Software Design Document

for

QuickCrave

Version 1.0

Prepared by: Cache Me If You Can (Team 9)

Dnyanesh Hemendra Shinde	230390	dnyanesh23@iitk.ac.in
Nikhilesh Joshi	230700	nikhileshj23@iitk.ac.in
Kamatam Akshay Reddy	230517	areddy23@iitk.ac.in
Arihant Kumar	230189	arihantkmr23@iitk.ac.in
Kshitij Gupta	230581	kshitijg23@iitk.ac.in
Parnika Mittal	230736	mparnika23@iitk.ac.in
Rohit Somani	230871	rohitvs23@iitk.ac.in
Suvid Goel	231065	gsuvid23@iitk.ac.in
Yashaswi Raj	231191	yashaswir23@iitk.ac.in
Yatharth Dangi	231197	yatharth23@iitk.ac.in

Course: CS253

Mentor TA: *Nij Bharatkumar Padariya*

Submission Date: *February 7, 2025*

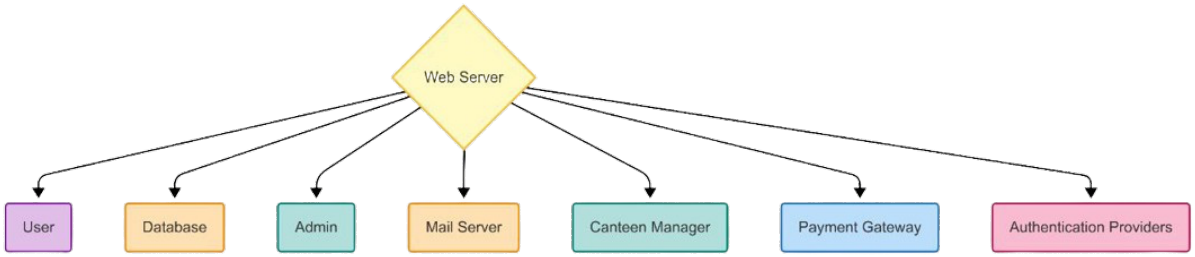
Contents

1 Context Design

1.1 Context Model

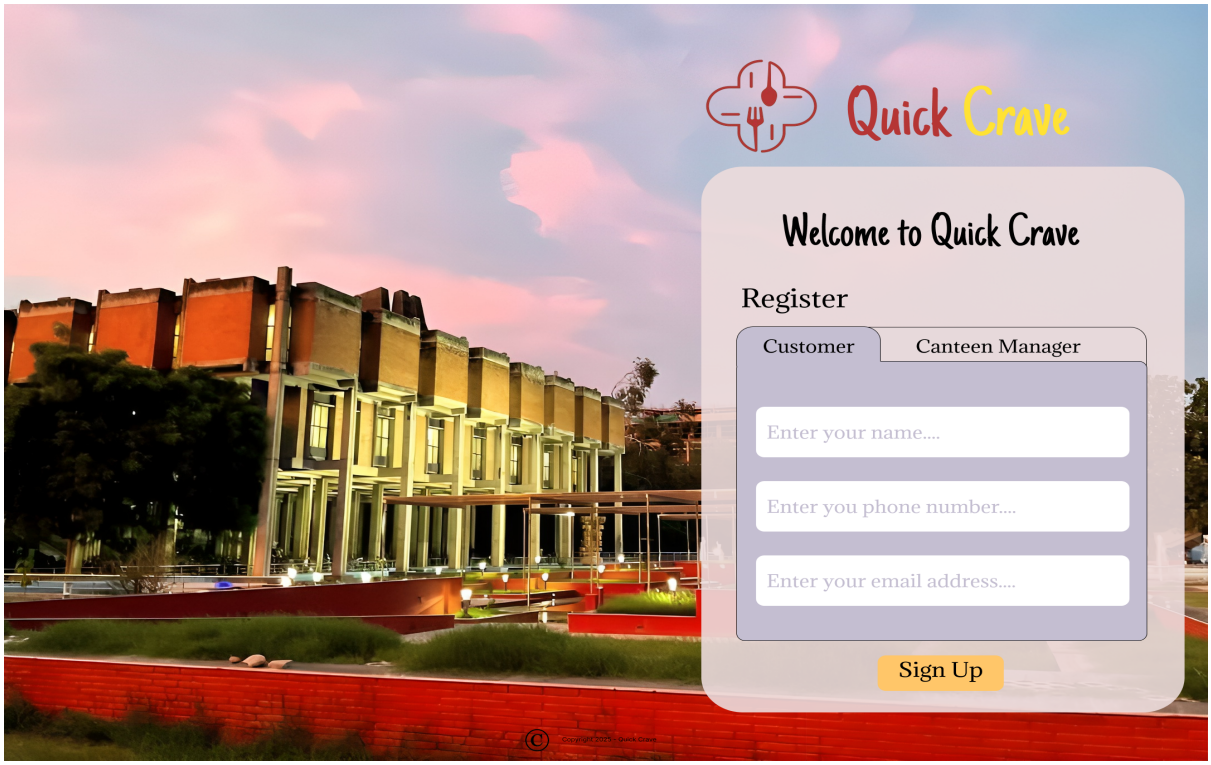
The context model shows how the system interacts with external entities, the environment, and their relationships. This aids in defining system requirements and ensuring that all external dependencies and interactions are properly accounted for. A brief explanation of the context model is as follows:

- **User:** Interacts with the server via website
- **Canteen Manager:** Interacts with the website to manage orders and update menu item details in the database.
- **Admin:** Monitor and oversee the smooth operation of the entire platform, ensuring efficiency and reliability.
- **Database:** Stores all the data of the system, including user credentials, order information, and menu item details.
- **Mail server:** Facilitates communication by sending OTPs and assisting with password resets through email verification.
- **Payment Gateway:** integrates payment functionality, allowing financial transactions within the system.
- **Authentication Providers:** Manages user authentication using Google OAuth for secure OTP verification.

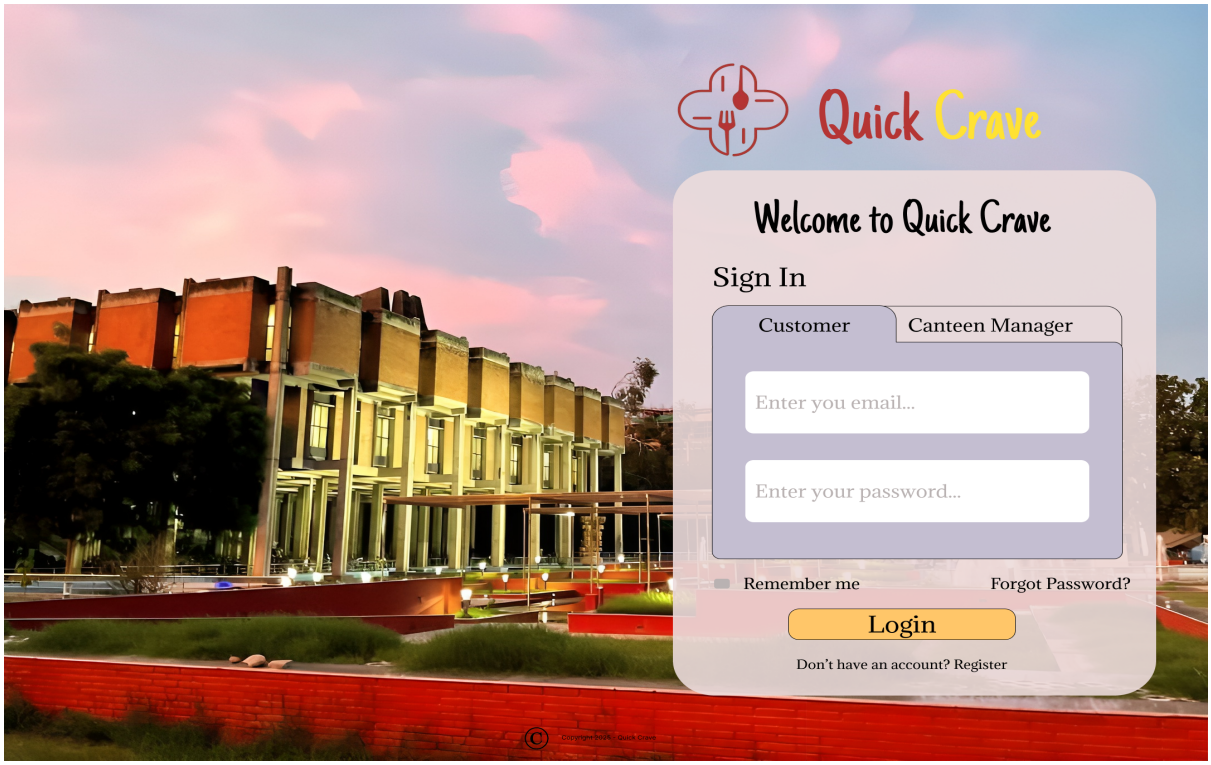


1.2 Human Interface Design

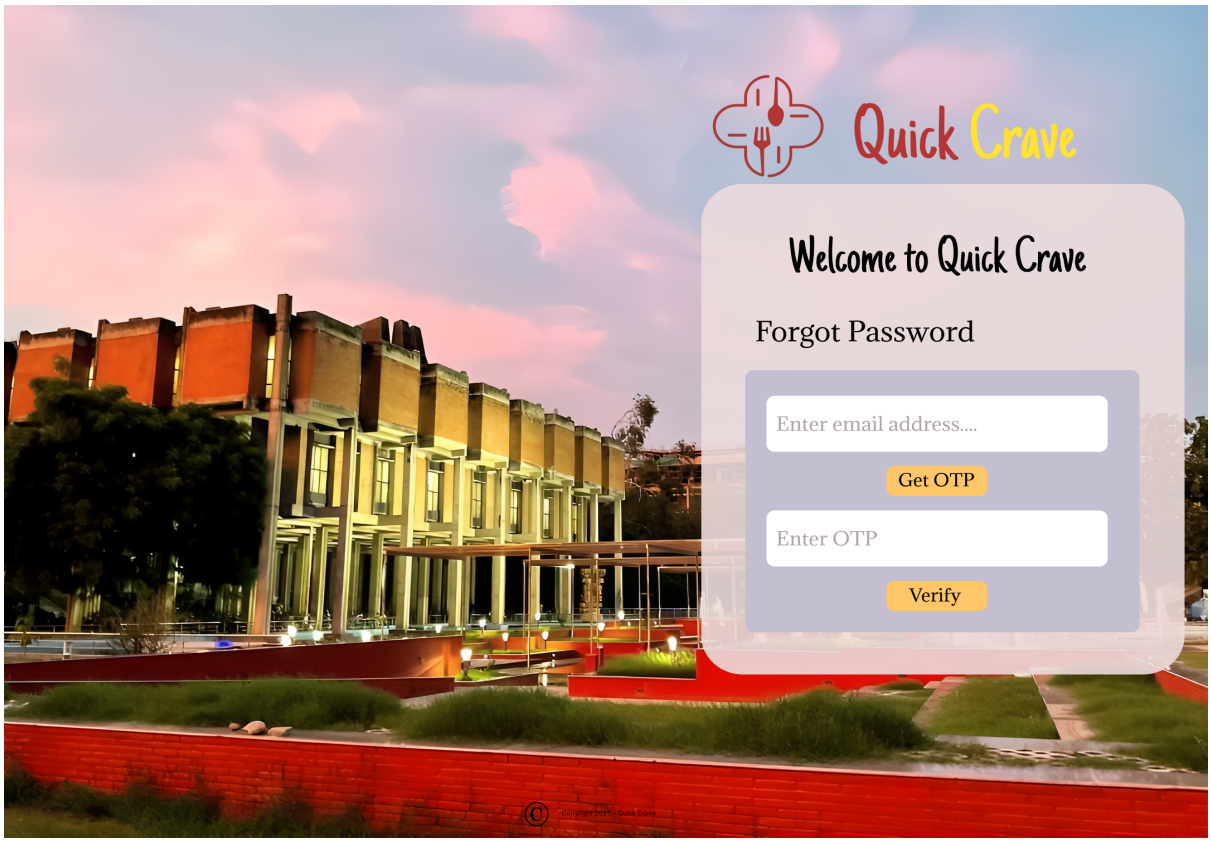
1.2.1 Sign Up Page

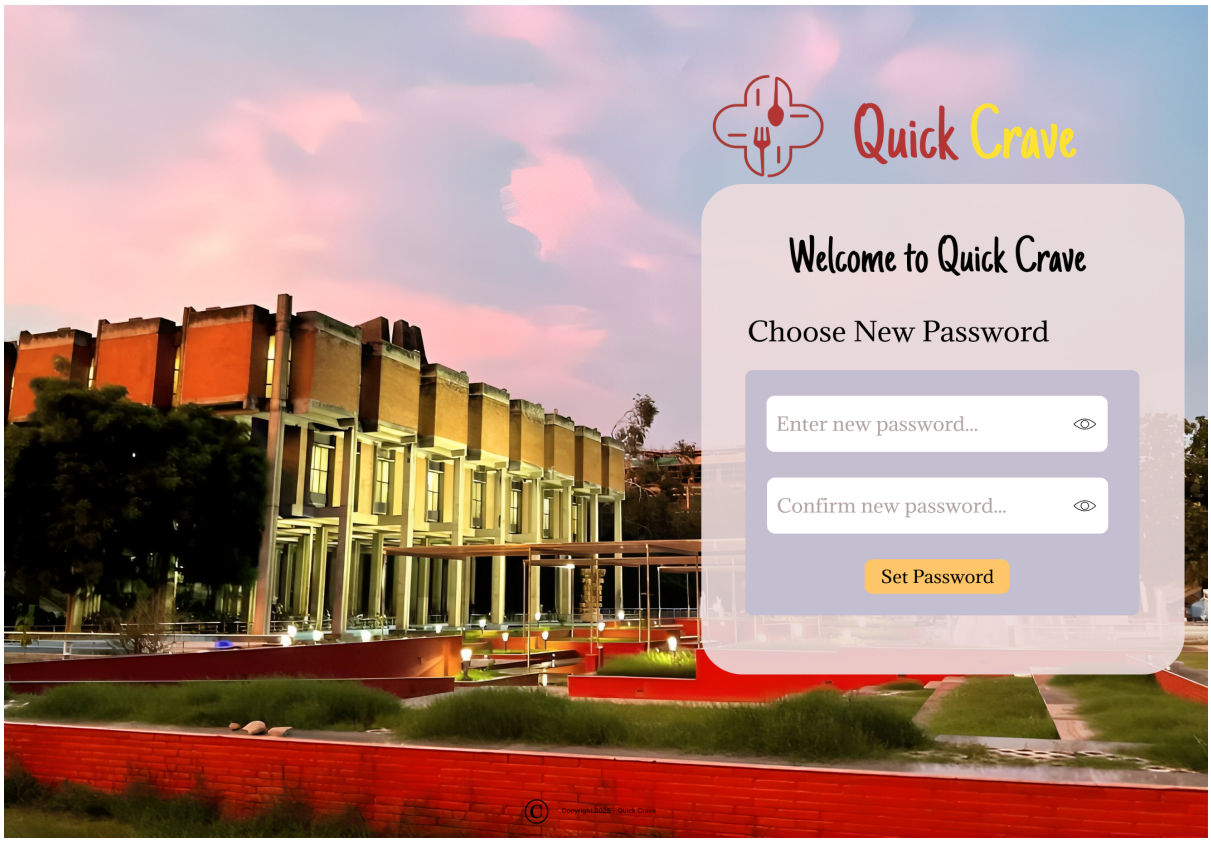


1.2.2 Login Page

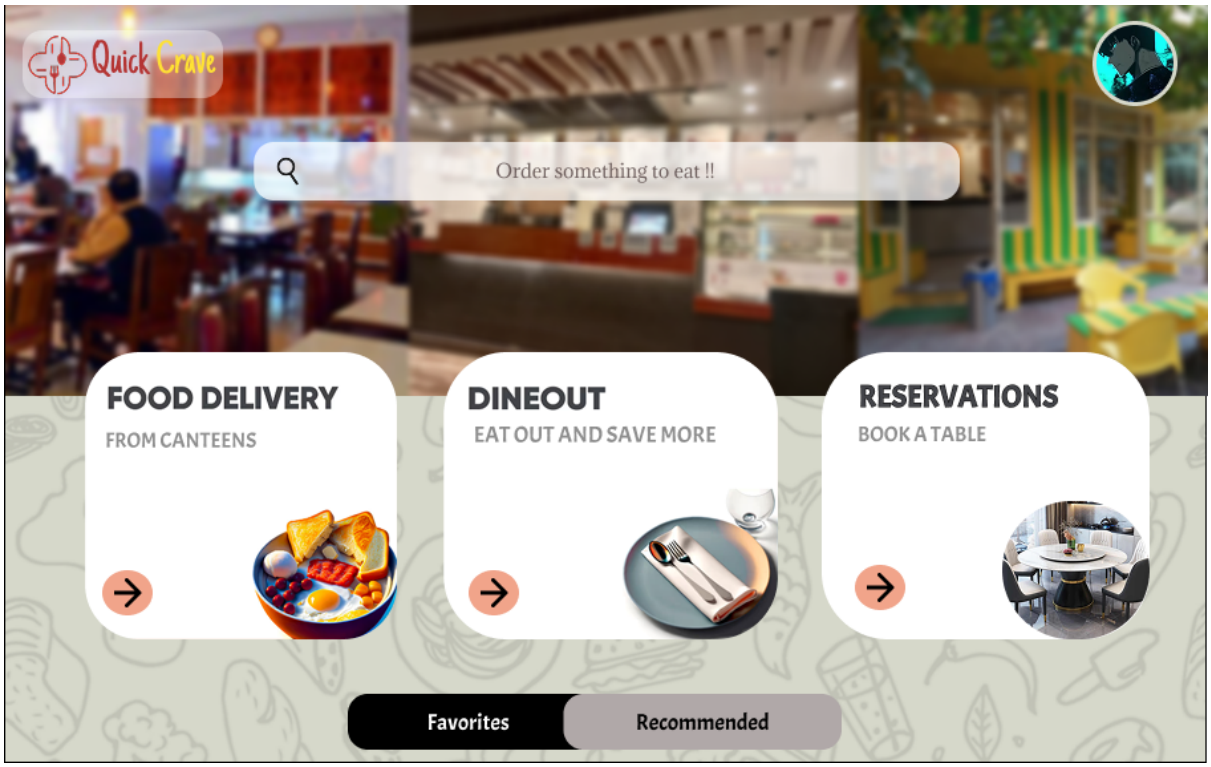


1.2.3 Forgot Password Pages

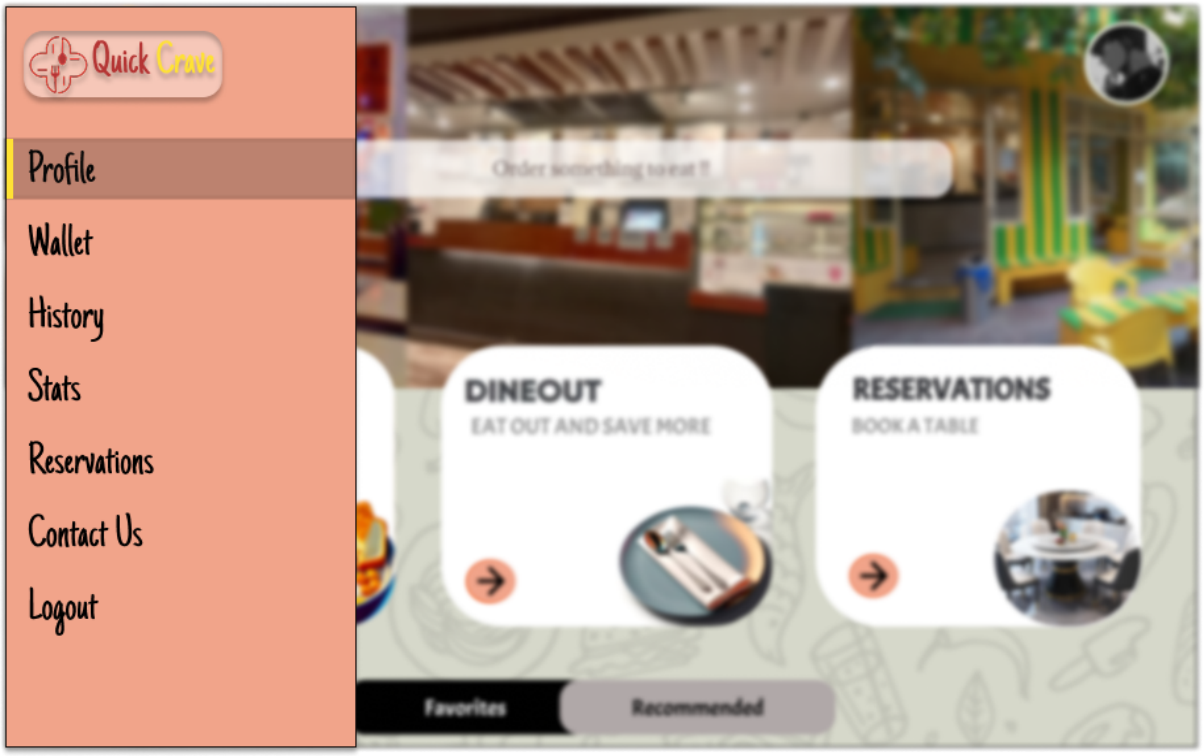




1.2.4 Customer Home Page

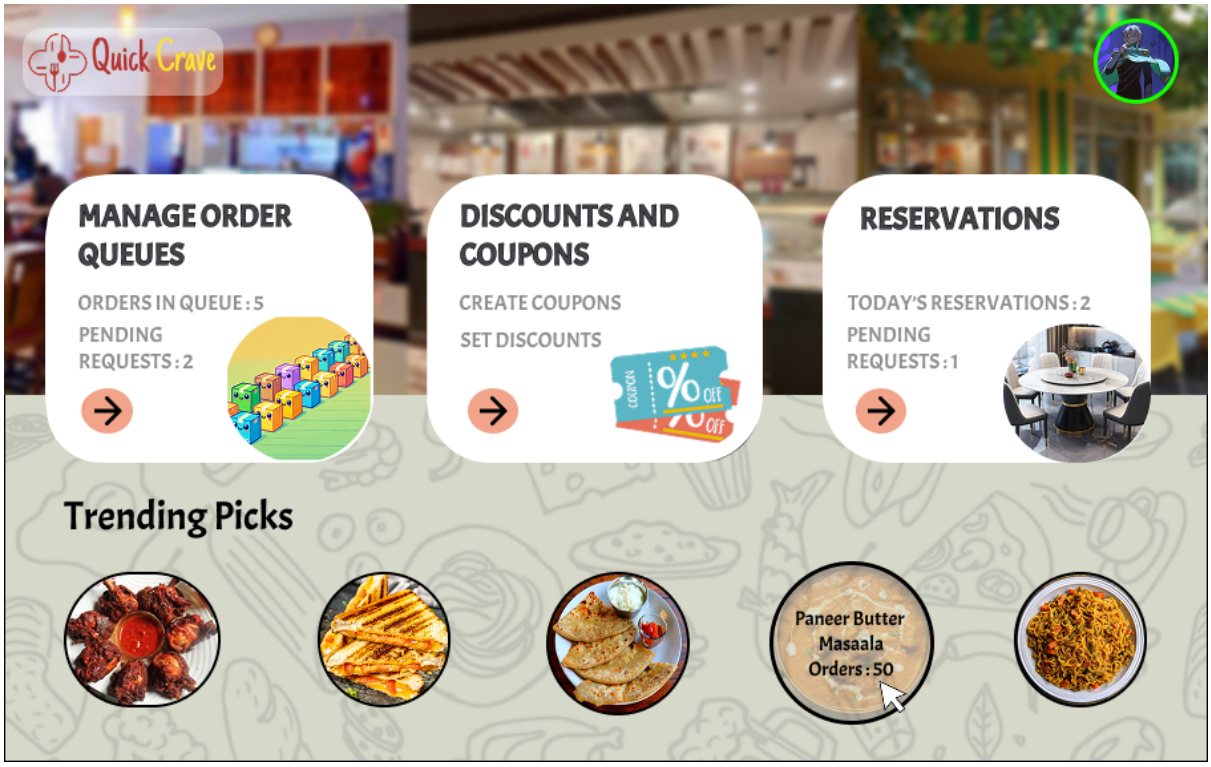


1.2.5 Customer Side Menu



1.2.6 Customer Profile Page

1.2.7 Canteen Manager Home Page



2 Architecture Design

2.1 Overview

For QuickCrave, we've integrated the Model-View-Controller (MVC) with Layered Architecture to achieve scalability, maintainability, and flexibility:

- **MVC** separates the app into:
  - **Model**: Manages data and business logic.
  - **View**: Handles the user interface.
  - **Controller**: Processes user requests and updates the Model and View.
- **Layered Architecture** adds further modularization:
  - **Presentation Layer**: Manages the UI.
  - **Business Logic Layer**: Handles core logic.
  - **Data Layer**: Manages database interactions.

2.2 Benefits of Integrated MVC and Layered Architecture

- **Clear Separation of Concerns**: MVC ensures distinct responsibilities for data, UI, and user interactions, while Layered Architecture isolates presentation, logic, and data.
- **Scalability**: New features or components can be added without disrupting other layers, ensuring the app can grow smoothly.
- **Maintainability**: Easy to modify or extend business logic and UI independently, reducing complexity in the long term.
- **Team Collaboration**: Clear divisions of work allow different teams to focus on specific layers without interference.

2.3 Data Flow in Integrated Architecture

1. **User Action (View)**: User interacts with the UI (e.g., placing an order).
2. **Controller**: Receives input, processes it, and updates the Model.
3. **Business Logic Layer**: Validates and processes the request (e.g., order details, payment).
4. **Model**: Interacts with the Data Layer to save or fetch data.
5. **View Update**: The Controller updates the View with the results.

2.4 Why Not Other Architectures?

- **Layered Architecture Alone**: Lacks explicit control over user interaction, making it harder to manage UI flow and scalability.
- **Microservices**: Too complex for the current scale of the app, adding unnecessary overhead.

2.5 Conclusion

The integrated MVC and Layered Architecture provides a clear structure for QuickCrave, enabling modular development, smooth scalability, and efficient maintenance. This architecture supports future growth while keeping development manageable and organized.

2.6 MVC Architecture for QuickCrave App

2.6.1 Model: Data and Business Logic

The Model represents the core data structure and business logic of the app. It handles the primary functionality, such as user data, food orders, and restaurant details. The Model interacts directly with the database to retrieve and store data.

Responsibilities of the Model:

- User authentication data and session management.
- Retrieving restaurant menus, availability, and prices.
- Processing orders, including status updates and transaction handling.
- Interfacing with external services (e.g., ML models for recommendations, payment APIs).

2.6.2 View: User Interface and Presentation

The View is responsible for displaying the UI and presenting data to the user. It only receives data from the Controller and does not directly modify the Model.

Responsibilities of the View:

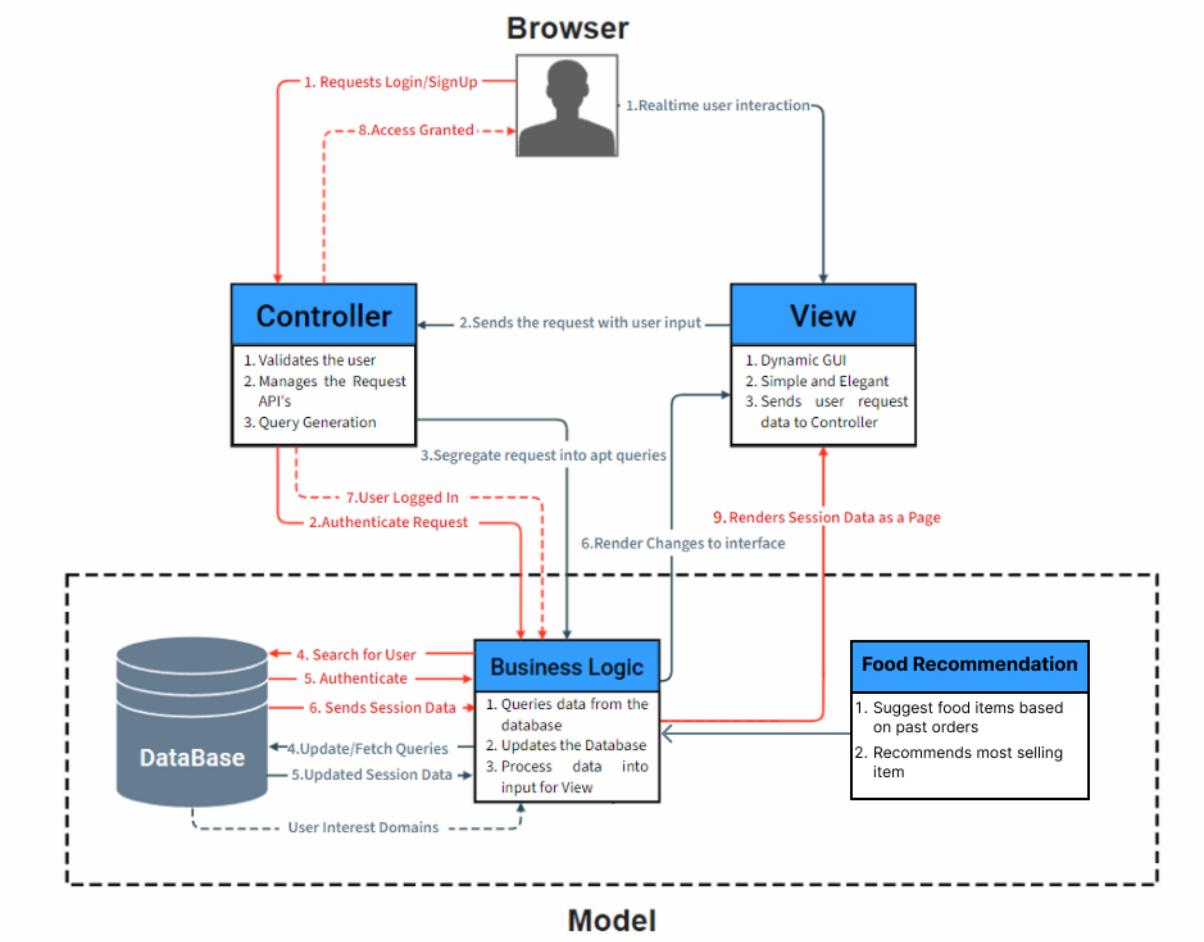
- Displaying menus, food items, and restaurant details.
- Showing user account information (profile, past orders).
- Handling user interactions such as login, signup, and order placement.
- Updating UI dynamically based on real-time events (e.g., order updates).

2.6.3 Controller: Handling Requests and Coordination

The Controller acts as the intermediary between the Model and View. It handles user inputs, updates the Model based on these inputs, and then updates the View accordingly.

Responsibilities of the Controller:

- Receiving and validating user input (e.g., login, signup, order request).
- Coordinating with the Model to fetch or update data.
- Sending data to the View for rendering.
- Handling user-driven events like button clicks, form submissions, and real-time interactions.



2.7 Integration of MVC with Layers

- **Model Layer**: Handles data-related operations, including database access, external API integrations, and business rules.
- **View Layer**: Focuses solely on presenting the data to the user in an intuitive manner.
- **Controller Layer**: Acts as a bridge, managing the flow of data between the Model and View, ensuring smooth communication and user interaction.

2.8 Layered Business Logic

In a Layered Architecture, the Business Logic Layer handles core functionalities, like processing orders, managing payments, and applying business rules, without interacting directly with the UI or data storage.

Key Components:

- **Application Services**: Coordinates high-level operations, e.g., order management and payment processing.
- **Domain Services**: Handles domain-specific logic, such as recommendations or discount calculation.
- **Validation & Business Rules**: Ensures data meets business requirements, e.g., user authentication or order validation.

- **Data Layer Integration**: Retrieves and stores necessary data to execute business logic.
- **Transactional Operations**: Manages multi-step processes, ensuring transactions are successful or rolled back.

Advantages:

- Separation of Concerns: Keeps UI and data handling separate from business rules.
- Maintainability: Simplifies updating business logic without affecting UI or data.
- Scalability: Easier to extend and modify business functionality as the app grows.
- Reusability: Allows reuse of services across different parts of the app.

Flow:

1. Controller sends a request to the Business Logic Layer.
2. Business Logic Layer processes the request and returns the result.
3. Controller updates the View based on the result.

This structure ensures a clean, modular, and maintainable design.

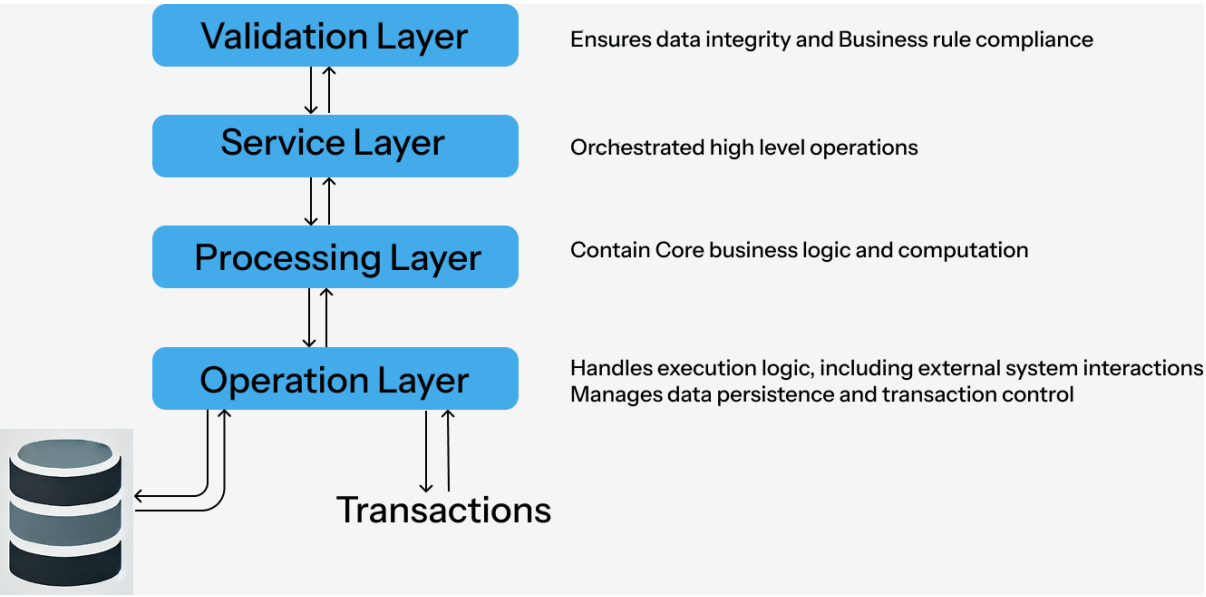
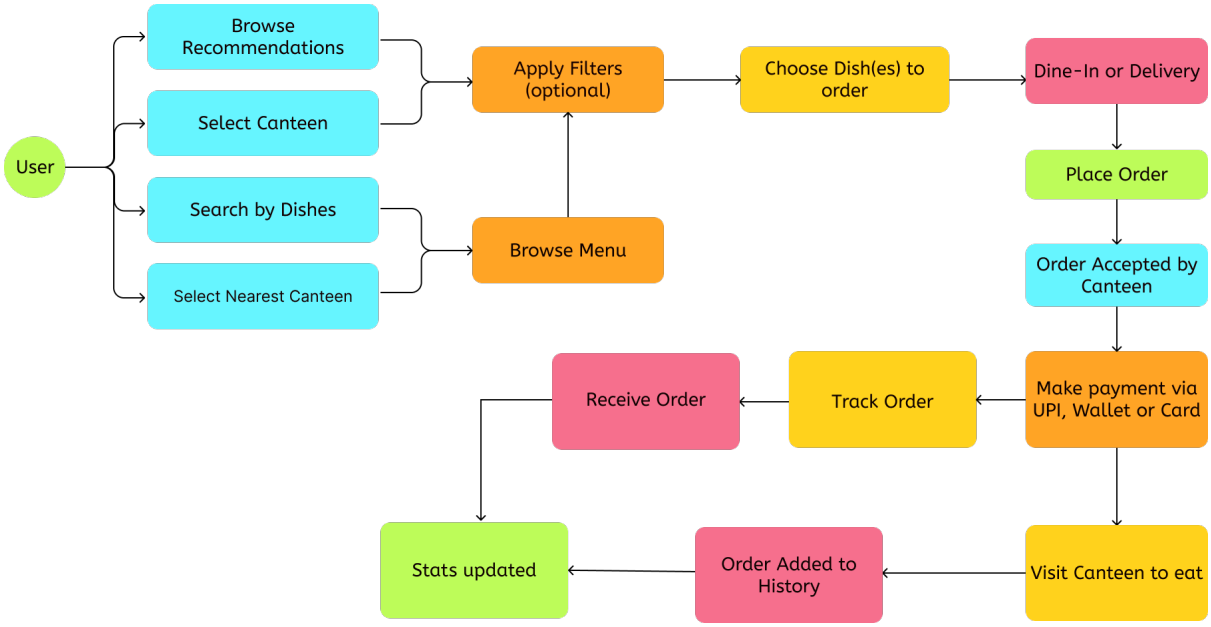


Figure 1: Business Logic Layered Architecture

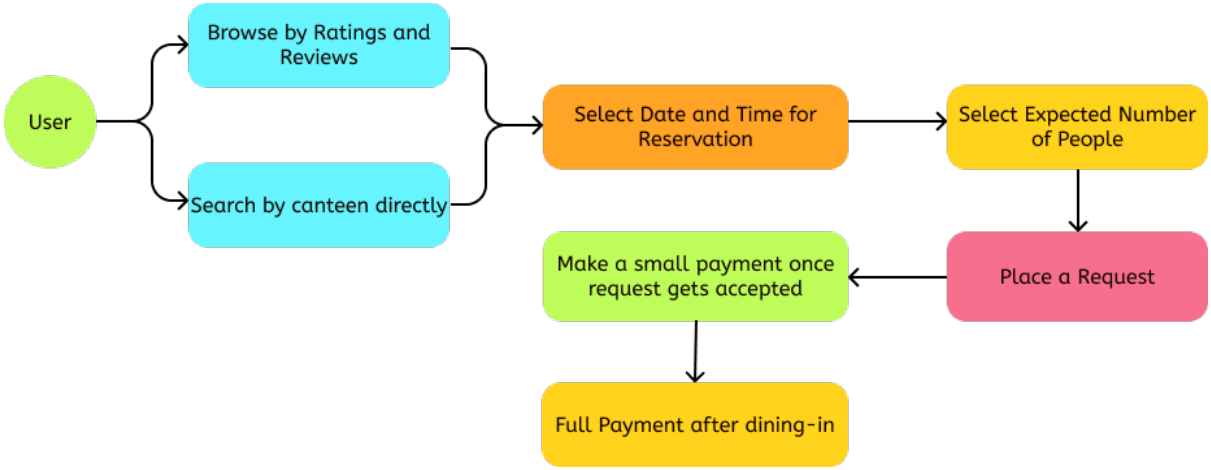
3 Object-Oriented Design

3.1 Use Case Diagrams

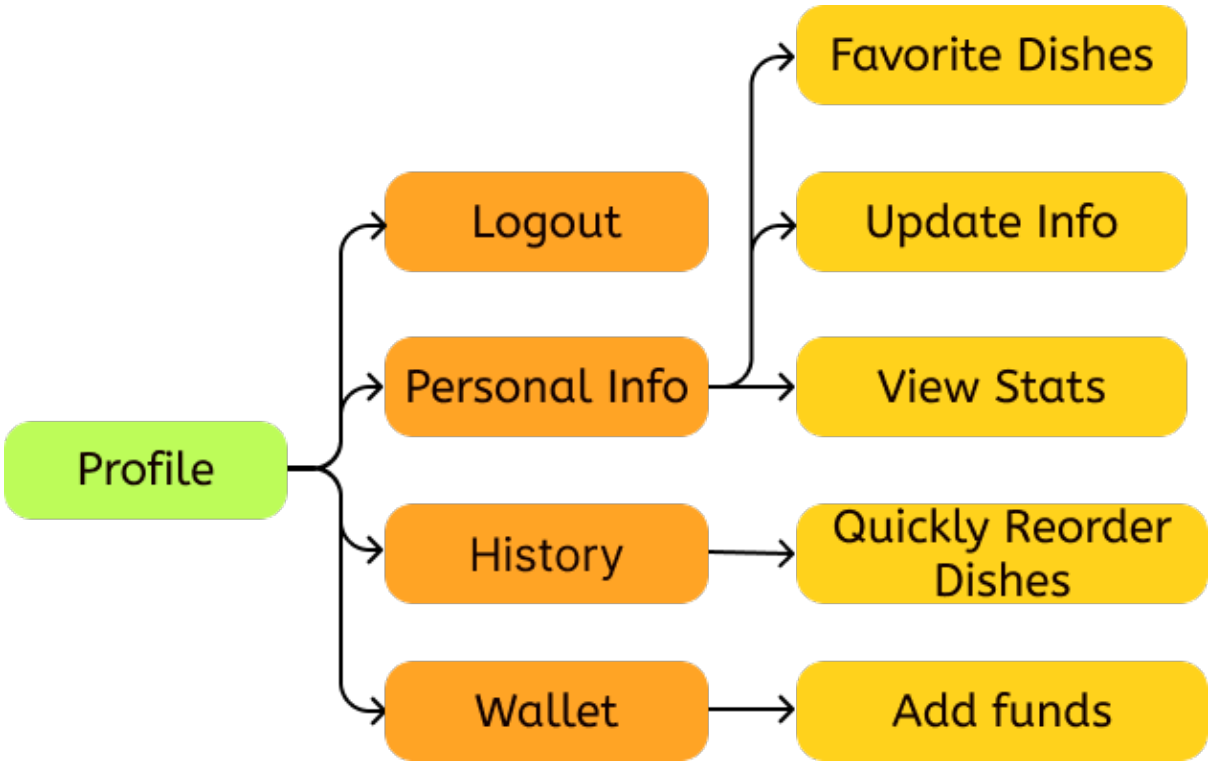
3.1.1 Customer Use Case 1: Ordering Food



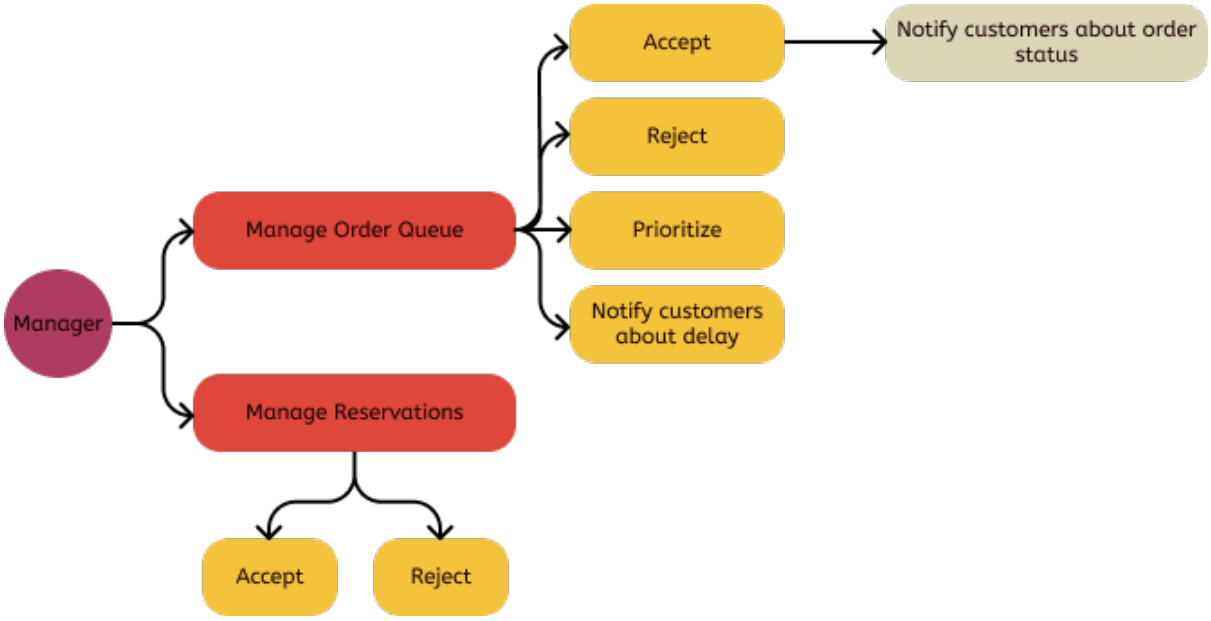
3.1.2 Customer Use Case 2: Making a Reservation



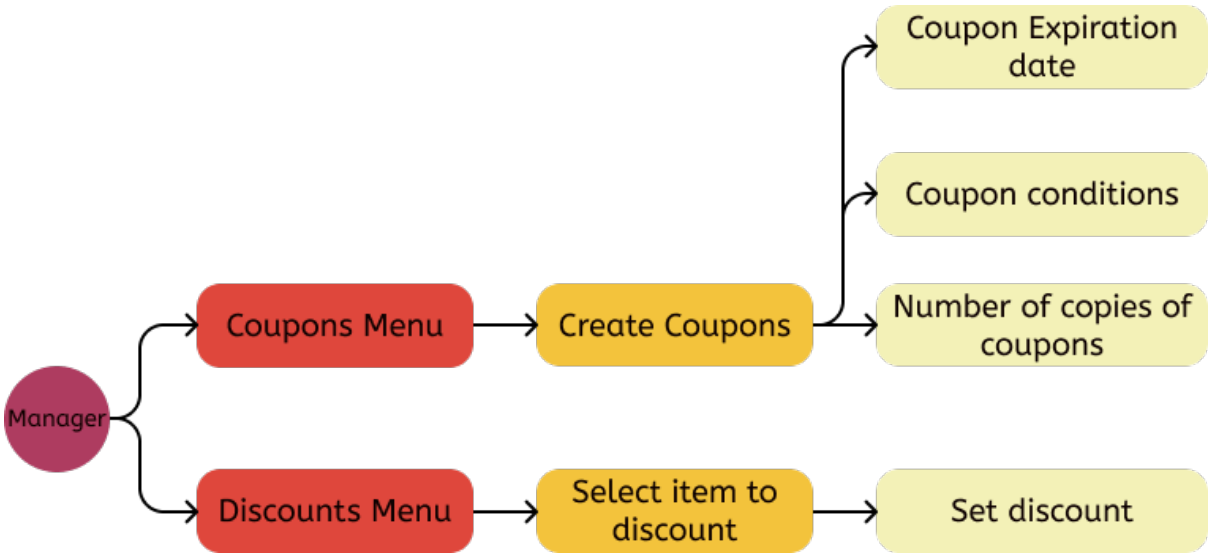
3.1.3 Customer Use Case 3: Updating Profile and Adding funds to Wallet



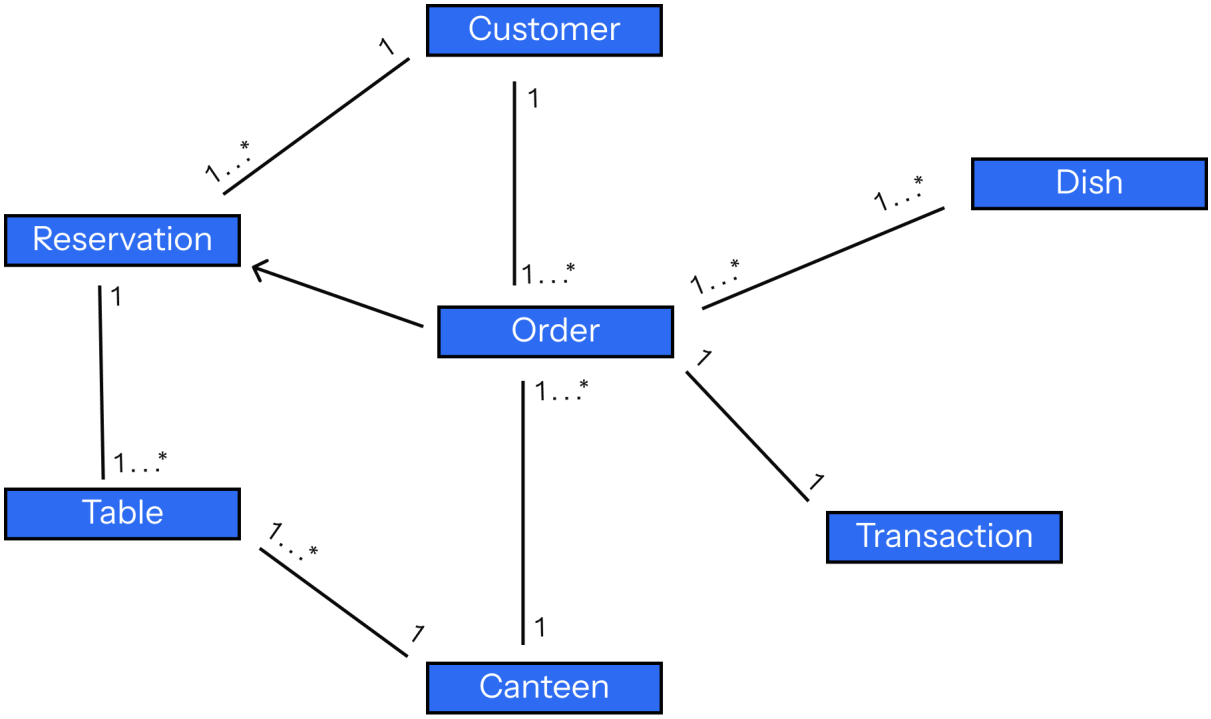
3.1.4 Canteen Manager Use Case 1: Managing Order and Reservation Queue



3.1.5 Canteen Manager Use Case 2: Updating Discounts on Dishes and Creating Coupons



3.2 Class Diagram



Customer/User	
ID:	Unsigned Integer
Password:	String
Wallet:	Number
Email:	String
Fav Dish:	Dish
Order History:	List of Orders
Current Orders:	List of Orders
Profile_update() **wallet_add_funds()** **order()** **reservation()**	

Table	
Num_of chairs:	Integer
Time interval: <Start , End>	List of Pairs

Dishes	
ID:	Unsigned Integer
Owner ID:	Unsigned Integer
Price:	Number
Canteen Name:	String
Availability:	Integer
Discount:	Integer
Photo:	File/Image
update() **availability_update()**	

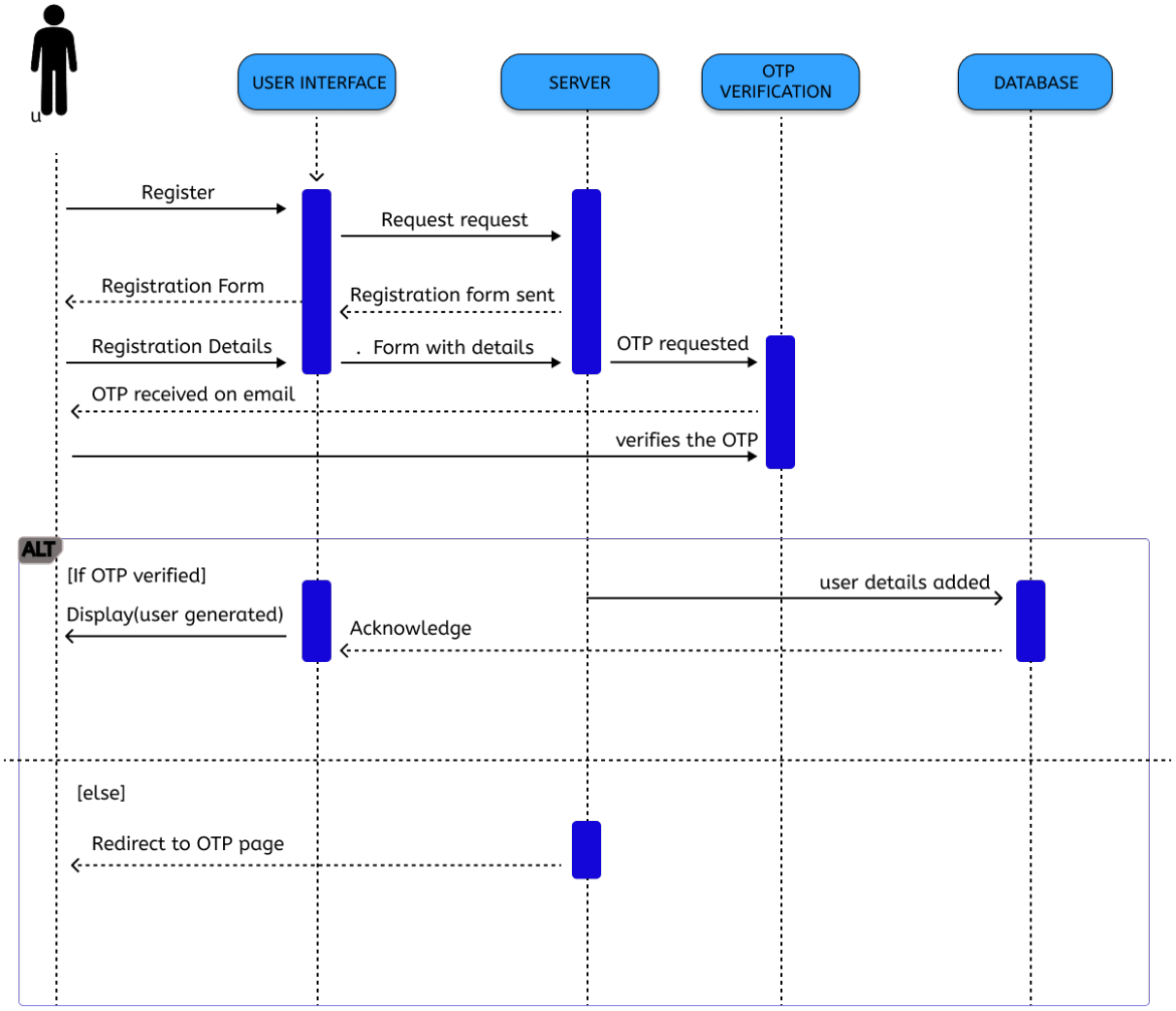
Transaction	
ID:	String
User ID:	Unsigned Integer
Ride ID:	Unsigned Integer
Amount:	Number
Time:	Time
Payment Method:	String
Status:	Boolean

Canteen	
ID:	Unsigned Integer
Name:	String
Owner ID:	Unsigned Integer
Password:	String
Menu:	List of Dishes
Queue:	List of Orders
Opening Time:	Time Format (HH:MM)
Closing Time:	Time Format (HH:MM)
List of Tables:	List of Table Objects
Auto_accept:	Boolean
profile_update() **order_accept_prompt()** **update_order_queue()** **manage_reservations()** **coupons_update()** **discounts_update()**	

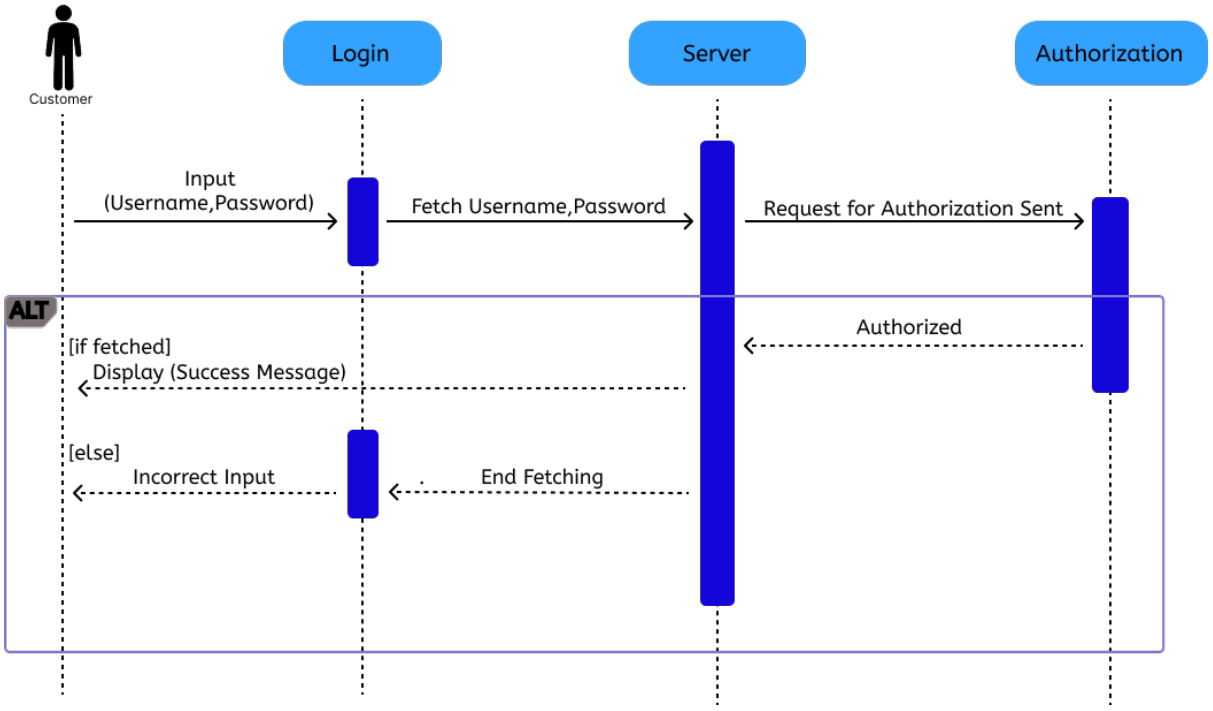
Order	
ID:	Unsigned Integer
User ID:	Unsigned Integer
Booking Time:	Time
List of Pair {Dish,Quantity}:	List of Pairs
Total Amount:	Number
update_order() **request_order()** **place_order()** **update_order_status()**	

3.3 Sequence Diagram

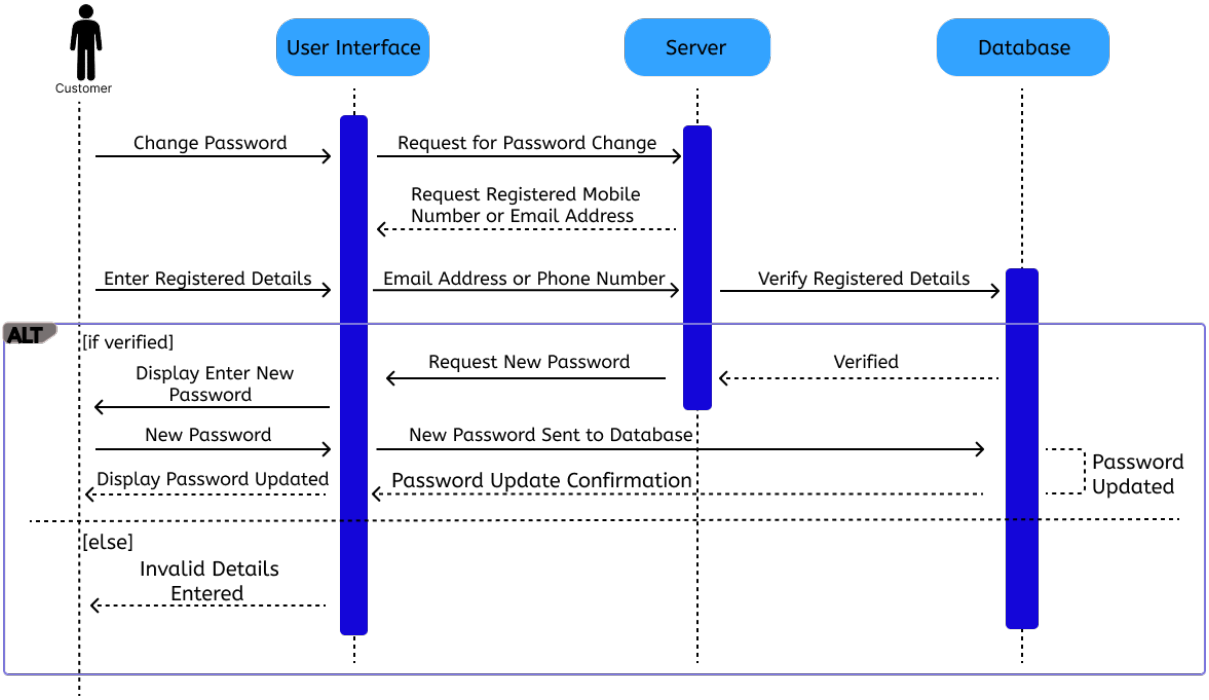
3.3.1 Registration Sequence



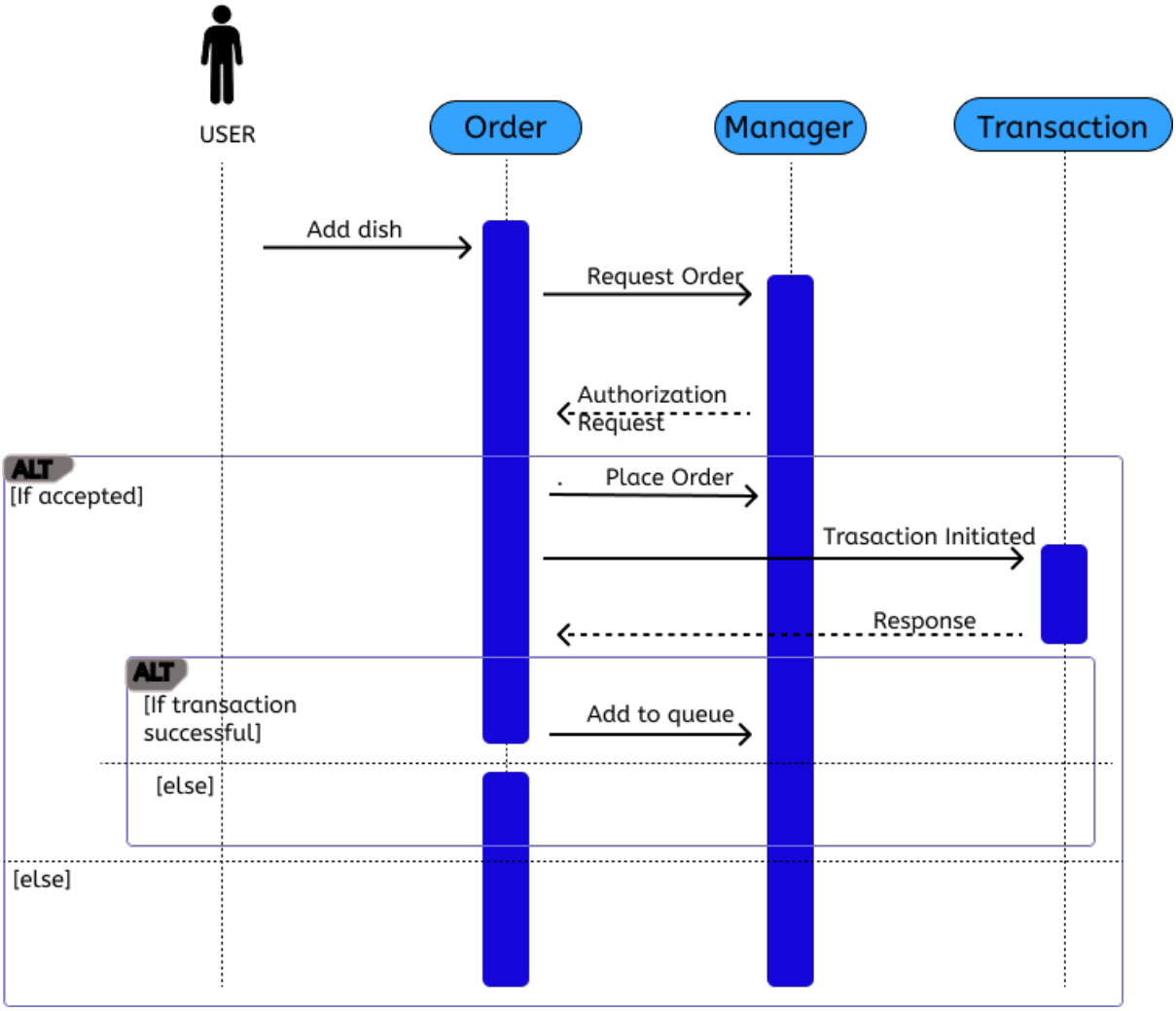
3.3.2 Login Sequence



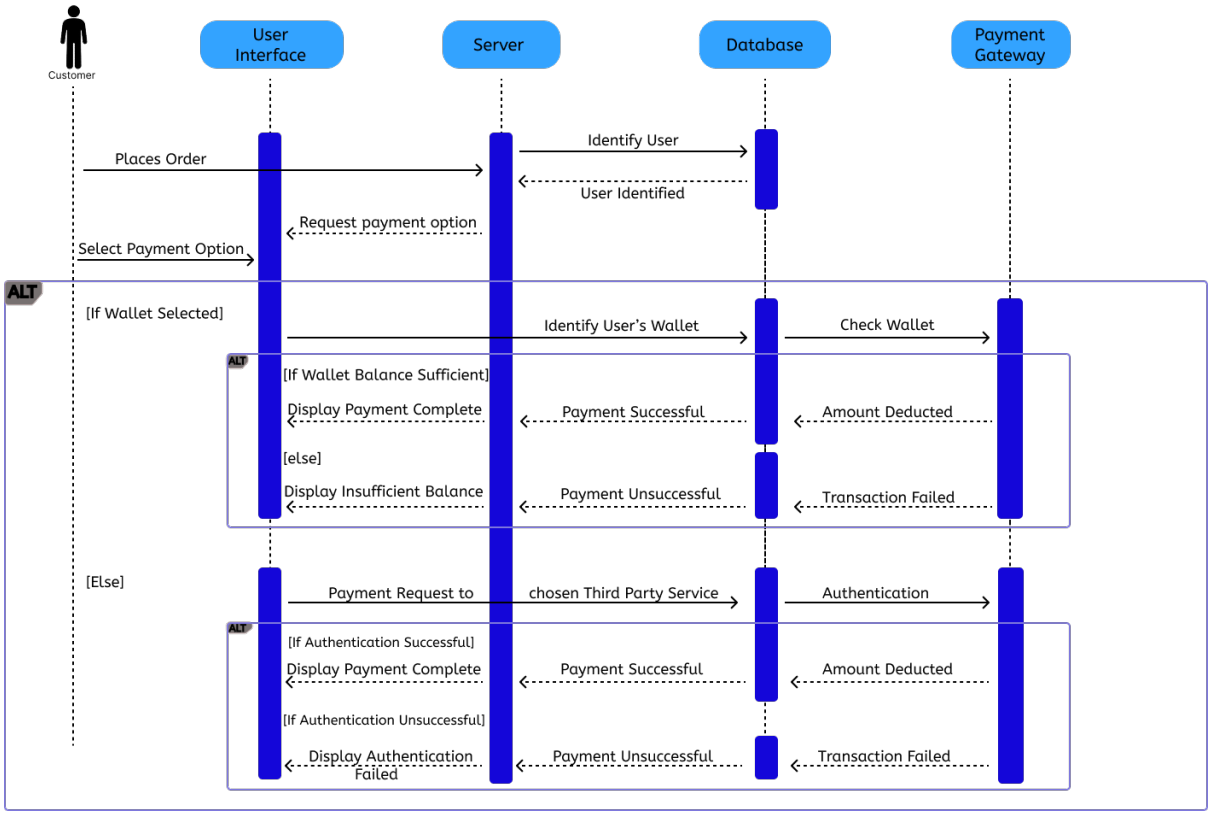
3.3.3 Reset Password Sequence



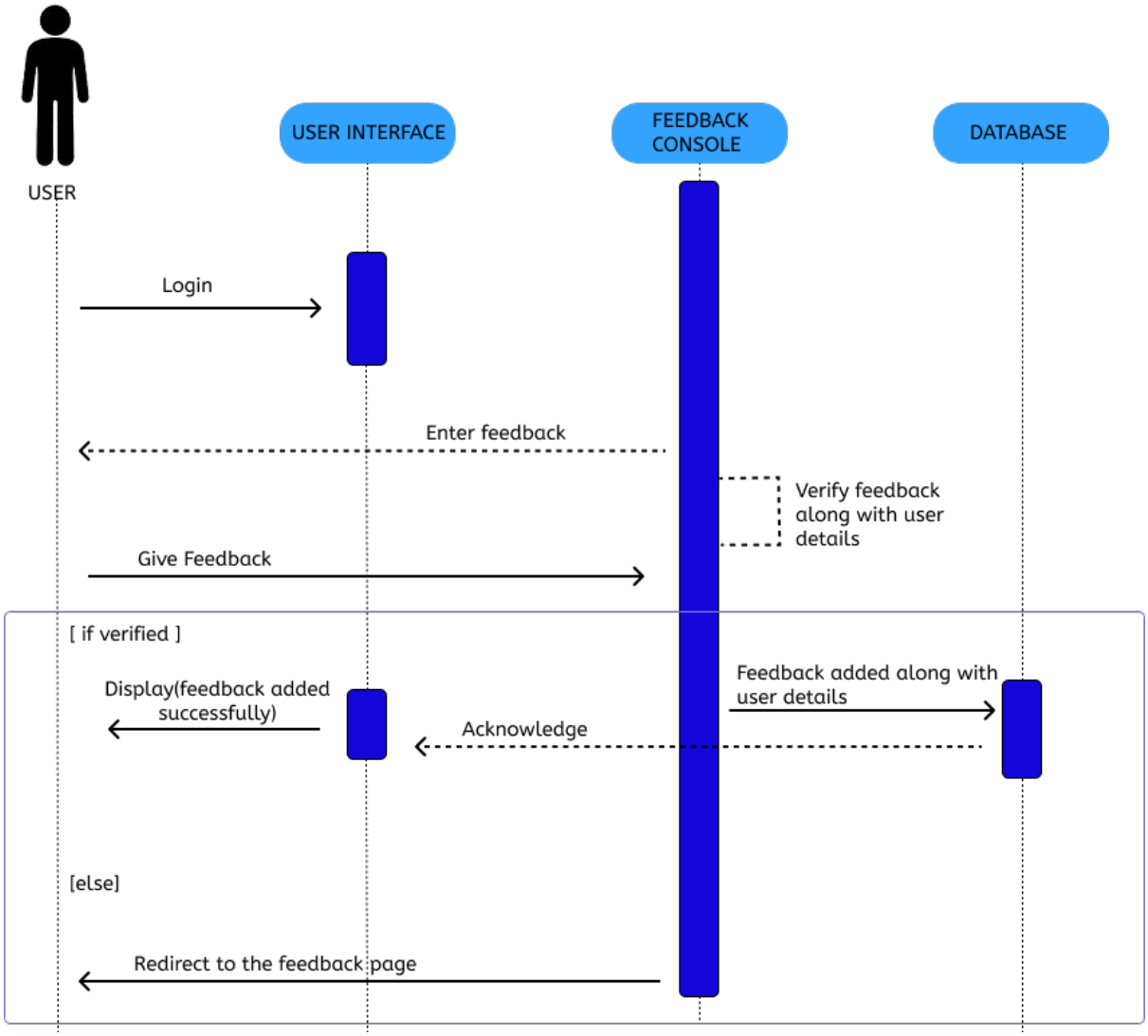
3.3.4 Food Ordering Sequence



3.3.5 Payment Sequence

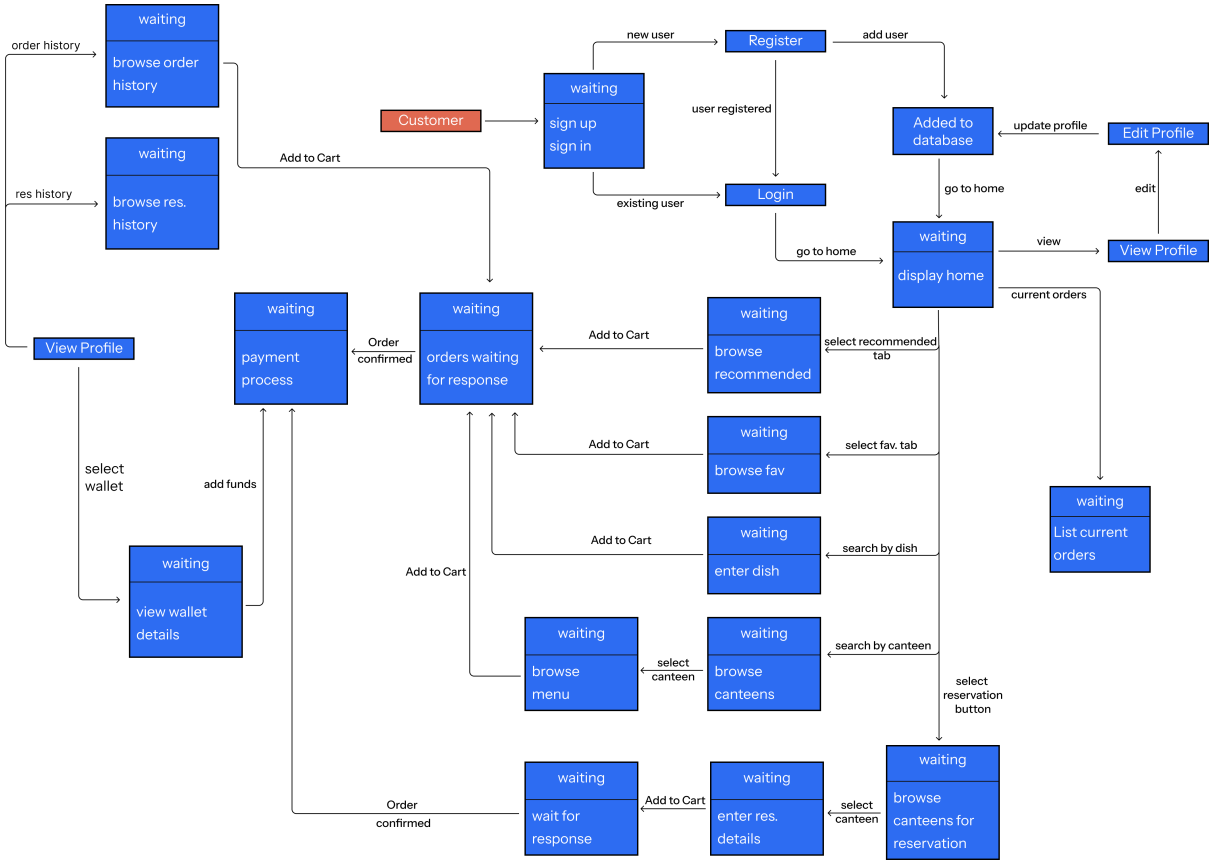


3.3.6 Feedback Sequence

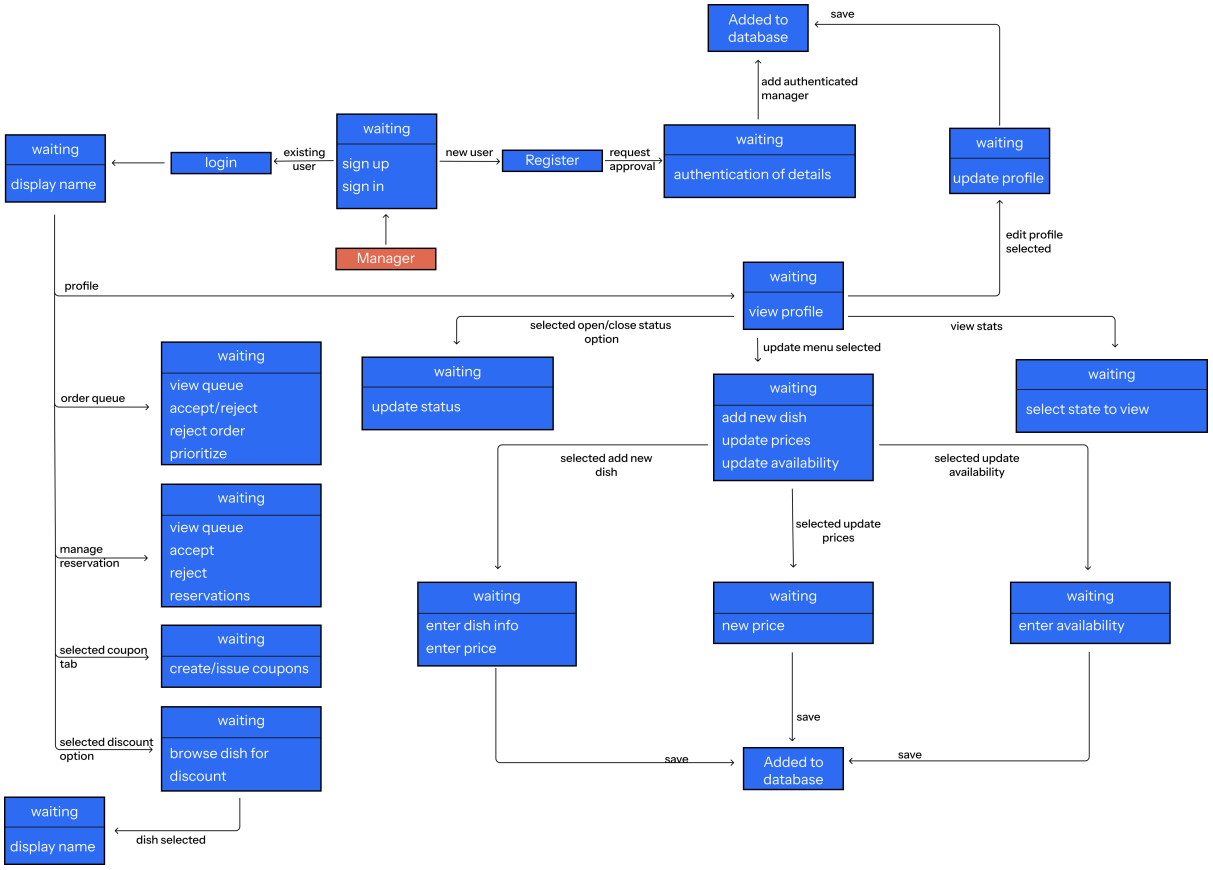


3.4 State Diagrams

3.4.1 Customer State Diagram



3.4.2 Canteen Manager State Diagram



4 Project Plan

4.1 Implementation and Testing Plan

The project follows a phased development approach with the following key stages:

Frontend Development

Designing and developing user interfaces. This includes:

Landing Page The landing page serves as the entry point for the website:

Register:

- New users (students) enter their institute email address.
- An OTP (One-Time Password) is sent to the provided email for verification.
- After successful verification, users are prompted to set a password for their account.
- Upon completion, users are automatically redirected to the login page.

Login:

- Existing users enter their email and password.
- The system verifies credentials and redirects to the homepage upon successful login.
- If the login attempt is unsuccessful, an error message appears, and users can retry.
- A ”Forgot Password” option allows users to reset their password via email verification.

Customer Homepage

Navigation bar The navigation bar provides easy access to various sections of the website:

Order Management:

- Displays a list of available canteens along with their real-time menu status.
- Users can select items with quantity and preferences.
- A table reservation system allows users to choose time slots.
- The system shows estimated preparation and wait times.

Personal Profile:

- Provides a link to the user’s personal profile page.
- Allows users to view and edit their personal information.
- Displays a complete order history.
- Manages user notification preferences.

Cart Interface:

- A platform to review selected items.

- Users can modify order specifications and update quantities.
- Users can process table reservations for their selected items.
- Special dietary requirements can be specified.

Search System:

- Enables search by canteen or dish name.
- Offers filters based on cuisine type and availability.
- Displays real-time table availability.
- Allows sorting by ratings or preparation time.

Canteen Homepage

Navigation bar The canteen homepage navigation bar offers control over various aspects:

Menu Control:

- Allows the update of menu items and availability status.
- Manages pricing and item descriptions.
- Sets preparation time estimates for menu items.
- Configures auto-accept preferences for orders.

Order Processing:

- Monitors the incoming order queue.
- Updates the status of orders in real-time.
- Manages table reservations.
- Handles special requests related to orders.

Profile Settings:

- Updates canteen profile information, including name and description.
- Configures operational parameters such as working hours and seating arrangements.
- Manages notification preferences for the canteen.

Backend Development

– The backend system implements several key functions to manage user interactions and order processing:

- `profile_update()`: Updates user or canteen profile information.
- `order()`: Processes new order requests from customers.
- `reservation()`: Handles table booking requests for users.
- `update_order()`: Updates existing orders with changes or additional requests.

- `request_order()`: Initiates the order processing cycle.
- `place_order()`: Finalizes the order placement and sends it to the kitchen.
- `update_order_status()`: Manages the lifecycle of orders (e.g., pending, in progress, completed).
- `update()`: Updates menu items and availability in real-time.
- `availability_update()`: Manages real-time stock status for ingredients and menu items.
- `manage_reservations()`: Manages table booking and availability.

Database Setup

The backend utilizes a single database to store various collections for the classes used across the system:

User Data:

- Instances of `UserProfile` and `LoginDetails` classes store user details, email addresses, and passwords.

Order Data:

- Instances of `Order` and `Menu` classes store order information and menu item details.

API Development

Establishing communication between frontend and backend.

- **Defining API Endpoints**: Identify the necessary endpoints for data exchange, such as user authentication, data retrieval, and updates.
- **Authentication & Authorization**: Implement security measures like API keys to ensure secure communication.
- **Middleware & Request Handling**: Use middleware for logging, error handling, request validation, and response formatting.
- **Versioning**: Maintain different API versions (e.g., v1, v2) to allow future upgrades without breaking existing functionality.

Integration and Testing

– Combining components and ensuring smooth functionality.

- **Frontend-Backend Integration** API calls from the frontend fetch or send data and render UI components accordingly.
- **Third-Party Services** Integrate external services like payment gateways (Stripe, Razorpay), authentication providers (Google OAuth), or cloud storage (AWS, Firebase).
- **Testing Strategies**

  Unit Testing: Validate individual functions or components using Jest, Mocha, or JUnit.

  Integration Testing: Ensure different modules work together using tools like Selenium, Cypress, or Postman for API testing.

  End-to-End (E2E) Testing: Simulate real-world user interactions across the entire system.

Security & Performance Testing

– Validating system security and optimizing performance.

- **Security Testing**
  - **Authentication & Authorization Testing**: Prevent unauthorized access by verifying proper role-based access control (RBAC).
  - **Input Validation**: Protect against SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
  - **Data Encryption**: Ensure data is encrypted at rest and in transit (SSL/TLS).
  - **Penetration Testing**: Simulate cyberattacks to find vulnerabilities using tools like OWASP ZAP or Burp Suite.
- **Performance Testing**
  - **Load Testing**: Measure how the system behaves under high traffic using Apache JMeter or k6.
  - **Stress Testing**: Push the system beyond its limits to identify breaking points.
  - **Latency Optimization**: Optimize database queries, caching (Redis, Memcached), and use a Content Delivery Network (CDN) for faster responses.

Deployment & Final Testing

– Deploying the final version and resolving last-minute issues.

- **Setting Up CI/CD Pipelines**: Automate deployment using tools like GitHub Actions, Jenkins, or GitLab CI/CD.
- **Infrastructure Setup**: Deploy on cloud platforms (AWS, Azure, GCP) or use containerization (Docker, Kubernetes).
- **Database Migration & Backup**: Ensure smooth transition of data and have rollback plans in case of failure.
- **Final Testing**:
  - **User Acceptance Testing (UAT)**: Validate that the product meets business requirements.
  - **Regression Testing**: Ensure recent changes don't break existing features.
  - **Monitoring & Logging**: Use tools like Prometheus, ELK Stack, or Datadog to track errors, performance, and system health post-deployment.

After successful deployment, ongoing maintenance and updates are necessary to keep the system secure and efficient.

4.2 Major Tasks and Dependencies

- **Team Structure**: The Members have been divided into Two Teams, A and B, each of 5 members.

  Team A: Team A Consists of Akshay, Arihant, Kshitij, Parnika and Suvid, responsible for Frontend Development.

  Team B: Team B consists of Dnyanesh, Nikhilesh, Rohit, Yashaswi and Yatharth, responsible for Backend and Database Development.

Task	Dependencies	Responsible Team(s)
Frontend Development		**TEAM A**
UI/UX Design	Design Finalization	
Implement Login & Authentication	UI/UX Design	
Home, Profile, Orders Page	UI/UX Design	
Canteen Dashboard UI	UI/UX Design	
Backend Development		**TEAM B**
Implement Authentication API	Database Schema Finalization	
Implement Order Management API	Backend Setup	
Payment Integration	Backend Setup	
Database Development		**TEAM B**
Set Up User & Order Database	Schema Finalization	
Implement Queries for Transactions	-	
Testing Phase		**Entire Team**
Unit Testing for Frontend Components	Completion of APIs/UI	
Backend Unit Testing	UI Completion	
Integration Testing	API Completion	
Performance & Security Testing	Backend + Frontend Done	
Functional Testing	Functional Testing Completed	
Deployment & Finalization		**Entire Team**
Deployment	All Testing Completed	

Table 1: Major Tasks and Dependencies

4.3 Gantt Chart

A **Gantt Chart** has been created to visually represent task execution timelines, ensuring efficient project tracking. The chart is included as a reference for scheduling and monitoring progress.

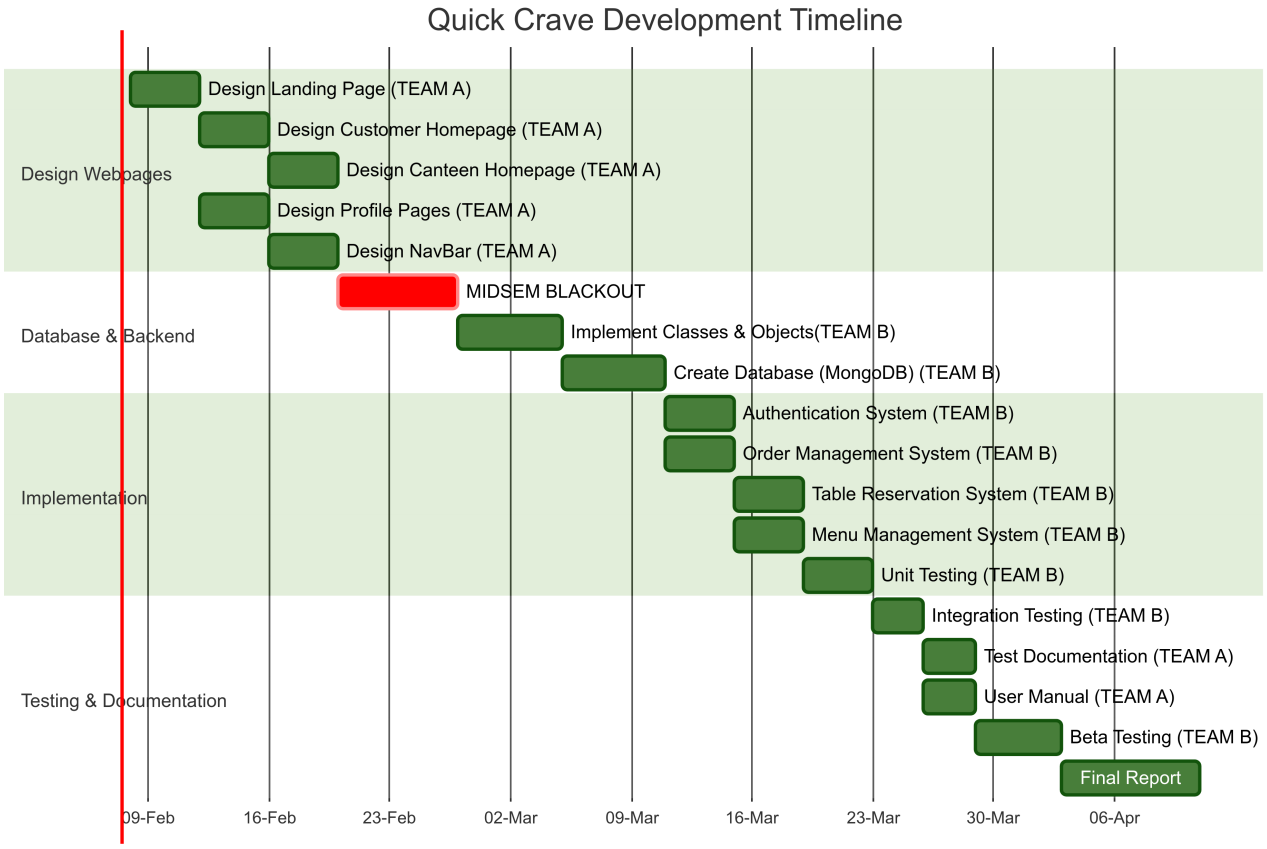


Figure 2: Quick Crave Gantt Chart

4.4 Communication

WhatsApp and Email are and will continue being used for intra-team communication

4.5 Code Collaboration

GitHub will be used to collaborate on code during the implementation phase

Appendix

Date	Team	Summary of the Meeting	Duration	Location
25th Jan	All	Commenced the initial stages of crafting the software design document.	45 min	Google Meet
31st Jan	All	Allocated subsections to each member of the group.	75 min	RM building
1st Feb	All	Met with the TA.	40 min	Google Meet
3rd Feb	C	Critically analysed various architecture models.	120 min	New Core Lab
5th Feb	C	Finalised and documented system architecture.	150 min	RM building
5th Feb	B	Discussed class diagrams and state diagrams.	150 min	RM building
6th Feb	B	Discussed sequence diagrams and use case diagrams.	180 min	RM building
6th Feb	A	Finalized Gantt Chart.	120 min	RM building
7th Feb	B	Frontend UI design discussion.	120 min	RM building
7th Feb	A	Finalized Context Design	120 min	RM building

Table 2: Summary of Meetings

Team	Members
A	Akshay Reddy, Suvid Goyal, Parnika Mittal
B	Dnyanesh Hemendra Shinde, Arihant Kumar, Yatharth Dangi, Nikhilesh Joshi, Kshitij Gupta
C	Yashaswi Raj, Rohit Somani

Table 3: Team Details