Software Implementation Document

for

Quick Crave

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 Vijaykumar 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*

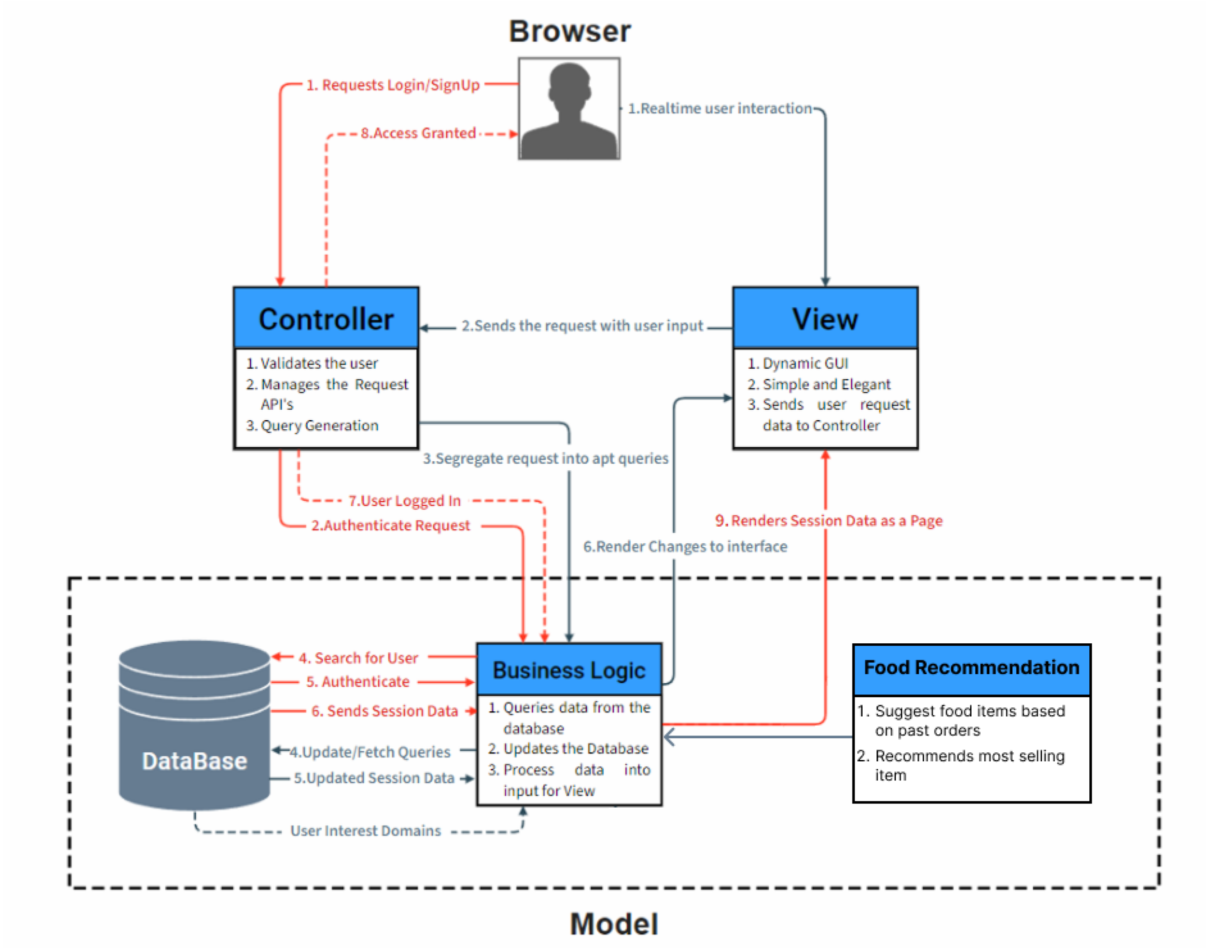
Submission Date: *January 24, 2025*

Contents

1 Implementation Details

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.


Browser
1. Requests Login/SignUp
8.Access Granted
1.Realtime user interaction
Controller
1. Validates the user
2. Manages the Request API's
3. Query Generation
2.Sends the request with user input
View
1. Dynamic GUI
2. Simple and Elegant
3. Sends user request data to Controller
3.Segregate request into apt queries
7.User Logged In
2.Authenticate Request
9.Renders Session Data as a Page
6.Render Changes to interface
4. Search for User
5. Authenticate
6. Sends Session Data
4.Update/Fetch Queries
5.Updated Session Data
DataBase
Business Logic
1. Queries data from the database
2. Updates the Database
3. Process data into input for View
Food Recommendation
1. Suggest food items based on past orders
2. Recommends most selling item
User Interest Domains
Model

1.1 MVC Architecture for QuickCrave

1.1.1 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 canteen menus, availability, and prices.
- Processing orders, including status updates and transaction handling.
- Interfacing with external services (e.g., ML models for recommendations, payment APIs).

We have used **PostgreSQL**, an **open-source object-relational database system** with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

These are the **main advantages/benefits** of PostgreSQL:

- **Open Source:** PostgreSQL source code is freely available under an open-source license. This allows us the freedom to use, modify, and implement it as per our business needs.
- **Ease of Learning:** To learn PostgreSQL, you don't need much training as it's easy to use.
- **Fault Tolerance:** Its write-ahead logging makes it a highly fault-tolerant database.
- **Low maintenance:** PostgreSQL requires minimal maintenance, making it a cost-effective and reliable choice for businesses.

We have used **Node.js**. Node.js is a powerful and efficient runtime environment for executing JavaScript applications. It is widely used for building scalable and high-performance applications.

- **Fast and Efficient:** Node.js uses the **V8 JavaScript engine**, making it extremely fast for executing code, especially for I/O-intensive operations.
- **Non-blocking I/O:** With its **asynchronous, event-driven architecture**, Node.js can handle multiple requests concurrently without blocking the system.
- **Single Programming Language:** Developers can use JavaScript for both frontend and backend, promoting seamless communication between components.
- **Extensive Package Ecosystem: The Node Package Manager (NPM)** offers access to thousands of reusable libraries and modules, accelerating development.
- **Scalable:** Node.js is well-suited for applications requiring real-time data processing, like chat applications, gaming platforms, or live-streaming services.
- **Cross-Platform:** It is compatible with multiple operating systems, including Windows, macOS, and Linux, ensuring flexibility in deployment.

1.1.2 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 canteen 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).

Our frontend was developed using **JSX** and **CSS**, leveraging **React** to create a dynamic and interactve user interface. **JSX(JavaScript XML)** allows us to write **HTML-like syntax** within **JavaScript**, making component creation more intuitive and readable.

JSX and CSS were preferred over traditional HTML and CSS file due to their:

- **Component-Based Structure:** JSX enables the creation of reusable UI components, reducing redundancy and improving code maintainability
- **Separation of Concerns:** Each page has its own JSX and CSS file, keeping structure and styles organized.
- **Dynamic Rendering:** JSX seamlessly integrates with React, allowing UI updates without requiring a full page reload.
- **Scoped Styling:** Maintaining separate CSS files for each JSX component prevents style conflicts and ensures a consistent look.

React Hooks were used to manage state and lifecycle events efficiently, eliminating the need for class components.

1.1.3 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.

We have used **Express.js** a popular backend framework for buiding web applications using **Node.js**. It simplifies application logic management and enhances the development process.

Advantages of Express.js:

- **Lightweight and Fast**: Express.js has a minimalistic core, providing only essential features, which makes it lightweight and efficient.
- **Easy Routing**: It offers a simple and intuitive routing system to handle different end points, making API development straightforward.
- **Middleware Support**: Express.js allows the use of middleware functions to process requests, handle errors, and add additional functionality.
- **Flexible**: It can be used to build single-page, multi- page, and RESTful web applications, adapting to a wide range of projects.
- **Seamless Integration**: Express.js integrates easily with databases, authentication services, and third-party APIs.

1.2 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.

1.3 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.

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.

1.4 Development and version control environment

We used **Git** as our version control system. It is the most commonly used version control system that is used for software development and version control tasks. Git is designed to keep track of modifications made to files, providing a comprehensive record of all changes made. The functionality permits users to revert to particular versions if needed. It also makes collaboration easier, allowing changes by multiple people to all to be merged into a single source.

We used **Github**, a web-based Git repository hosting service to manage our repositories and collaborate among ourselves.

1.5 Authentication and Authorization

We have implemented **authentication** by storing the email and password in our database, utilizing **cryptographic hashing** and **salting** techniques for password security to prevent any potential data exposure in the event of a data breach. To implement **authorization**, we have used **JWT(JSON Web Token)**.

2 Codebase

An organization has been setup on GitHub for storing and collaborating on the source code of this project **Quick Crave**

The repository contains the following files:

~~~
    ```C:.
|   .DS_Store
|   package.json
|   README.md
|   structure.txt
|
+---backend
|   |   .DS_Store
|   |   package-lock.json
|   |   package.json
|   |   server.js
|   |
|   +---controllers
|   |       .DS_Store
|   |       authController.js
|   |       canteenController.js
|   |       customerController.js
|   |
|   +---db
|   |       .DS_Store
|   |       index.js
|   |
|   +---middleware
|   |       authenticate.js
|   |       authenticateOTP.js
|   |
|   +---node_modules
|   |   [content not shown]
|   |
|   \---routes
|           auth.js
|           canteen.js
|           customer.js
|
\---frontend
    |   .gitignore
    |   package-lock.json
    |   package.json
    |   README.md
    |
    +---public
    |   |   favicon.ico
    |   |   index.html
    |   |   manifest.json
    |   |   robots.txt
    |   |
~~~

```
|   \---images
|           [image files]
|
\---src
    |   [root files]
    |
    +---pages
    |   |   [authentication pages]
    |   |
    |   +---Canteen
    |   |   |   [canteen manager pages]
    |   |   |
    |   |   +---[various canteen sections]
    |   |
    |   \---Customer
    |           [customer pages]
    |
    +---services
    |       canteenService.js
    |
    +---styles
    |   |   [root style files]
    |   |
    |   +---[various style directories]
    |
    \---temp
            [temporary files]```
```

3 Completeness

The ”Section 3: Specific Requirements” of Software Requirements Specification Document lists all the desired product functionality.

3.1 Functionalities that have been implemented

1. Customer Interface

- Customer Login
- Customer Signup
- Customer Forgot Password
- Customer OTP
- Customer Home
- Customer Profile
- Customer Food Delivery
- Customer Dineout
- Customer Reservation Booking
- Customer Order History
- Customer Favourites
- Customer Statistics
- Customer Cart

2. Canteen Manager Interface

- Canteen Login
- Canteen Signup
- Canteen Forgot Password
- Canteen Verification
- Canteen Home
- Canteen Manage Order Queue
- Canteen Manage Reservations
- Canteen create Discounts and Coupons
- Canteen Profile
- Canteen Statistics
- Canteen Menu Management

3.2 Non-Functional requirements that have been achieved

Performance Requirements:

- **Response Time for Simple Actions(Database Queries, UI Updates):** The system processes simple actions like fetching data and querying the database within 500 milliseconds
- **Concurrent Users Handling:** The system handles upto 2000 concurrent users during peaktimes.
- **System Throughput:** The system processes atleast 200 orders per minute during peak times.

Safety and Security:

- User Authentication and Access Control
- Data Protection and Privacy
- Safeguards against Data Loss

Software Quality Attributes:

- Usability
- Maintainability
- Interoperability
- Robustness
- Flexibility

3.3 Things we plan to update in our web app in future

- **Payment:** Enabling customers to make payments through various methods such as UPI, credit cards, and digital wallets, while updating transaction records and ensuring secure payment processing.

4 Group Log

Meeting notes and activities will be maintained here.
Team A: Akshay, Arihant, Kshitij, Parnika, Suvid, Yashaswi
Team B: Dnyanesh, Rohit
Team C: Nikhilesh, Yatharth

S.No	Team	From Date	To Date	Venue	Agenda
1	All	24/02/2025	28/02/2025	Online	We discussed the technologies we will be using and assigned specific topics for each of us to learn.
2	Team A	01/03/2025	04/03/2025	Hall 5	To get a rough idea of the frontend design, we started designing pages in Figma.
3	Team B	02/03/2025	06/03/2025	Hall 5	To get a rough idea of the canteen component of the backend, we started discussing plausible designs.
4	Team C	01/03/2025	05/03/2025	Hall 5	To get a rough idea of the customer component of the backend, we started discussing plausible designs.
5	Team A	05/03/2025	24/03/2025	RM and Online	Developed JSX files and corresponding CSS for customer and canteen manager web pages.
6	Team B	07/03/2025	24/03/2025	RM and Online	Developed the router and controller files for canteen side.
7	Team C	05/03/2025	24/03/2025	RM and Online	Developed the router and controller files for customer side.
8	All	25/03/2025	28/03/2025	RM	Integrated frontend with backend.

Table 1: Meeting Log