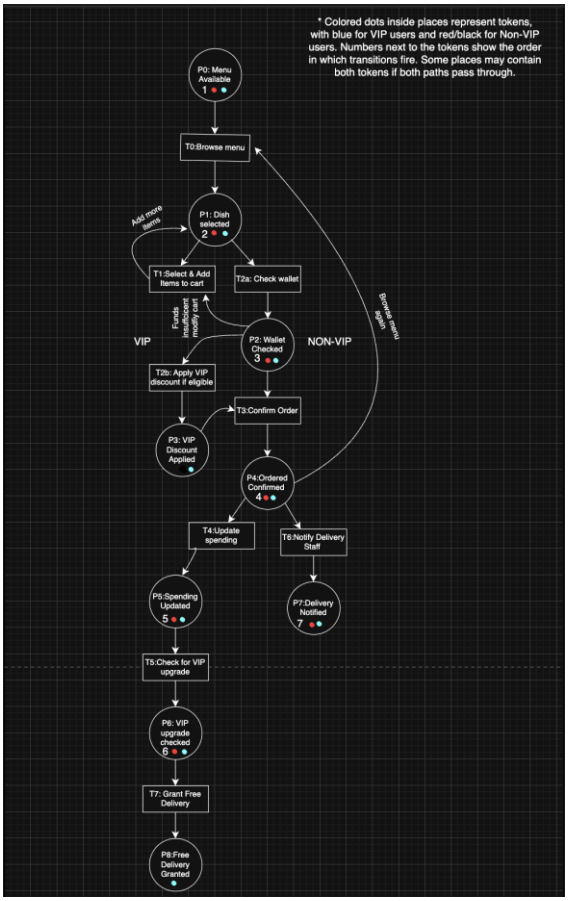
Purpose: This use case allows customers to view available dishes, add items to their cart, and complete orders with automatic VIP discount and delivery assignment.

Normal Scenario

1. Customer views menu with dishes (image, price, rating, chef, VIP badge if exclusive)
2. Customer filters/searches and adds items to cart
3. System validates VIP restriction on exclusive dishes
4. Customer proceeds to checkout
5. System calculates: subtotal, VIP discount (5% if applicable), delivery fee ($5 or free every 3rd order for VIP)
6. System validates wallet balance ≥ total
7. Customer confirms; system deducts payment, creates order, updates spending
8. System checks VIP upgrade criteria and upgrades if met
9. System notifies delivery staff and opens bidding
10. Customer receives order confirmation with tracking

Exceptional Scenarios

- E1 - Insufficient Balance: System rejects order, issues warning, prompts to add funds
- E2 - Item Unavailable: System removes item from cart, recalculates total
- E3 - Non-VIP Orders VIP Dish: System blocks, shows VIP requirements
- E4 - No Delivery Staff: System notifies manager, displays delay message to customer
- E5 - Payment Error: System rolls back transaction, no funds deducted



UC-03: Rate Food and Delivery

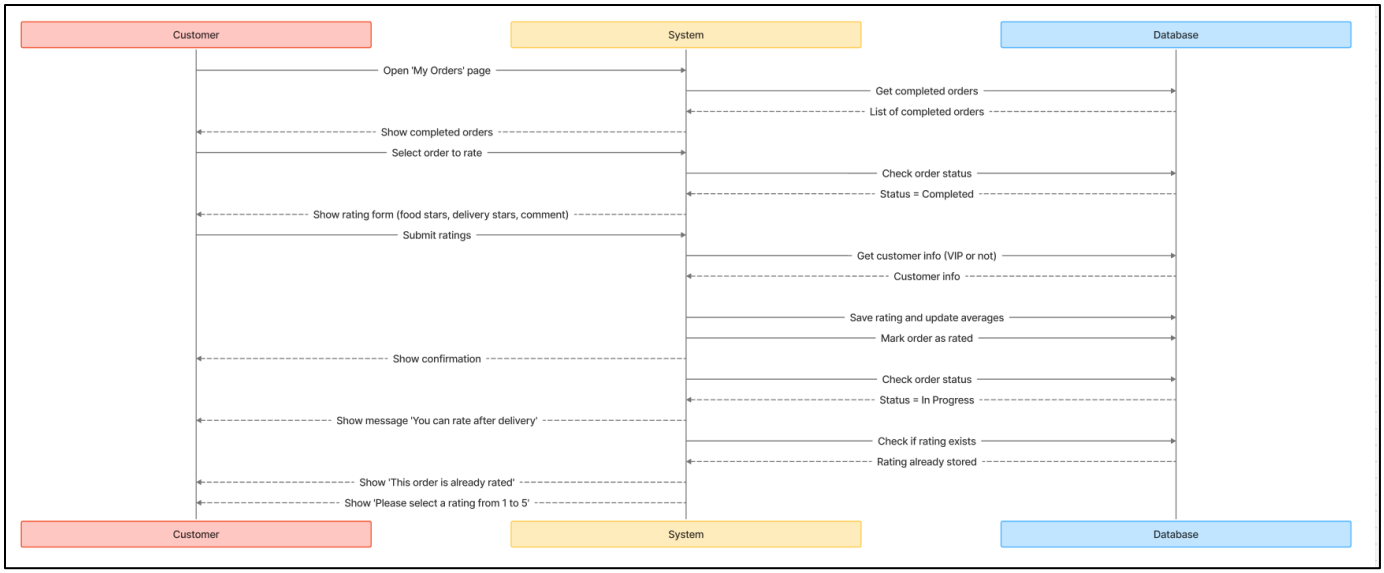
Purpose: This use case enables customers to rate completed orders with separate ratings for food quality and delivery service, with VIP ratings carrying double weight.

Normal Scenario

1. Customer opens completed order from "My Orders"
2. System displays rating form (1-5 stars for food and delivery, optional comment)
3. Customer submits ratings
4. System applies VIP weight (×2 if VIP, ×1 if regular)
5. System updates averages for dish and delivery person
6. System marks order as rated
7. System flags HR actions if needed (average < 3.0 = demotion review, > 4.5 = bonus)

Exceptional Scenarios

- E1 - Rating Twice: System blocks, shows "already rated" message
- E2 - Order Incomplete: System hides rating form until order is completed
- E3 - Invalid Rating: System validates 1-5 range, rejects if outside



UC-04: Handle Complaints and Feedback

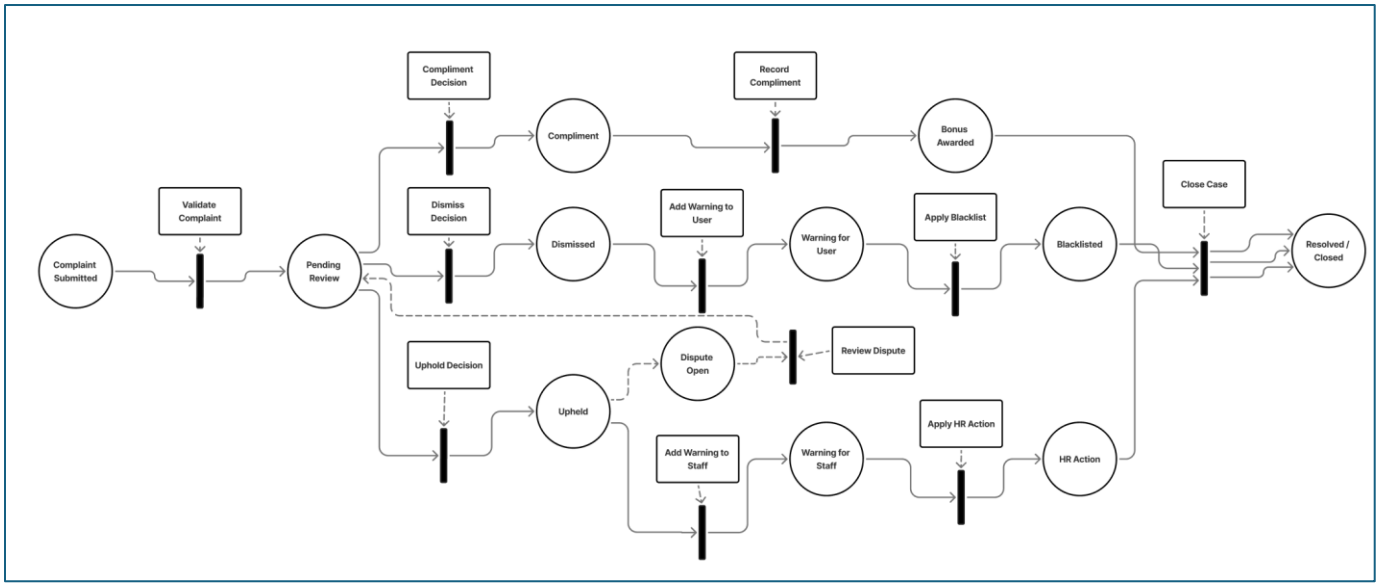
Purpose: This use case manages the submission, review, and resolution of complaints and compliments, including warning systems and HR actions for employees and customers.

Normal Scenario

1. User submits complaint/compliment about chef, delivery person, or order
2. System validates interaction and creates record (status = "pending", weight = 2 if VIP)
3. Manager reviews and decides: Uphold, Dismiss, or Compliment
4. If Upheld: Subject gets warnings (×2 if from VIP). If warnings ≥ 3: employee demoted/terminated, customer blacklisted, VIP downgraded at 2 warnings
5. If Dismissed: Complainant gets 1 warning for false complaint
6. If Compliment: Cancels 1 complaint (1:1 ratio). 3 compliments = bonus review
7. System updates status and notifies parties

Exceptional Scenarios

- E1 - False Complaint: Manager dismisses, complainant receives warning
- E2 - Subject Disputes: System allows one dispute, manager makes final decision
- E3 - VIP Complaint → Blacklist: VIP complaint weight causes immediate 3 warnings
- E4 - Compliment Cancels Complaint: System reduces warnings by 1
- E5 - 3 Warnings: Automatic demotion/termination for employee, blacklist for customer



UC-05: AI Chat Interaction

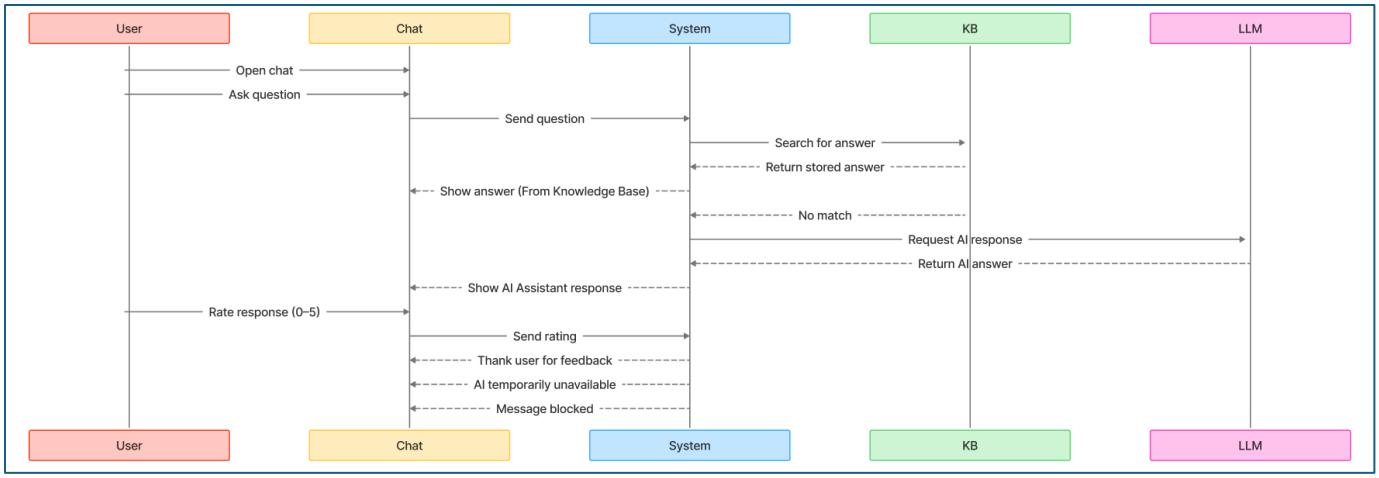
Purpose: This use case provides AI-powered customer support by searching a local knowledge base first, then querying an external LLM if needed, with user rating and quality control.

Normal Scenario

1. User types question in chatbot
2. System searches Knowledge Base (KB) first
3. If found: Returns answer labeled "From Restaurant Knowledge Base"
4. If not found: Queries external LLM (Ollama/Hugging Face), returns answer labeled "AI Assistant"
5. User rates response (0-5 stars)
6. System logs interaction
7. If rated 0: flags for manager review, may remove KB entry and ban author

Exceptional Scenarios

- E1 - AI Unavailable: System displays "temporarily unavailable", saves question for manager
- E2 - Inappropriate Content: System blocks message, issues warning
- E3 - KB Answer Rated 0 Multiple Times: System flags for removal, notifies manager



UC-06: Payment and Finance Management

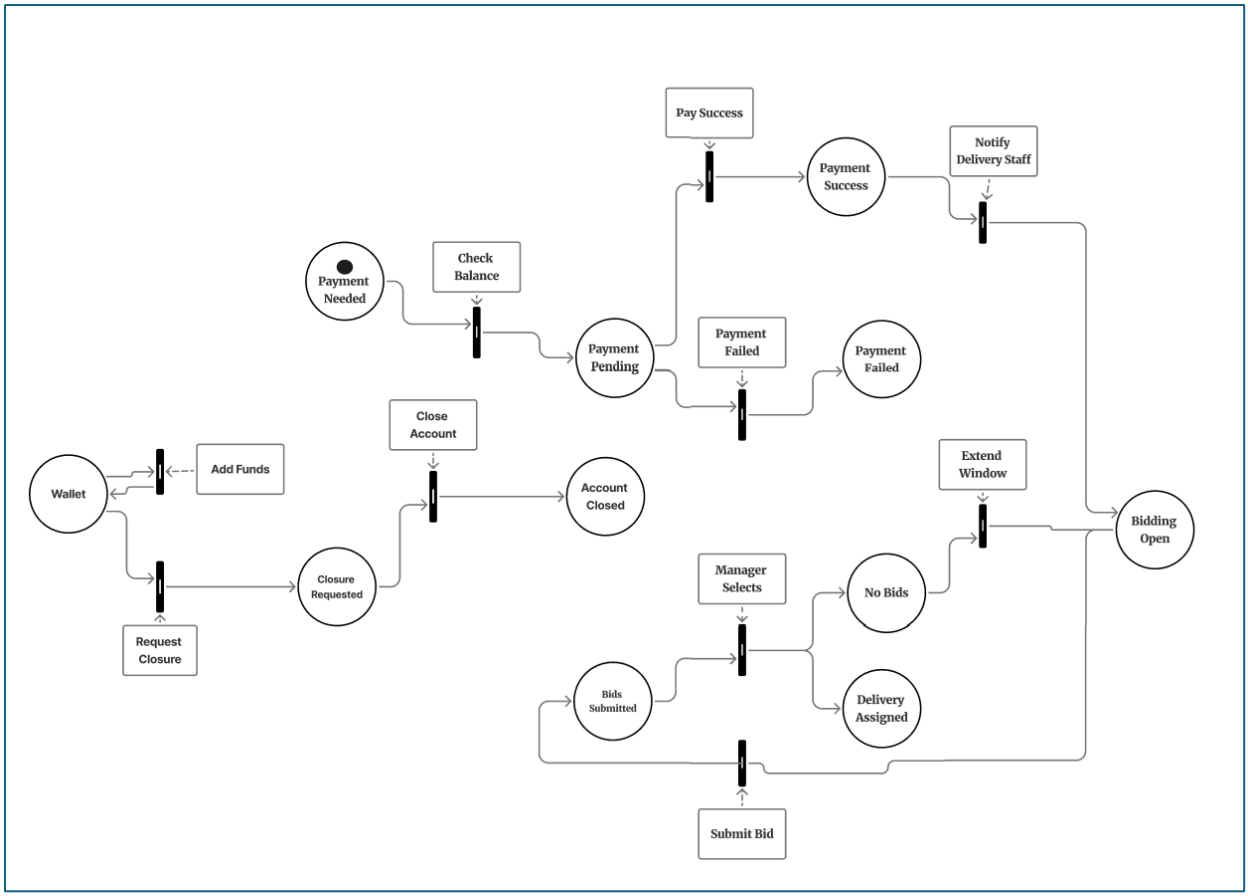
Purpose: This use case handles wallet deposits, order payments, refunds, and the competitive delivery bidding process with manager oversight.

Normal Scenario

1. User navigates to "Wallet" page and clicks "Add Funds"
2. User enters deposit amount and payment method
3. System processes payment and updates wallet balance
4. System logs transaction in transaction history
5. Customer places order; system validates wallet balance ≥ total
6. System deducts amount from wallet and logs order payment transaction
7. Order enters delivery bidding phase
8. Delivery staff view available orders and submit bid amounts
9. Manager reviews all bids and selects delivery person (typically lowest bid)
10. If non-lowest bid selected, manager must provide written justification
11. System assigns delivery person and notifies all bidders
12. Order status updated to "assigned" with delivery person information

Exceptional Scenarios

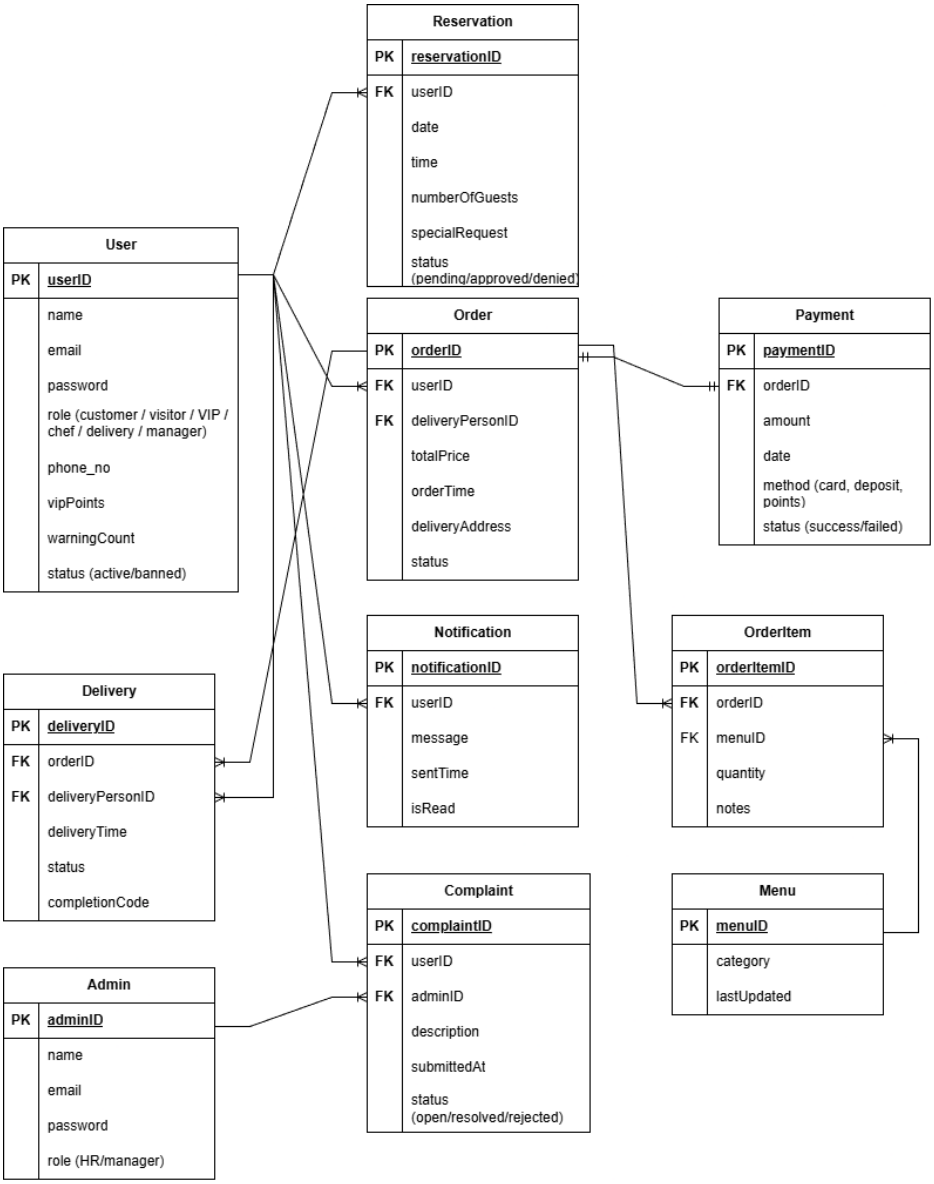
- E1 - Insufficient Balance: System blocks payment, issues warning
- E2 - Payment Gateway Error: Transaction fails, no funds deducted, no warning
- E3 - Non-Lowest Bid Selected: System requires justification from manager
- E4 - Account Closure: System refunds remaining balance, blacklists user
- E5 - No Bids Received: System extends window, notifies manager if still no bids



3. E-R Diagram

The E-R diagram shows all the main data tables used in our restaurant system and how they are related to each other. Each table includes its primary key (PK), any foreign keys (FK), and the important attributes needed to store information.

This model helps us understand how data flows through the system. Customers (Users) interact with Reservations, Orders, Payments, and Notifications. The Menu table stores the items available for ordering, and OrderItem links each order to the menu items inside it. Admins are included as well, mainly for handling complaints. Delivery records keep track of the delivery process for each order.



4. Detailed Design:

4.1 User Module:

Method: register (name, email, password, phone)
Input: name, email, password, phone
Output: success / failure
Main Functionality: Creates a new user account.

Pseudocode:

1. Check if a user with the same email already exists.

2. If yes, return "failure".
3. Validate input fields (email format, password requirements, etc.).
4. Insert new user record into User table with default role = customer.
5. Return "success".

Method: login (email, password)
Input: email, password
Output: success / failure
Main Functionality: Authenticates a user.

Pseudocode:

1. Retrieve user record using the provided email.
2. If no record found, return "failure".
3. Compare input password with stored password.
4. If match, return "success".
5. Otherwise return "failure".

Method: placeOrder(itemsList, deliveryAddress)
Input: itemsList (menuID + quantity), deliveryAddress
Output: orderID
Main Functionality: Allows a user to submit a new order.

Pseudocode:

1. Initialize totalPrice = 0.
2. For each item in itemsList:
    a. Retrieve menu item price.
    b. totalPrice = totalPrice + (price * quantity).
3. Insert a new order into the Order table and get generated orderID.
4. For each item in itemsList: a. Insert into OrderItem (orderID, menuID, quantity).
5. Return orderID.

Method: makeReservation (date, time, guests, specialRequest)
Input: date, time, numberOfGuests, specialRequest
Output: reservationID / denied
Main Functionality: Creates a reservation request.

Pseudocode:

1. Check if the date and time slot is available.
2. If not available, return "denied".
3. Insert a new reservation with status = "pending approval".
4. Return reservationID.

4.2 Admin Module:

Method: reviewComplaint (complaintID, decision)
Input: complaintID, decision (approve/resolve/reject)
Output: updated status message
Main Functionality: Admin handles complaints.

Pseudocode:

1. Retrieve complaint using complaintID.
2. If complaint not found, return "invalid complaint".
3. Update complaint status based on decision.
4. Save the updated complaint record.
5. Return "complaint updated".

Method: manageUserStatus (userID, action)
Input: userID, action (ban/unban/warn)
Output: success / failure
Main Functionality: Admin moderates users.

Pseudocode:

1. Retrieve user using userID.
2. If user not found, return "failure".
3. If action = "ban", set status = "banned".
4. If action = "unban", set status = "active".
5. If action = "warn", increase warningCount by 1.
6. Save changes.
7. Return "success".

Method: updateMenu (menuID, category, lastUpdated)
Input: menuID, category, lastUpdated
Output: success / failure
Main Functionality: Admin modifies menu items.

Pseudocode:

1. Retrieve menu item using menuID.
2. If not found, return "failure".
3. Update the menu category and lastUpdated fields.
4. Save changes.
5. Return "success".

4.3 Order System:

Method: createOrder(userID, itemsList, address)
Input: userID, itemsList, address
Output: orderID
Main Functionality: Creates a new order.

Pseudocode:

1. Verify user exists.
2. Calculate total price from itemsList.
3. Insert new order into Order table.
4. Get generated orderID.
5. Return orderID.

Method: addOrderItem (orderID, menuID, quantity)
Input: orderID, menuID, quantity
Output: success / failure
Main Functionality: Adds an item to an existing order.

Pseudocode:

1. Verify the order exists.
2. Verify the menu item exists.
3. Insert new entry into OrderItem table.
4. Return "success".

Method: updateOrderStatus (orderID, status)
Input: orderID, newStatus
Output: success / failure
Main Functionality: Updates order status.

Pseudocode:

1. Find order using orderID.
2. If not found, return "failure".
3. Update order status field.
4. Save changes.
5. Return "success".

4.4 Delivery System:

Method: assignDeliveryPerson (orderID)
Input: orderID

Output: deliveryID
Main Functionality: Assigns a delivery worker.

Pseudocode:

1. Find available delivery person (role = "delivery").
2. If none available, return "no delivery staff available".
3. Insert a new Delivery record with deliveryPersonID and orderID.
4. Return deliveryID.

Method: updateDeliveryStatus (deliveryID, status)
Input: deliveryID, status
Output: success
Main Functionality: Updates delivery tracking status.

Pseudocode:

1. Retrieve delivery record using deliveryID.
2. If not found, return "failure".
3. Update status field (out for delivery, delivered, failed).
4. Save changes.
5. Return "success".

4.5 Payment System:

Method: processPayment (orderID, method, amount)
Input: orderID, method (card/cash/points), amount
Output: paymentID / failure
Main Functionality: Processes payment for an order.

Pseudocode:

1. Retrieve order using orderID.
2. If order not found, return "failure".
3. Validate amount.
4. Insert new payment record with status = "processing".
5. Simulate or request payment confirmation.
6. Update payment status = "success".
7. Return paymentID.

Method: updatePaymentStatus (paymentID, status)
Input: paymentID, status
Output: success

Main Functionality: Updates payment result.

Pseudocode:

1. Retrieve payment record.
2. Update the status field.
3. Save changes.
4. Return "success".

4.6 Notification System:

Method: sendNotification (userID, message)
Input: userID, message
Output: notificationID
Main Functionality: Sends a message to the user.

Pseudocode:

1. Verify user exists.
2. Insert a new notification with the message and current time.
3. Return notificationID.

4.7 Complaint System

Method: createComplaint (userID, description)
Input: userID, description
Output: complaintID
Main Functionality: Submits a new complaint.

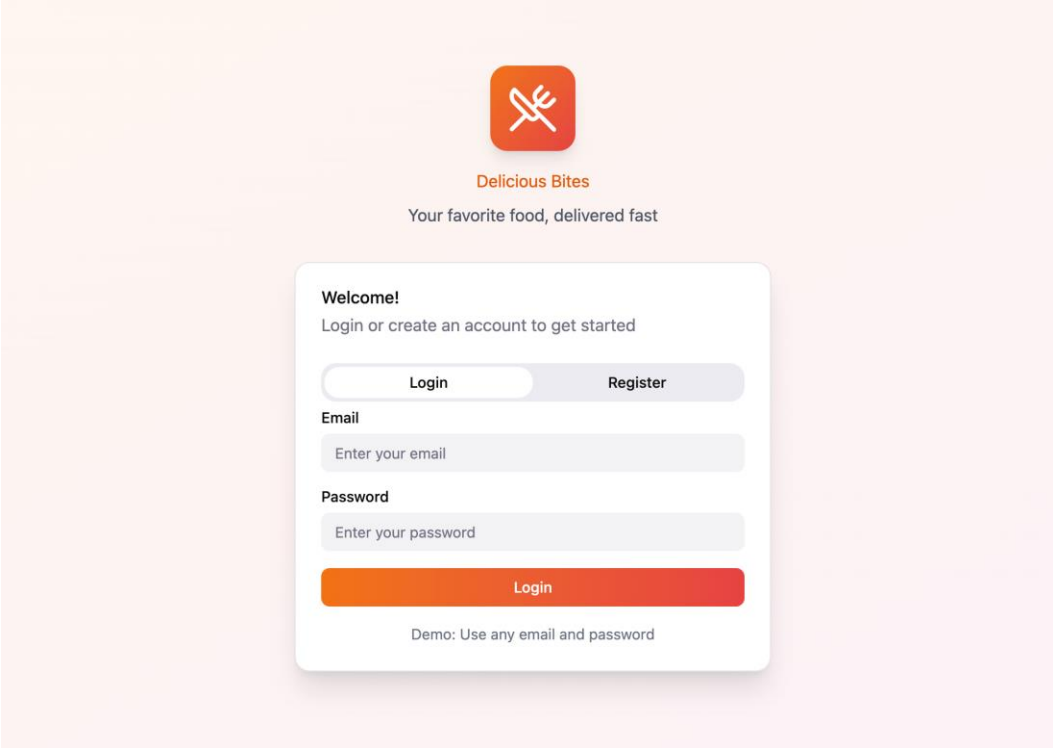
Pseudocode:

1. Verify user exists.
2. Insert a new record into Complaint table with status = "open".
3. Return complaintID.

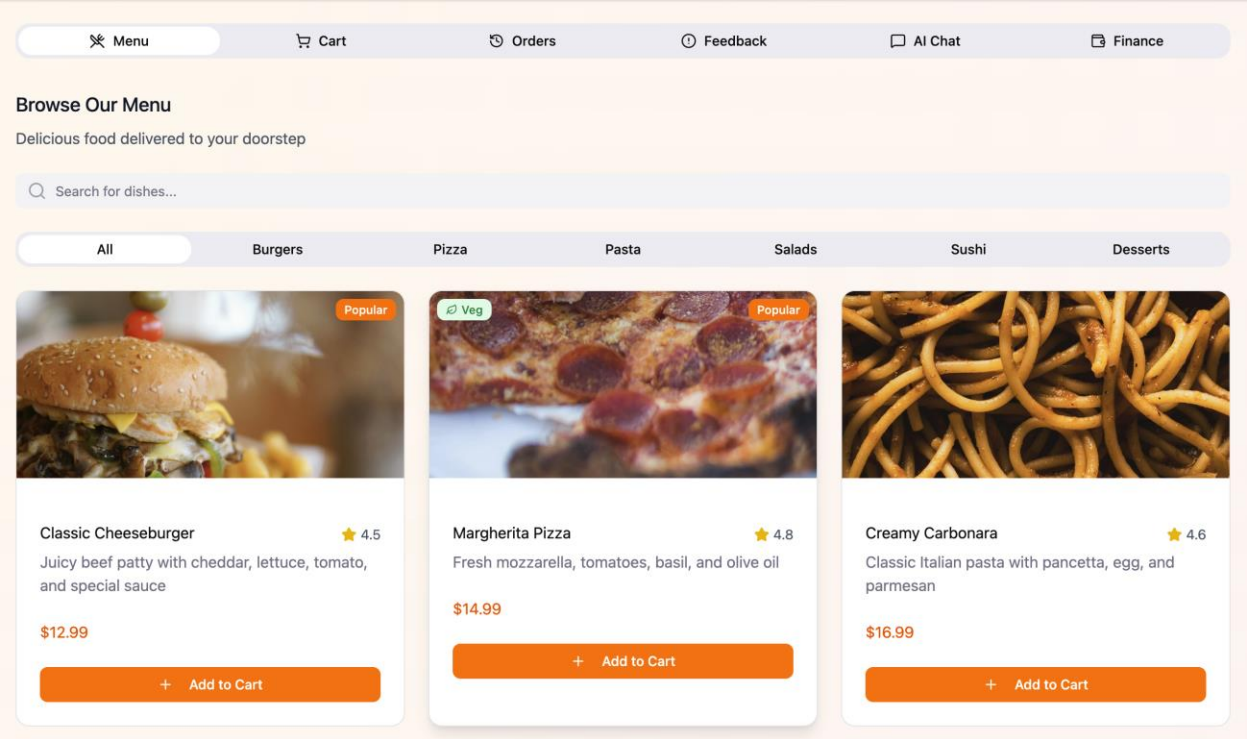
5. System Screens

These UI designs were created using Canva, and they serve as visual prototypes for our project. Our goal is to develop the actual website so that its layout, features, and overall user experience closely match these designs.

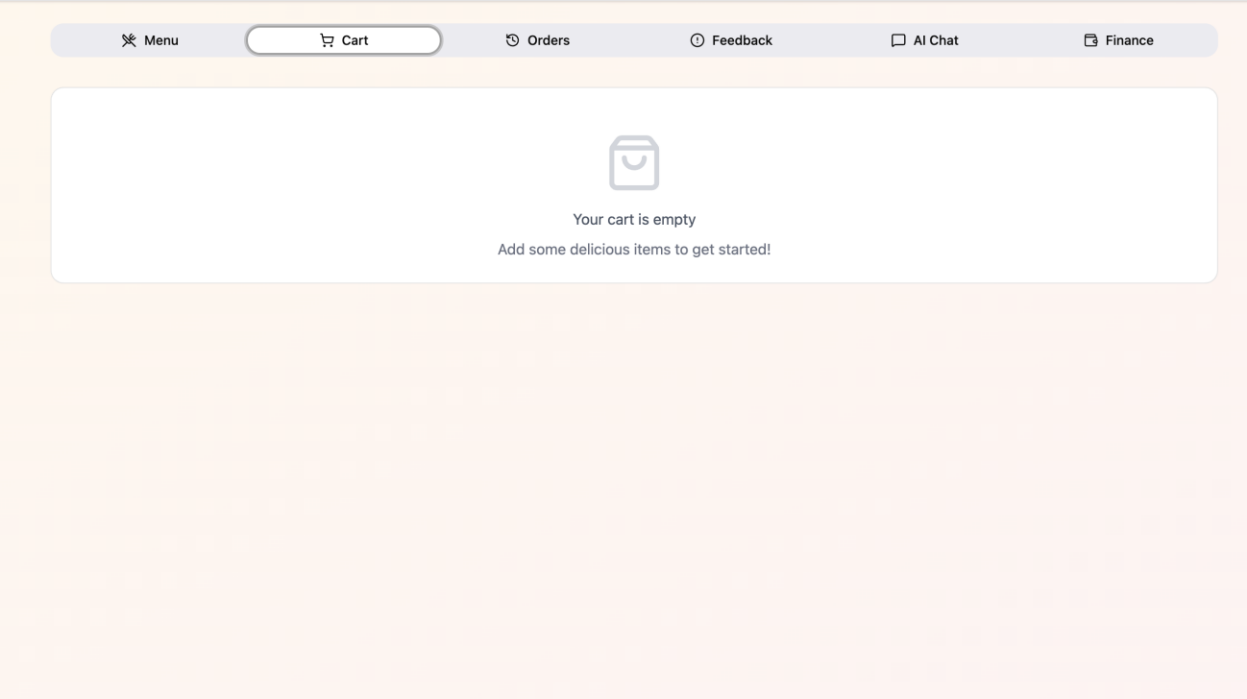
- GUI mockups or screenshots :



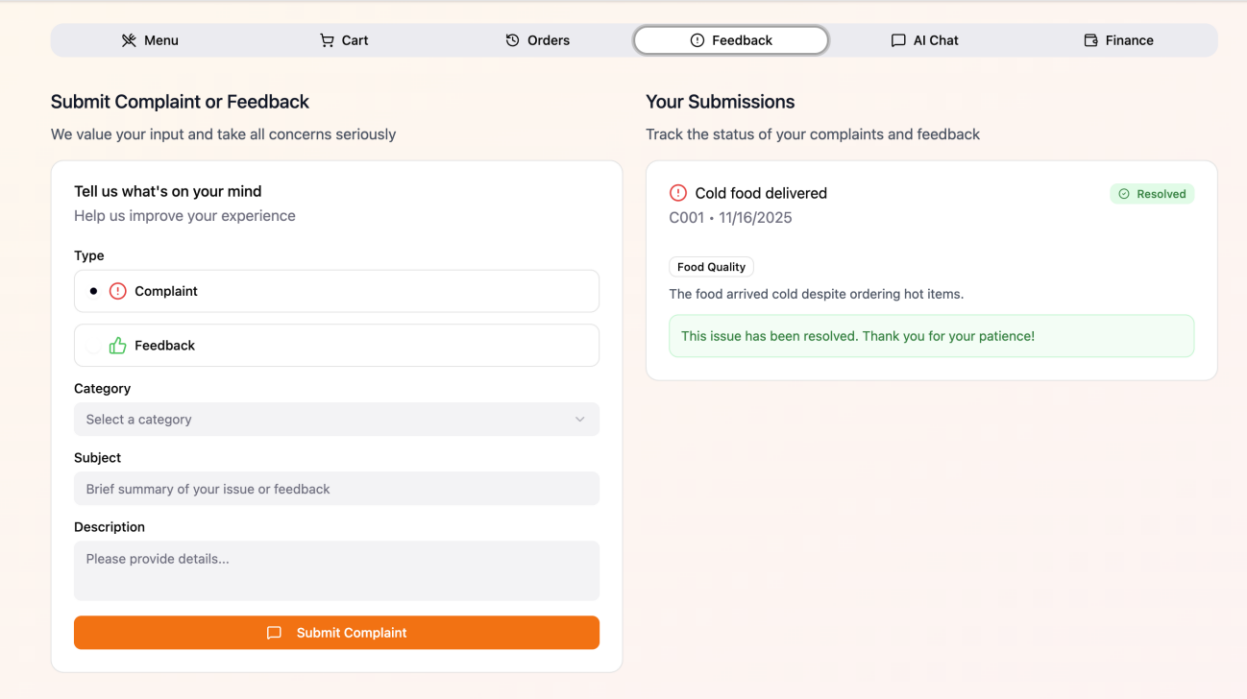
- Prototype Functionality:
    - Basic Login Simulation
    - How it Works:
        - The user enters any email and password.
        - When they click Login, the system automatically proceeds to the next screen (e.g., Home, Menu, or Dashboard).



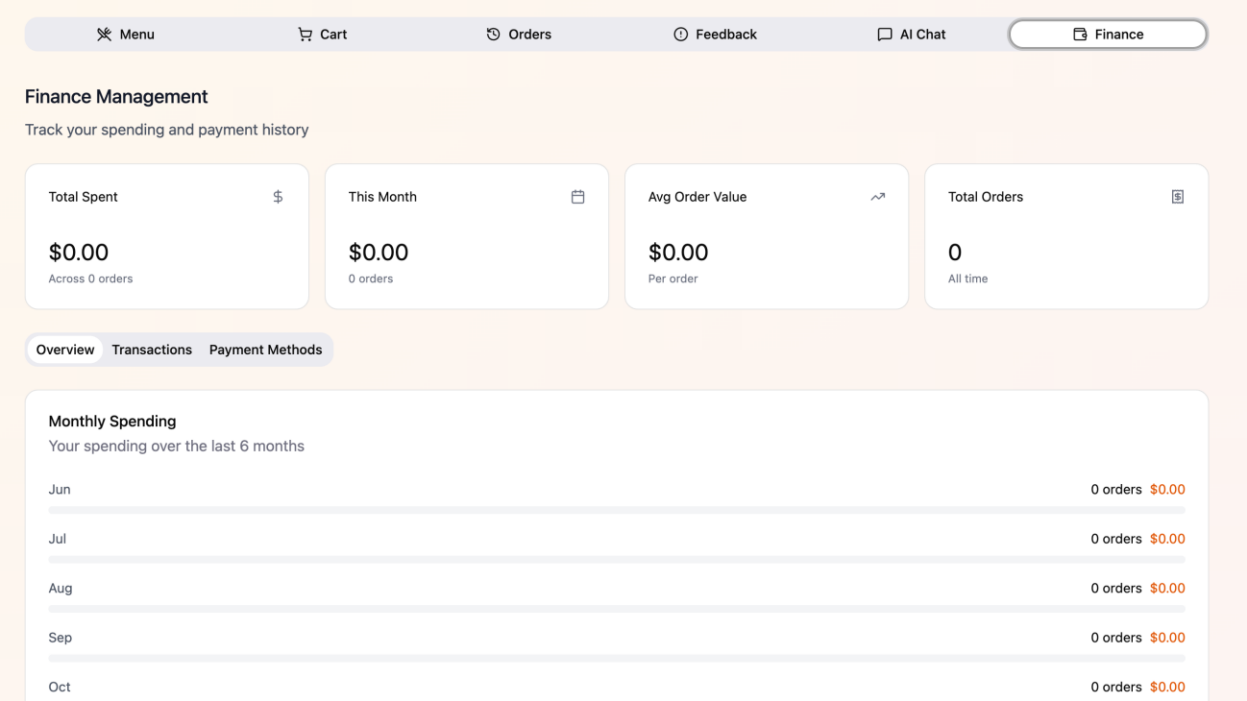
- Prototype Functionality:
    - Browse & Add-to-Cart Simulation: In this prototype, the menu page allows users to browse dishes and add items to the cart.
    - How it Works:
        - Users can scroll through different categories (Burgers, Pizza, Pasta, etc.).
        - Each food card displays a picture, description, rating, and price.
        - When the user clicks “Add to Cart,” the system visually confirms the action



- Prototype Functionality:
  - Cart (When empty):
    - this prototype, when the user first opens the Cart tab, they see an empty cart screen. This shows what the cart looks like before the user selects any dishes.
  - Added Items Prototype:
    - When a user clicks “Add to Cart” on any menu item, the system simulates the cart being updated.
    - The selected food item appears inside the cart.
    - It displays:
      - ★ The item name Price
      - ★ Quantity selector (e.g., + / – buttons)
      - ★ A subtotal



- Prototype Functionality:
    - Feedback & Complaint Submission:
        - Submitting a Complaint or Feedback:
            - In this prototype, users can fill out a form to submit a complaint or feedback by selecting a type, choosing a category, and writing a subject and description. When the user clicks Submit Complaint, the system simulates a submission.
        - Viewing Submission Status:
            - On the right side, users can see a sample submission that demonstrates how their complaint or feedback would appear after being submitted.



- Prototype Functionality: Finance Management
    - Viewing Spending & Order Stats: In this prototype, the Finance screen shows a dashboard with statistics like:
        - Total Spent Spending
        - This Mont1h Average
        - Order Value
        - Total Number of Orders

6. Group Memos

- Meeting notes
    - Task division to avoid overlapping responsibilities. (2 people front end and 2 people backend)
    - Group meeting Tuesdays and Thursdays (4 to 5 pm)
    - Next meeting will focus on connecting the front-end components with backend.
- Team concerns:
    - We have concerns about time balancing workload with other classes.
    - There is uncertainty about implementing backend features such as login authentication and order tracking. This will be extra features we might work on if we extra time.
    - We discussed challenges integrating the AI Chat module, especially using external APIs.

7. Address of the git repo - https://github.com/shreychow/SD_Foods_csc322