"Healthy Mealz" - An Online Home Food Application

A Project Report of Major Project Phase-II Submitted to
Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal
Towards Partial Fulfillment of the Degree Of
Bachelor of Technology in Computer Engineering

Guided By:

Mrs. Priyanka Bamne
Assistant Professor
Department of Computer Engineering.

Submitted By:

Dipti Sharma	0801CS201030
Goutam Patidar	0801CS201035
Harsh Saboo	0801CS201038
Manal Choudhary	0801CS201053
Sidharth Jain	0801CS201090

DEPARTMENT OF COMPUTER ENGINEERING
SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE, INDORE(M.P.)
2023-24

SHRI G. S. INSTITUTE OF TECHNOLOGY AND SCIENCE,
INDORE (M.P)

SESSION: 2023-24

RECOMMENDATION

The project report for Major Project Phase-II entitled *"**Healthy Mealz**"* submitted by: **0801CS201030 - Dipti Sharma, 0801CS201035 - Goutam Patidar**, **0801CS201038 - Harsh Saboo**, **0801CS201053 - Manal Choudhary**, **0801CS201090 - Sidharth Jain** Students of Bachelor of Technology IV year in the session 2023-2024, towards fulfillment of the degree of **B.Tech. in Computer Engineering** of Rajiv Gandhi Proudyogiki VishwaVidhyalaya, Bhopal is a satisfactory account of their work towards CO44498: Major Project (Phase II) and is recommended for the award of degree.

Mrs. Priyanka Bamne
Project Guide
Assistant Professor
Department of Computer Engineering

Dr Vandan Tewari
Head of Department
Department of Computer Engineering

Dean(Academics)
S.G.S.I.T.S. Indore

SHRI G. S. INSTITUTE OF TECHNOLOGY AND SCIENCE,
INDORE (M.P)

CERTIFICATE

The project report for CO44498: Major Project Phase-II entitled *"**Healthy Mealz**"* submitted by: **0801CS201030 - Dipti Sharma, 0801CS201035 - Goutam Patidar**, **0801CS201038 - Harsh Saboo**, **0801CS201053 - Manal Choudhary**, **0801CS201090 - Sidharth Jain** Students of B.Tech. IV year in the session 2023-2024, towards partial fulfillment of the degree of **Bachelor of Technology in Computer Engineerin**g of Rajiv Gandhi Proudyogiki VishwaVidhyalaya, Bhopal is a satisfactory account of their work.

Internal Examiner: External Examiner:

Date:

SHRI G. S. INSTITUTE OF TECHNOLOGY AND SCIENCE,
INDORE (M.P)

DECLARATION

We **0801CS201030 - Dipti Sharma, 0801CS201035 - Goutam Patidar**, **0801CS201038 - Harsh Saboo**, **0801CS201053 - Manal Choudhary**, **0801CS201090 - Sidharth Jain,** hereby declare that the work presented in the B.Tech. Major project report has been carried out by us. We further declare that the work submitted for the award of the degree doesn't contain any part of the work which has been submitted for the award of any degree either in this university or any other University without proper citation.

Date:

0801CS201030 - Dipti Sharma

0801CS201035 - Goutam Patidar

0801CS201038 - Harsh Saboo

0801CS201053 - Manal Choudhary

0801CS201090 - Sidharth Jain

ACKNOWLEDGEMENT

We express our profound sense of gratitude to our project guide **Mrs. Priyanka Bamne** and who had advised us to do project work entitled "***Healthy Mealz***". Their continuous support and motivation always made us deliver our best. Presence of such guides have always been an amazing experience, which is also a valuable gift for an engineer to progress in his/her life.

We are also grateful to **Dr. Vandan Tewari**, Head of Department of Computer Engineering and **Dr. Rakesh Saxena**, Director, S.G.S.I.T.S Indore, for providing us with numerous facilities and academic environment during the course of study.

We sincerely wish to express our gratitude to all the members of staff of Department of Computer Engineering, S.G.S.I.T.S Indore, who have extended their cooperation at all times and have contributed in their own way in developing the project.

The successful completion of the project is not an individual effort. It is an outcome of the cumulative effort of a number of people, each having their own importance to the objective. We express love and respect towards our parents and the entire family member who are our strength in everything we do. We are thankful to them for their constant support and motivation.

With a blend of gratitude, pleasure and great satisfaction we convey our indebtedness to all those who have directly or indirectly contributed to the successful completion of our project work.

0801CS201030 - Dipti Sharma _____

0801CS201035 - Goutam Patidar _____

0801CS201038 - Harsh Saboo _____

0801CS201053 - Manal Choudhary _____

0801CS201090 - Sidharth Jain _____

ABSTRACT

Food is an integral part of the survival strategy that human civilization has inculcated to adapt and the 21st century folks have disrupted the way in which humans interact with their food. Digital transformation today is like the "World Wide Web" or Internet of the late 90's and tech-giants have learnt to not follow the "*re-inventing the wheel*" formula. Simple innovations have led to surprisingly huge market-caps for investors and deal-breakers for customers.

The Online Home food is for ordering food from messes around. It is an application that is intended to provide complete solutions for messes as well as customers through a single gateway using the internet. It will enable messes to set-up online, allowing customers to browse through the list of messes and order online without having to visit the mess physically. The administration module will enable a system administrator to approve and reject requests for new messes and maintain various lists of messes.

Key features encompass an intuitive user interface facilitating seamless browsing and ordering, coupled with an administrative module enabling efficient management of mess registrations and essential databases. Serving as a pivotal intermediary, our application fosters streamlined communication between users and mess platforms, enhancing accessibility and convenience in the food procurement process.

Our application will work as an interface between users and a mess platform to communicate. a holistic solution bridging the divide between messes and consumers through a unified online platform. Powered by the PERN stack technology (PostgreSQL, Express, React, Node.js), our application empowers messes to seamlessly transition into the digital realm, offering customers the convenience to explore, select, and order food online sans physical constraints.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Preamble

In the midst of burgeoning job opportunities across the nation, many individuals are drawn to bustling metropolises such as Mumbai, Pune, Hyderabad, and Delhi in pursuit of their career aspirations. However, the excitement of moving to a new city is often tempered by the practical challenge of managing expenses. Among these expenses, rent and food reign supreme, shaping the trajectory of one's financial stability and the pursuit of their dreams. While cooking at home may seem like a viable option, the demands of time and energy often render it impractical and exhausting. It is in this scenario that tiffin services emerge as a saving grace, offering a convenient and efficient solution to the perennial dilemma of securing wholesome meals amidst the hustle and bustle of urban life. With tiffin services at hand, individuals can alleviate the burden of meal preparation and focus wholeheartedly on realizing their professional ambitions, thus embarking on their journey in a new city with greater ease and confidence.

1.2 Need of the Project

Locating Tiffin Centres and Mess Facilities: Many individuals, especially those who are new to a city or have busy schedules, struggle to find convenient and reliable options for meals. By assisting users in locating nearby tiffin centres and mess facilities, the project simplifies the process of accessing affordable and nutritious food within their locality. This addresses the immediate need for convenient dining solutions, particularly for individuals seeking hassle-free meal options.

Connecting Users with Home-Based Kitchens: In addition to traditional tiffin centres and mess facilities, the project also bridges the gap between users and individuals operating kitchens from their homes or from tiffin centres. This innovative approach not only expands the range of available food options but also fosters connections within local communities. By facilitating these connections, the project empowers home-based cooks to showcase their culinary talents while providing users with the opportunity to enjoy homemade meals that evoke the comfort and flavours of home. This aspect of the project enhances social engagement and supports local entrepreneurship, thereby enriching the overall dining experience for users.

Optimized Delivery with Fuel Cost Savings: In addition to helping users locate tiffin centres, mess facilities, and home-based kitchens, the project also offers optimized delivery services. By leveraging

advanced routing algorithms, the platform minimizes delivery distances and travel times, thereby reducing fuel consumption and associated costs. This environmentally conscious approach not only benefits the users by ensuring prompt and efficient delivery of meals but also contributes to sustainability efforts by minimizing carbon emissions. By integrating this feature, the project aligns with the broader goal of promoting eco-friendly practices while enhancing the overall user experience.

Multiple hop Delivery for Efficiency: Furthermore, the project implements a multiple hop delivery system to optimize efficiency in food delivery logistics. By orchestrating a series of interconnected delivery stops, the platform maximizes the utilization of delivery resources while minimizing travel distances. This innovative approach allows for the consolidation of multiple orders into a single delivery route, reducing both time and fuel consumption per order. Through multiple hop delivery, the project not only enhances operational efficiency but also contributes to environmental sustainability by mitigating the carbon footprint associated with food delivery. By offering this feature, the project underscores its commitment to resource optimization and eco-friendly practices while providing users with timely and cost-effective delivery services.

1.3 Problem Statement

Navigating expenses in a new city poses a significant challenge, particularly when it comes to managing food costs. Amidst a plethora of options, finding a tiffin service that aligns with our daily dietary requirements becomes a daunting task. The abundance of choices makes it challenging to identify one that offers the taste akin to home-cooked meals at a reasonable price point. Consequently, we often find ourselves grappling with uncertainty regarding the availability and suitability of tiffin centres that meet our culinary preferences. This problem underscores the need for a solution that streamlines the process of discovering tiffin services tailored to individual tastes, ensuring both affordability and the comfort of home-cooked flavours.

1.4 Objectives

User Convenience: The primary objective is to enhance user convenience by providing a centralized platform or application where users can easily locate tiffin centres and mess facilities in their vicinity. This aims to address the challenge of finding suitable and convenient food options, especially for individuals living away from home or those with busy schedules.

Location-Based Services: Implementing location-based services is crucial to ensure users can find relevant options based on their current location or preferred area. This involves integrating GPS or

location tracking functionalities into the platform/application to accurately identify nearby tiffin centres and mess facilities.

Comprehensive Database: Building a comprehensive database of tiffin centres, mess facilities, and home-based kitchens is essential. This database should include details such as location, menu options, pricing, operating hours, contact information, and user reviews/ratings to help users make informed decisions.

User Engagement: Promoting user engagement is vital for the success of the platform/application. This can be achieved through features such as user reviews, ratings, comments, and interactive maps, allowing users to share their experiences, recommendations, and feedback with others in the community.

Connectivity with Home-Based Kitchens: Facilitating connections between users and individuals running kitchens from their homes is an additional objective. This can offer users a wider range of food options, including homemade meals, and provide opportunities for local cooks and homemakers to showcase their culinary skills and earn additional income.

Quality Assurance: Maintaining quality standards across listed tiffin centres, mess facilities, and home-based kitchens is crucial for user satisfaction and trust. Implementing measures such as verification checks, hygiene inspections, and adherence to food safety regulations can help ensure consistent quality and reliability.

Promotion and Growth: Promoting the platform/application to target users and expanding its reach across different localities is essential for its growth and sustainability. This involves strategic marketing efforts, partnerships with local businesses, and leveraging social media platforms to increase awareness and user adoption.

Feedback Mechanism: Establishing a feedback mechanism is necessary to continuously improve the platform/application based on user suggestions and evolving needs. This can involve regular surveys, feedback forms, and direct communication channels to gather insights and implement enhancements accordingly.

Monetization Strategy: Developing a monetization strategy to sustain the project financially is also an objective. This can include options such as subscription plans for premium features, advertising partnerships with local businesses, commission-based models for referrals, or transaction fees for online orders, depending on the business model adopted.

1.5 Organization of the Report

Introduction introduces the context of the project giving an insight of the motivation, need and scope of the project. Background Study describes the literature survey done to study the concepts of the existing technologies. It also elucidates the area of concern where the **"*Healthy Mealz*"** is of great importance and also describes in detail the background survey conducted to understand the software components required to design the system. It also includes the tools and technologies used in developing the system. Literature review contains the written after reading several research papers on relevant topics. It also analyses the progress done till date and the improvements made during the project. Analysis deals with the functional and the non-functional requirements, system components, use case diagram, use case scenario, data flow diagram and feasibility study. Design focuses on details comprising the basic architecture of the system, activity diagram. Implementation details the programming code of the project. Testing and results are the record of all the tests done on the system and its subsequent results. Conclusion concludes and summarizes the work done. The appendix and references are listed in a separate section.

Chapter 2

BACKGROUND STUDY

2.1 Area of Concern

There are many areas of concern for “Online Home Food” like Food safety: As a food delivery service, you must prioritize food safety and ensure that the food you deliver to your customers is safe to eat. This includes maintaining proper hygiene and sanitation practices in your kitchen, handling food properly to avoid contamination, and ensuring that food is stored at the correct temperatures to prevent spoilage.

Another one is Packaging and delivery: The packaging and delivery of your food are crucial to the success of your business. You should choose packaging that keeps the food fresh and hot during transport, and ensure that the food is delivered promptly to avoid delays that could compromise its quality. You also need to consider the logistics of delivery, including the distance you can deliver, delivery fees, and how you will manage the delivery process.

Menu selection and variety is also essential for online food delivery service. You should offer a range of dishes that cater to different tastes and dietary requirements, and ensure that your menu is updated regularly to keep customers interested.

2.2 Tools and Technologies

In this section, description of all the tools- software or hardware, is given. In depth details of their functionality and market price is also included in this section.

- Frontend: React.js, Tailwind, Capacitor, Google Map API, Geolocation API
- Backend: Node.js, Express.js, Razor Pay payment gateway API
- Database: Postgres, Elephant SQL

2.2.1: React

React is a framework that employs Webpack to automatically compile React, JSX, and ES6 code while handling CSS file prefixes. React is a JavaScript - based UI development library. Although, react is a

library rather than a language, it is widely used in web development. The library first appeared in May 2013 and is now one of the most commonly used frontend libraries for web development.

2.2.2: Postgres

PostgreSQL is a powerful, open-source object-relational database system. It has more than 15 years of active development phase and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. PostgreSQL is an open-source relational database management system (DBMS) developed by a worldwide team of volunteers. PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge.

2.2.3: Express.JS

Express.js is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to Node.js HTTP objects and facilitates the rendering of dynamic HTTP objects.

2.2.4: Google Maps API

The Google Maps API is one of those clever bits of Google technology that helps you take the power of Google Maps and put it directly on your own site. It lets you add relevant content that is useful to your visitors and customize the look and feel of the map to fit with the style of your site.

2.2.5: Capacitor

A cross-platform native runtime for web apps. Capacitor is an open-source native runtime for building Web Native apps. Create cross-platform iOS, Android, and Progressive Web Apps with JavaScript, HTML, and CSS. Capacitor delivers the same cross-platform benefits, but with a more modern approach to app development, taking advantage of the latest Web APIs and native platform capabilities.

2.2.6: Geolocation API

The Geolocation API is a service that accepts an HTTPS request with the cell tower and Wi-fi access points that a mobile client can detect. It returns latitude/longitude coordinates and a radius indicating the accuracy of the result for each valid input.

Chapter 3

LITERATURE REVIEW

3.1 Inception

The inception of this project defines the primary purpose of the “Healthy Mealz” (e.g., connecting home cooks with customers, providing healthy meal options, promoting local cuisines). Understand user requirements, preferences, and challenges related to home-cooked meals, delivery services, dietary needs, etc. Determine the appropriate tech stack (platforms, databases, frameworks, APIs) for developing the application. Build interactive prototypes to visualize user interactions and refine user experience (UX). Analyse trends, consumer preferences, and gaps in the market.

3.2 Tools Required for Implementation

- Code Editor: A code editor is essential for writing and editing code. Popular choices include Android Studio, Visual Studio Code, Atom, Sublime Text, or WebStorm.
- Node.JS: Node.js is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Ryan Dahl developed it in 2009, and its latest iteration, version 15.14, was released in April 2021. Developers use Node.js to create server-side web applications, and it is perfect for data-intensive applications since it uses an asynchronous, event-driven model.
- Version Control System: Version control is crucial for tracking changes to the codebase, collaborating with a team, and maintaining a history of code revisions. Git is a widely used version control system, and tools like Git Kraken, SourceTree, or the command-line interface can help with managing Git repositories.
- Elephant SQL: Elephant SQL is a PostgreSQL database hosting service. Elephant SQL will manage administrative tasks of PostgreSQL, such as installation, upgrades to latest stable version and backup handling. Elephant SQL is also integrated to several cloud application platforms (also known as PaaS). With a click of a button your database is provisioned in the same data centre as your application is hosted, and is ready to be used immediately.

- Integrated Development Environment (IDE): While not strictly necessary, an IDE can enhance productivity and provide additional features for development. IDEs like Visual Studio Code, Android Studio, or Atom has extensions/plugins available for XML and Java development.

3.3 Summary

"Healthy Mealz" revolutionizes the management of dietary needs, especially in bustling urban settings, by offering a comprehensive solution that prioritizes convenience, affordability, and health-conscious choices. By seamlessly connecting users with nearby tiffin centres, mess facilities, and home-based kitchens, the app simplifies the process of sourcing nutritious and delicious meals tailored to individual preferences. Its innovative multiple hop delivery system enhances efficiency, minimizing environmental impact and reducing fuel costs for timely deliveries. With "Healthy Mealz," users can confidently embark on their city journey, assured that their dietary requirements are met, allowing them to focus on their aspirations without compromise.

Beyond catering to user needs, "Healthy Mealz" serves as a platform for economic empowerment by facilitating connections between users and home-based entrepreneurs. Through this feature, the app creates job opportunities for individuals looking to start businesses from home, fostering community engagement and supporting local economies. By enjoying convenient and nutritious meals through "Healthy Mealz," users not only satisfy their dietary needs but also contribute to the growth and empowerment of home-based cooks, nurturing a sense of shared prosperity within the community.

Chapter 4
ANALYSIS

4.1 Detailed Problem Statement

Whenever we move to a new city managing expenses is a crucial task. We spend most of the money on food. We face lots of problems in figuring out tiffin centres who can fit our daily food needs. With plenty of tiffin service available, it gets difficult to choose the one that fits and of course we do not have any idea about the number of options we have who can give us the taste which resembles the taste of home cooked food at an affordable cost.

4.2 Requirement Analysis

This section covers various functions that are the main promises of the system. These functionalities constitute the minimum offerings of a complete system. There are two types of requirements for any system functional and non-functional requirement.

4.2.1 Functional Requirements

In software engineering and systems engineering, a functional requirement defines a function of a system or its component, where a function is described as a specification of behaviour between outputs and inputs.

1. Login/Registration Procedure

1.1 Sign up / Sign in

a) Description and Priority

- Customer shall be able to login or can perform registration (if not registered) to enter the menu dashboard. Registration is compulsory for delivery man and Mess owner to use privileged functionalities available for delivery boy and Mess owner
- Also, for customer login can only be done using mobile number after OTP (one time password) verification so that convenient for user and for us(admin) also so that only verified customer can enter into our system.

1.2 Functional Requirement

a) Sign Up

- User: Customer, Delivery man, Mess owner
- Input: Detail about the user
- Output: Successfully Signed-Up.

b) Login

- User: Customer, Delivery man, Mess owner
- Input: Login credential that is mobile number or userid and password
- Output: Successfully logged in.

2. Selecting a mess

2.1 Description and Priority

- Customers can choose any mess according to their convenience depending on the distance to the mess from their locality also depending on the rating of the mess also. Customers can choose mess according to the particular type of meals the customer is interested in.

2.2 Functional Requirements

a) Selecting a category

- User: user can select a mess for booking or ordering according to his choice
- Input: The customer can choose the mess from the list of mess provided in homepage
- Output: All the meals available on the mess will be listed after choosing the mess.

3. Update Daily Tokens

3.1 Description and Priority

- This will allow the customer to cancel the meal if he doesn't want order the update daily tokens are zero.

3.2 Functional requirement

- User: Customer
- Input: Select meal which customer want to be cancelled
- Output: At the owner's end, he got the message about cancelling the meal.

4. Payment Options

4.1 Description and Priority

- This will allow the customer to select the option of payment and with help of this, if he doesn't have cash at that time, he can choose one of the options example net banking, debit, credit card and UPI payments

4.2 Functional requirement

a) Payment

1. User: Customer
2. Input: Select the mode of payment
3. Output: Mess Owner gets the money in the account or cash.

5. Profile Detail

5.1 Description and Priority

- This will allow the users to know about the mess owner or customer. This will list all the details related to corresponding stakeholders.

5.2 Functional requirement

a) View Profile

1. User: Customer, Mess owner, Delivery man
2. Input: Click on view profile of the Owner, customer
3. Output: Get all the details of the worker, and customer.

6. Mess owner's Functional Requirements

- Mess shall be able to receive orders from customers
- Mess shall be able to view the orders which has been ordered by customers.

- Mess shall be able accept or cancel order depends upon the order received and availability of order
- Mess shall be able to assign delivery boy to deliver order
- Mess shall be able to receive acknowledgement from Delivery Boy
- Mess shall be able to view the payment
- Mess shall able to receive notifications once order delivered
- Mess shall able to give offers
- Mess owner can update the address by Google Map, Geolocation API, Dragging Marker.

7. Delivery Person's Functional Requirements

The following are the identified functional Delivery boy requirements that directly relate to the entire today's Meal system.

Requirements Description

- Delivery boy shall able to receive incoming request from Mess
- Delivery boy shall able to acknowledge for the request
- Delivery boy shall view all the orders of particular mess allotted.
- Delivery boy shall able to reach the Mess and check with order details
- Delivery boy shall able to receive customer details from Mess.
- Delivery boy shall able to pick up and deliver order to customer.
- Delivery boy shall able to receive rating from customers.
- Delivery boy shall able to receive notification for Mess details, payment details of customer.
- Delivery boy shall able to receive payment if order is done with cash.

8. Admin's Functional Requirements

- Admin shall able to Manage providers
- Admin shall able to Mange accounts
- Admin shall to manage Mess details

- Admin shall to manage Mess rate and reviews
- Admin shall able to manage Delivery boy details
- Admin shall able to display top rated Mess
- Admin shall able to display offers for specific Mess
- Admin shall able to contact between Delivery boy and Mess to replace or cancel order if items are not available.
- Admin can update the address by Google Map, Geolocation API, Dragging Marker.

9. Notification

9.1 Description and Priority

- In this customer can see the notifications when the remaining tokens of customer is less than 10 for mess.

9.2 Functional Requirements

- User: Customer
- Input: Click on the Notification icon.
- Output: Customer will notify about the mess tokens.

10. Tracking

10.1 Description and Priority

- In this delivery boy can track its location and get route which is optional for delivery.

10.2 Functional Requirements

- User: Deliver Boy
- Input: The delivery boy must be logged in to their account.
- Output: Delivery Boy can track its location.

11. Update Mess Menu

11.1 Description and Priority

- The mess owner can add a mess menu by interactive UI just by clicking text.

11.2 Functional Requirements

- User: Mess Operator
- Input: The Mess Operator must log in to their account and have the access to the menu.
- Output: The menu of the mess is successfully updated to include healthy meal options.

12. Search Mess

12.1 Description and Priority

- The Customer can see the result for messes that are under a 3km radius.

12.2 Functional Requirements

- User: Customer
- Input: The Customer must be logged in to their account and click on the search button.
- Output: The Customer can see the messes successfully.

13. View Subscribed Mess and Rating

13.1 Description and Priority

- The Customer can view their subscribed mess and also rate the messes.

13.2 Functional Requirement

- User: Customer
- Input: Click on the mess profile section
- Output: Customer can view their subscribed mess and rate them from there.

14. Cart

14.1 Description and Priority

- The Customer can view their subscribed messes in the cart.

14.2 Functional Requirement

- User: Customer
- Input: Click on the cart icon.
- Output: Customer can view their messes in the cart.

15. Navigation

15.1 Description and Priority

- The Customer can navigate with the google map.

15.2 Functional Requirement

- User: Delivery boy
- Input: Click on the start navigation with google map button.
- Output: Customer can navigate the directions through google map.

16. Order Status

16.1 Description and Priority

- The Customer can track their order status.

16.2 Functional Requirement

- User: Customer
- Input: The Customer have to go to the profile section.
- Output: The Customer can track their order status.

4.2.2 Non-Functional Requirement

This section specifies the required system quality factors that are not related to specific functional requirements documented in the use case module. These requirements are always meant to be fulfilled.

a) Performance

- System should be efficient so that it won't get hung up if heavy traffic of order is placed. Performance should be fast.
- System should not take more than 10 sec to reload any page of the software.

b) Compatibility and Security Requirements

- The system shall be able to use the app in different platforms like different versions of OS/Mobiles.
- The system shall be able to do the authentication process for login and payment through bankcard.
- The user shall able to do payment with secured bank payment mode

- The system shall be able to do encryption and decryption of data for password which is given by the user for login.

c) **Safety and Reliability Requirements**

- Data in the database of system should not loss or damage
- The system shall be capable of restoring itself to its previous state in the event of failure (e.g. a system crash or power loss).
- The system shall be able to display a menu at all times to facilitate manual order taking should the need arise.

d) Maintainability

- A commercial database is used for maintaining the database and the application server takes care of the site.

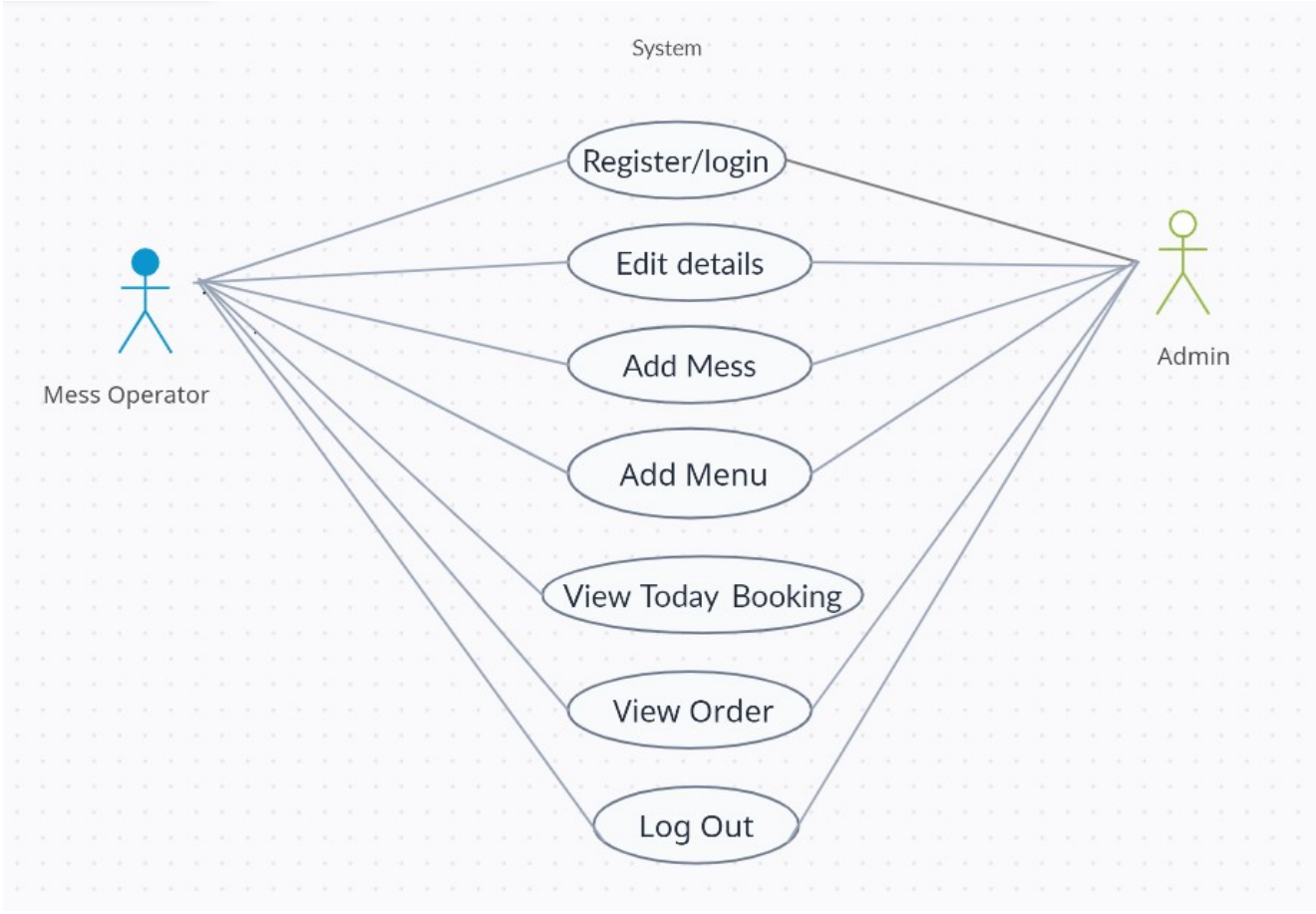
e) Usability

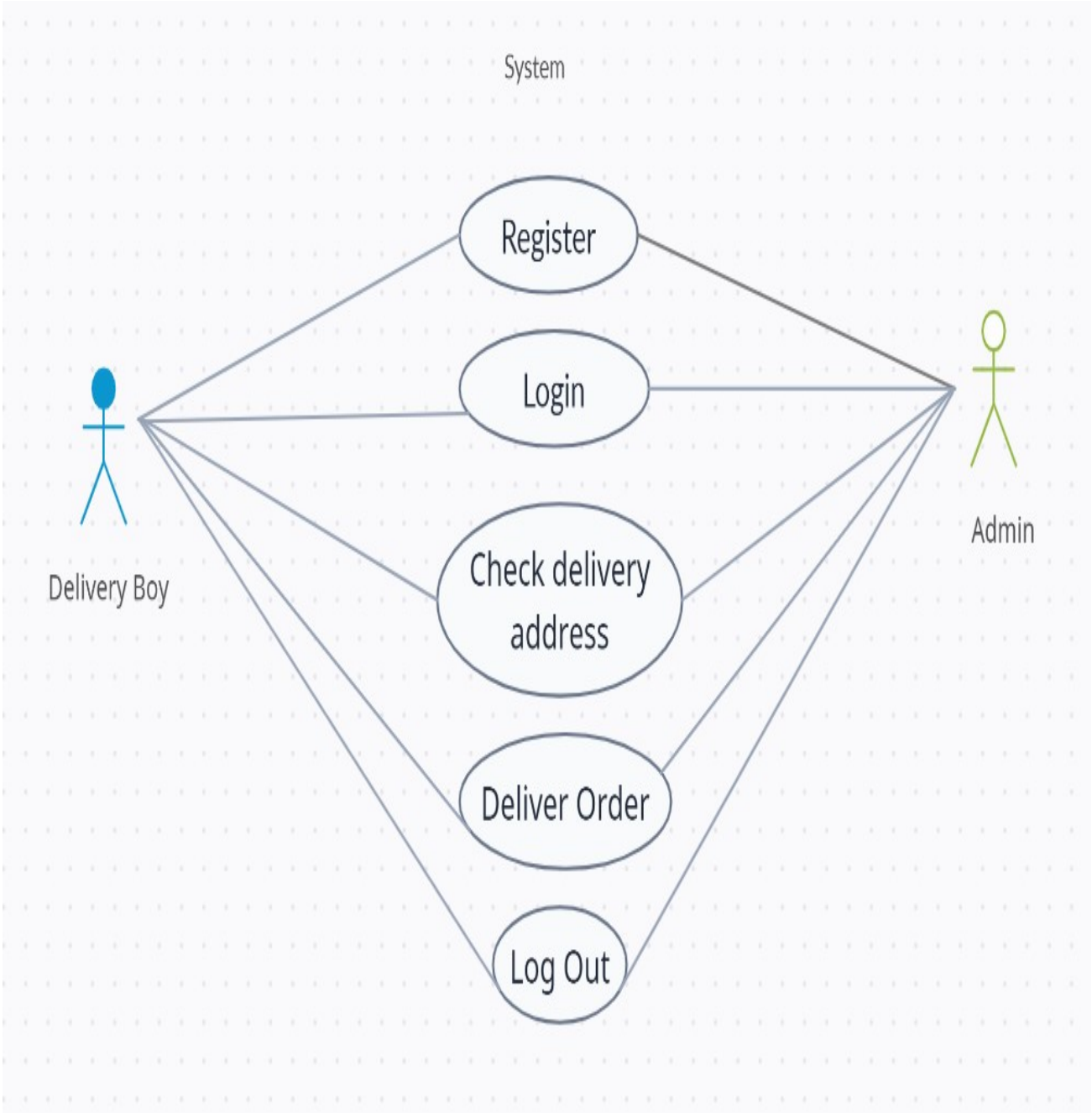
- System will be GUI based therefore anybody can use it easily.

4.3 Use Case Analysis

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

Use case diagram represented by Figure. 4.1 illustrates major functionalities.


System
Register/login
Edit details
Add Mess
Add Menu
View Today Booking
View Order
Log Out
Mess Operator
Admin


System
Register
Login
Check delivery address
Deliver Order
Log Out
Delivery Boy
Admin

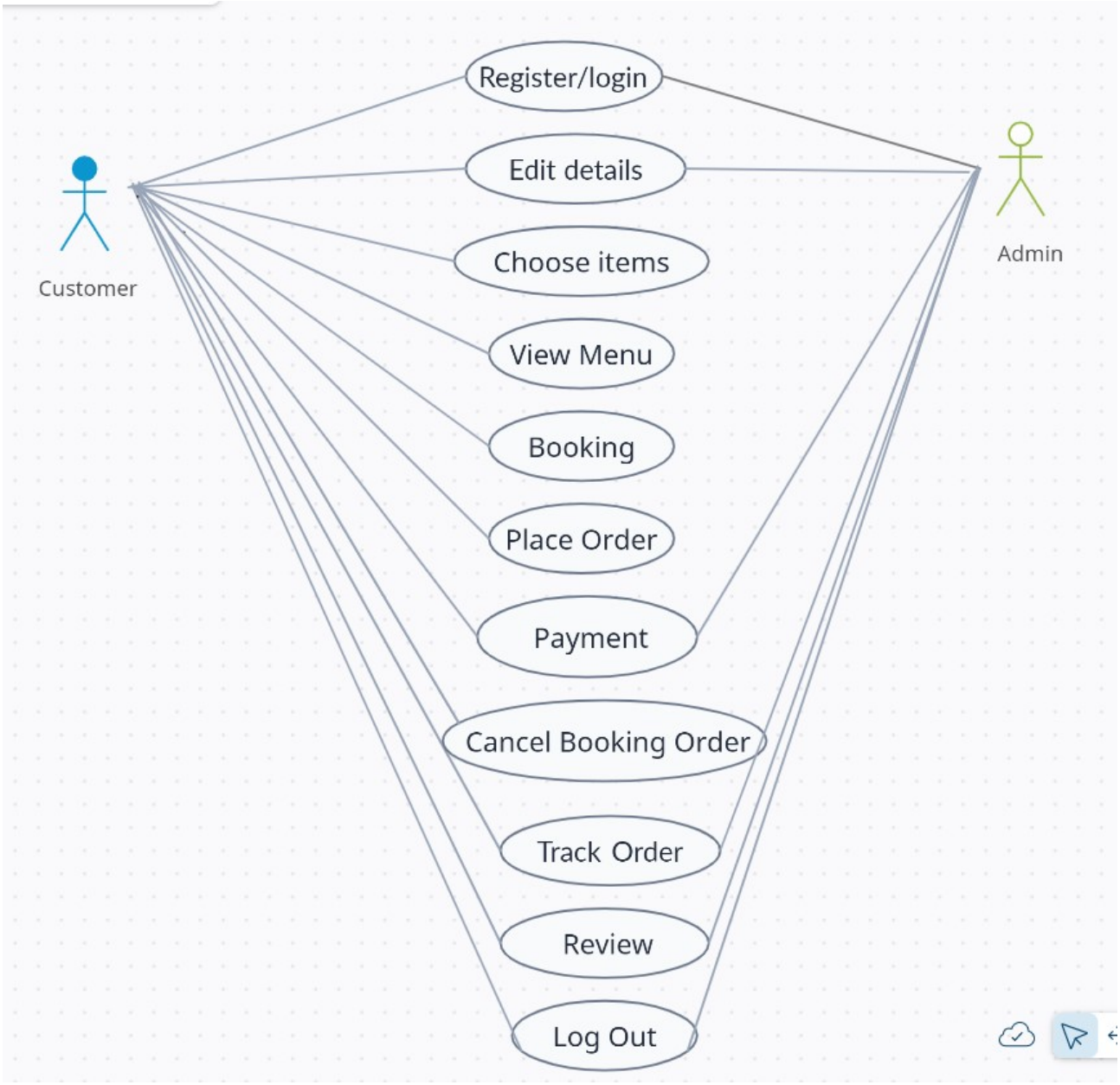


Figure 4.1: Use Case Diagram of Healthy Mealz

(I) Actors

- Customer
- Mess Operator
- Delivery boy

(II) **Roles and Responsibilities**

Role: Application

Responsibility:

- To provide an interface between users and mess.
- Help customers to order food from the preferred mess.

Role: User

Responsibility:

- Find a mess and place an order.

(III) **Use Cases**

- **Customer Registration**

  Actor: Customer

  Precondition: The customer does not have an account.

  Basic Flow: The customer intending to get a tiffin service does not have an existing account, initially. The system asks for basic profile information like phone number, and password for registering a new account. The customer enters the information and after validation, the account is created.

  Post Condition: The customer is registered successfully.

- **Mess registration**

  Actor: Mess operator

  Precondition: The mess operator does not have an account.

  Basic Flow: The mess operator intending to provide a tiffin service does not have an existing account, initially. The system asks for basic profile information like phone number, and password for registering a new account. The mess operator enters the information and after validation, the account is created.

  Post Condition: The mess is registered successfully.

- **Delivery boy registration**

  Actor: Delivery boy

  Precondition: The delivery boy does not have an account.

Basic Flow: The delivery boy intending to provide a tiffin service does not have an existing account, initially. The system asks for basic profile information like phone number, and password for registering a new account. The delivery boy enters the information and after validation, the account is created.

Post Condition: The delivery boy is registered successfully.

- **Login**

  Actor: Customer/ Mess operator / Delivery boy

  Precondition: The user is already registered.

  Basic Flow: This use case starts when an individual is not logged in. The system asks for login credentials and after validation, logs the individual in and redirects to the home page.

  Postcondition: The user logs in successfully.

- **Edit details**

  Actor: Customer/ Mess operator

  Precondition: The user is already registered, logged in and is on the edit details page.

  Basic Flow: The user can edit the profile information from here, such as address, phone no., etc.

  Postcondition: The user details are modified successfully.

- **Add mess**

  Actor: Mess operator

  Precondition: The mess operator is already registered, logged in and is on the add mess page.

  Basic Flow: The mess operator enters basic information about the mess, such as name, address, subscription fee, etc. of the mess.

  Postcondition: The mess is registered successfully.

- **Add menu**

Actor: Mess operator

Precondition: The mess operator is already registered, logged in and has added a mess. He is also on the add menu page.

Basic Flow: The mess operator enters the various items available in the tiffin, along with the quantity.

Postcondition: The menu gets added successfully.

- **View today booking**

  Actor: Mess operator

  Precondition: The mess operator is already registered, logged in and has added a mess.

  Basic Flow: The mess operator can navigate to the bookings page, and check today's tiffin orders.

  Postcondition: The mess owner can prepare food accordingly.

- **View order**

  Actor: Mess operator

  Precondition: The mess operator is already registered, logged in and has a mess.

  Basic Flow: The mess operator can view existing orders, subscription numbers, etc.

  Postcondition: The mess operator gets informed about the orders and expected revenue.

- **Check delivery address**

  Actor: Delivery boy

  Precondition: The delivery boy is already registered, logged in and is ready for delivery.

  Basic Flow: The delivery boy can check the delivery address of an order, to help him through navigation.

  Postcondition: The delivery boy gets the navigation details.

- **Deliver order**

Actor: Delivery boy

Precondition: The delivery boy is already registered, logged in and out for delivery.

Basic Flow: When the delivery boy delivers an order, he clicks on the deliver order button on the delivery page, and the order is considered delivered by the system.

Postcondition: The order gets delivered.

- **Choose items**

  Actor: Customer

  Precondition: The customer is already registered and logged in.

  Basic Flow: The customer can choose various items available in the tiffin.

  Postcondition: The items get chosen.

- **View menu**

  Actor: Customer

  Precondition: The customer is already registered and logged in.

  Basic Flow: The customer can view the menu of various messes, to choose the tiffin of their choice.

- **Booking**

  Actor: Customer

  Precondition: The customer is already registered and logged in.

  Basic Flow: The customer can choose various messes according to their choice. Once ready, they can book any one of the messes.

  Postcondition: The customer goes on the place order page.

- **Notification**

  Actor: Customer

Precondition: After login/signup customer can see the notification.

Basic Flow: When the remaining tokens of the customer are less than 10 for the mess, then the customer will get the notification.

Postcondition: The customer can click on the notification icon to see the notifications.

- **Place order**

  Actor: Customer

  Precondition: The customer is already registered, logged in and is on the place order page.

  Basic Flow: The customer has to confirm the basic details, delivery address, subscription fee. Once confirmed, the system redirects the customer to the payment page.

  Alternate flow: The customer goes back to the view mess page.

  Postcondition: The customer is directed to the payment page.

- **Payment**

  Actor: Customer

  Precondition: The customer is logged in. They chose a mess, confirmed the basic details and is on the payment page.

  Basic Flow: The customer chooses a payment method according to their choice. The system redirects them to the selected payment method. The customer inputs the payment credentials and gets redirected to the confirmation page.

  Alternate flow: The payment is not successful, for any reason, and the customer gets redirected to a payment failed page.

  Postcondition: For the basic flow, Payment will be done.

- **Cancel booking order**

  Actor: Customer

  Precondition: The customer is already registered, logged in and is on the cancel booking order page.

Basic Flow: The customer goes to the cancel booking order page, and cancels the booking for the day.

Postcondition: The tiffin gets cancelled for the day.

- **Track order**

  Actor: Customer

  Precondition: The customer is logged in, and already has a mess subscription.

  Basic Flow: The customer gets to track the status of the tiffin delivery: Preparing, out for delivery, delivered.

  Postcondition: The customer is directed to the payment page.

- **Review**

  Actor: Customer

  Precondition: The customer has a mess subscription and got at least one tiffin delivery.

  Basic Flow: After one tiffin gets delivered to the customer, the customer can rate and review the mess.

  Postcondition: The review gets added to the mess details page, in the reviews section.

- **Logout**

  Actor: Customer/ Mess Operator/ Delivery boy

  Precondition: The user is already registered, logged in.

  Basic Flow: The user goes on to the profile page and presses the logout button, and confirms it.

  Postcondition: The user is logged out.

- **Searching**

  Actor: Customer

  Precondition: The user must have a basic understanding of how to navigate a search interface.

Basic Flow: The user can search through mess, menu and location.

Postcondition: The user may proceed to prepare the selected meal or save it for future reference.

- **Update Mess Menu**

  Actor: Mess Operator

  Precondition: The Mess Operator must have access to the menu.

  Basic Flow: Mess owner can add mess menu by interactive UI just by clicking text.

  Postcondition: The menu of the mess is successfully updated to include healthy meal options.

- **Update Address**

  Actor: Customer, Mess Owner

  Precondition: The user must be logged in to their account.

  Basic Flow: The user and mess owner can update the address by using Google Maps and the customer and mess owner can mark a marker on the map and use Geolocation API to update the address.

  Postcondition: All ordered meal deliveries will be sent to the newly updated address.

- **Delivery Request**

  Actor: Mess Operator, Delivery boy

  Precondition: The Mess Operator, Delivery boy must be logged in to their account

  Basic Flow: The Mess Operator can send request to delivery boy to connect for Pickup and delivery.

  Postcondition: The Mess Operator, Delivery boy are connected and delivery boy can make Delivery for mess operator.

- **Cancel Meal Options**

  Actor: Customer

  Precondition: The Customer must be logged in to their account.

  Basic Flow: This will allow the customer to cancel the meal(set daily token to zero) if he doesn't want to order or book the same.

  Postcondition: The Customer can cancel the meal.

- **Search Mess**

Actor: Customer

Precondition: The Customer must be logged in to their account.

Basic Flow: The Customer can see the result for messes that are under a 3km radius.

Postcondition: The Customer can see the list of messes successfully.

- **Update Daily Tokens**

  Actor: Customer

  Precondition: The Customer must be logged in to their account.

  Basic Flow: The Customer can update their daily tokens and if they don't want order the daily tokens are zero.

  Postcondition: The Customer can see the updated daily tokens.

- **Tracking**

  Actor: Delivery Boy

  Precondition: The delivery boy must be log in to their account.

  Basic Flow: In this delivery boy can track its location and get the route which is optional for delivery.

  Postcondition: The delivery boy can track its location.

- **Navigation**

  Actor: Delivery Boy

  Precondition: The delivery boy must be log in to their account.

  Basic Flow: In this delivery boy can start navigating with the google map.

  Postcondition: The delivery boy can get the direction.

4.4 Future Implementation

4.4.1 Real Time Tracking

For implementing real-time tracking in our application, we propose integrating Firebase for backend services and the Google Maps API for mapping functionalities. By leveraging Firebase's real-time database capabilities, we can seamlessly update the location of user orders as they progress through the

delivery process. This allows users to track their orders in real-time, providing them with accurate and up-to-date information on the status and location of their deliveries.

Furthermore, integrating the Google Maps API enables us to visually represent the order's journey on a map interface, enhancing the user experience with intuitive and interactive tracking features. Users can easily visualize the route, estimated time of arrival, and any potential delays, empowering them with greater transparency and control over their deliveries.

In summary, by combining Firebase's real-time capabilities with the robust mapping functionalities of the Google Maps API, we can create a seamless and user-friendly real-time tracking system. This will not only enhance the user experience but also instill confidence and satisfaction in our service by providing users with accurate and transparent order tracking capabilities.

4.4.2 Cron Job

In our future project scope, we plan to implement an Automated Token Deduction Cronjob to streamline and automate token deduction operations on a daily basis. This proactive approach aims to enhance efficiency, accuracy, and user experience within our platform.

4.4.3 Machine Learning

By integrating machine learning into our platform, we aim to elevate the food discovery process, offering users a seamless and personalized experience. Through intelligent recommendations and intuitive search capabilities, we will empower users to discover new culinary delights and tiffin centers tailored to their tastes, ultimately enhancing satisfaction and engagement with our platform.

To achieve this, we will employ sophisticated machine learning models that analyze user behavior, preferences, and historical data. These models will continuously learn and adapt, ensuring that recommendations are tailored to each user's unique tastes and requirements. Through collaborative filtering and content-based filtering techniques, we can accurately predict food preferences and suggest relevant options that align with users' preferences.

Additionally, natural language processing (NLP) algorithms will enhance our search functionality, enabling users to find their desired dishes and tiffin centers with ease. By understanding user queries in natural language and accounting for synonyms, context, and user intent, our search feature will deliver more precise and intuitive results.

4.5 System Requirements

Resources are the means that are used to achieve project objectives. The main grouping of resources needed in this system is HARDWARE and SOFTWARE requirements.

(I) LIST OF HARDWARE USED

- Laptop

(II) LIST OF SOFTWARE USED

- Postgres, Elephant SQL
- React.js, Node.js
- Express.js
- Capacitor
- Google Map Platform
- Git

Chapter 5

DESIGN

5.1 System Architecture

A system architecture or systems architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system. Figure 5.1 is the Architecture Diagram of the System.

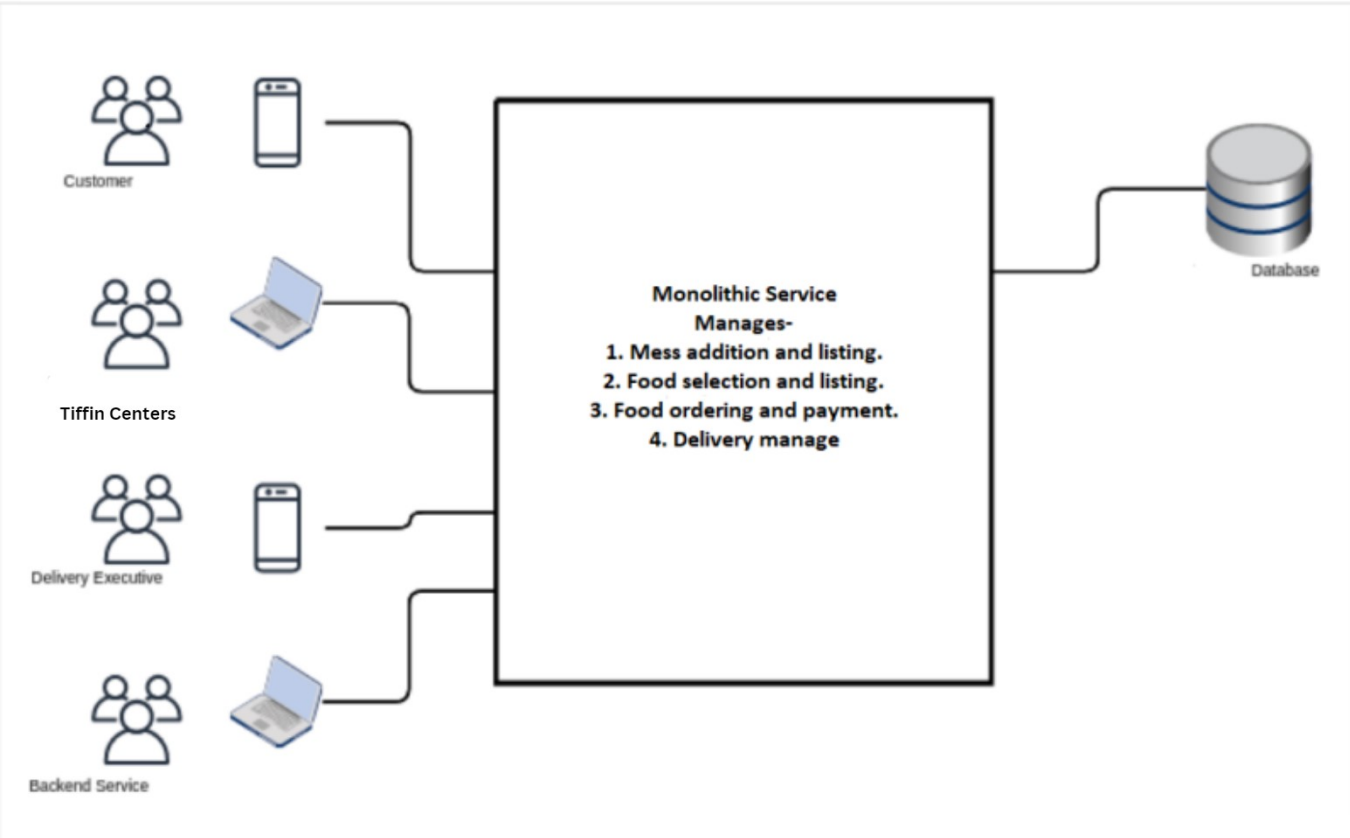
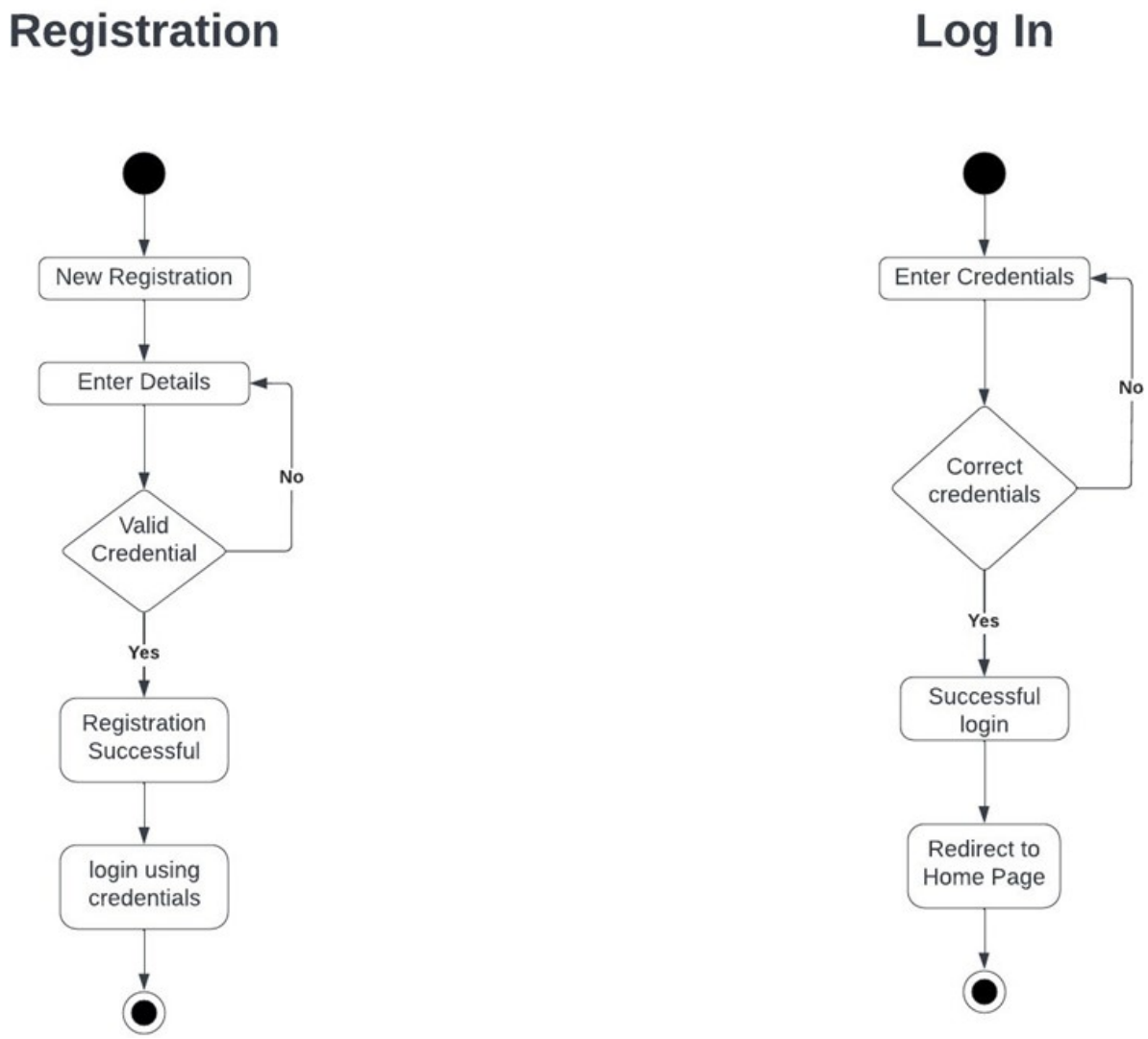


Figure 5.1: Architecture diagram of the System

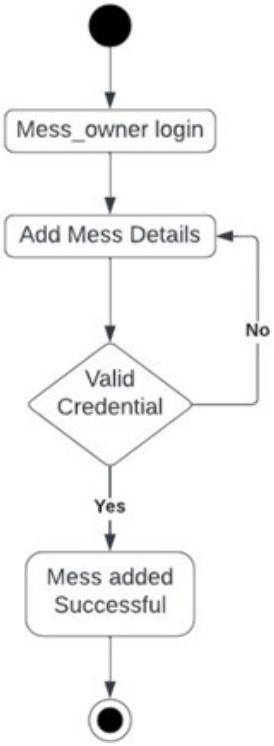
5.2 Activity Diagram

Activity diagram is used to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another.

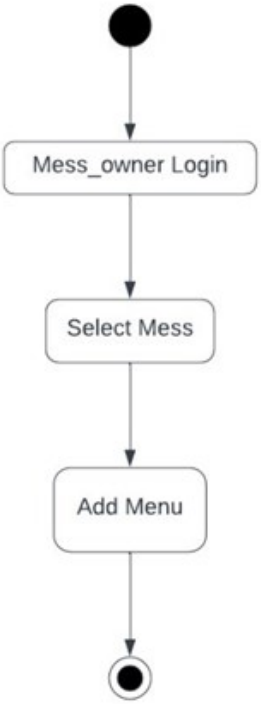
Figure 5.2 is the Activity Diagram of the System. This figure shows the flow of activities categorizing them on the basis of the role of the user performing the activity.



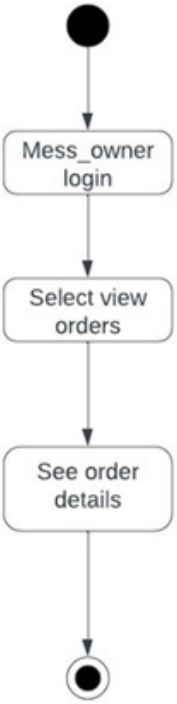
Add Mess



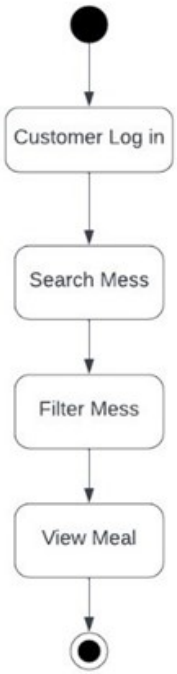
Add Meal



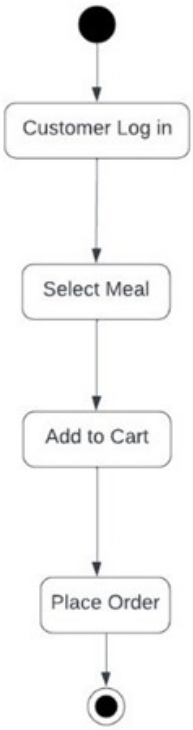
View Orders Details



View Meal



Place Order



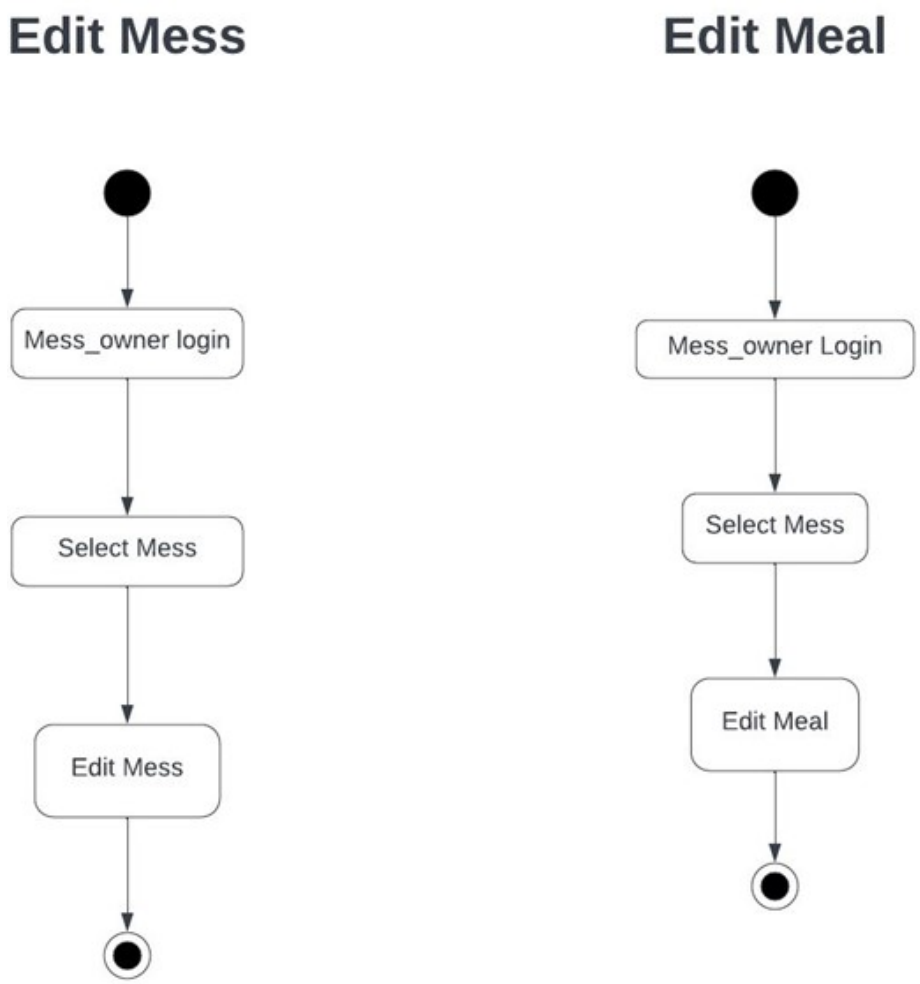
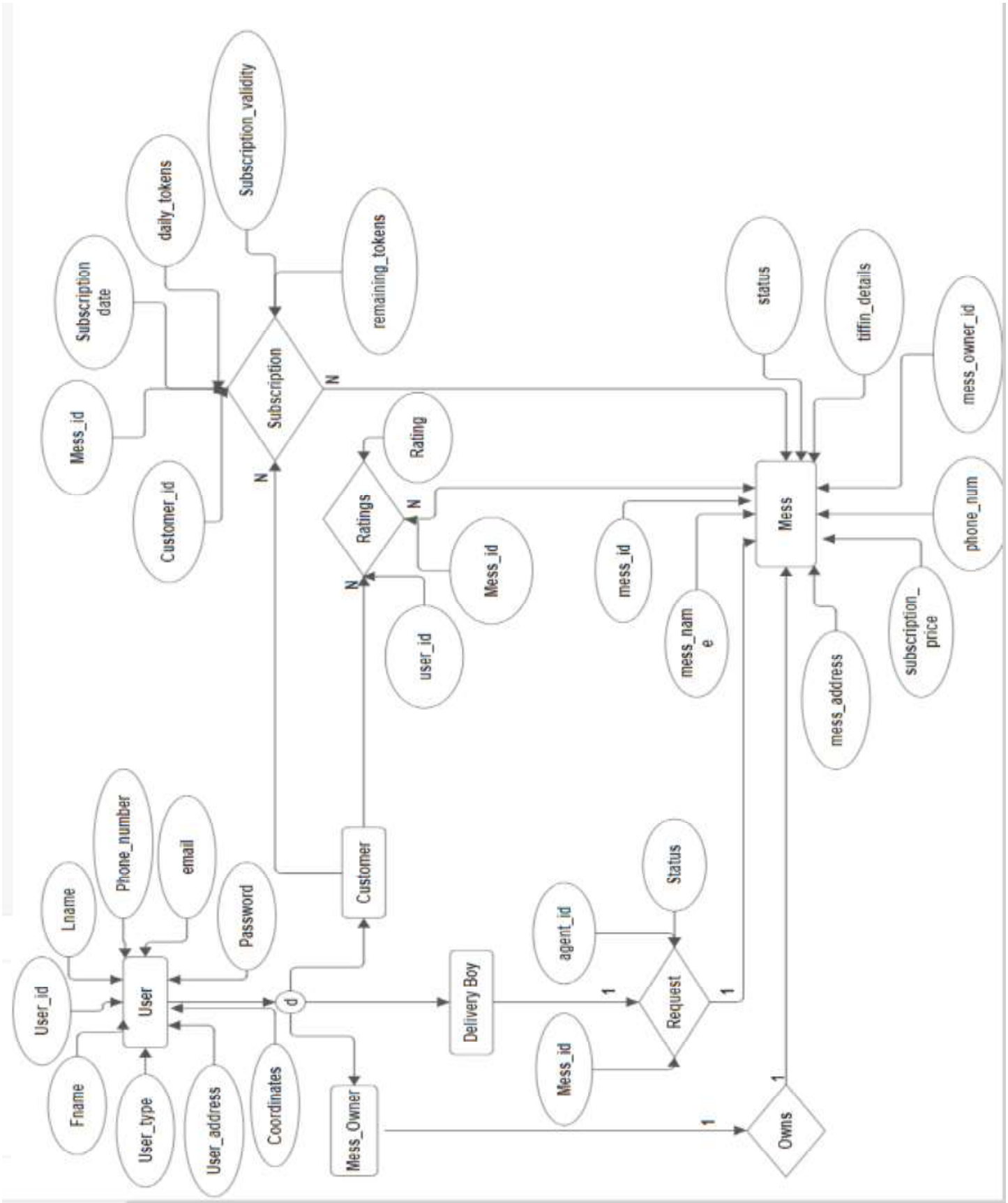
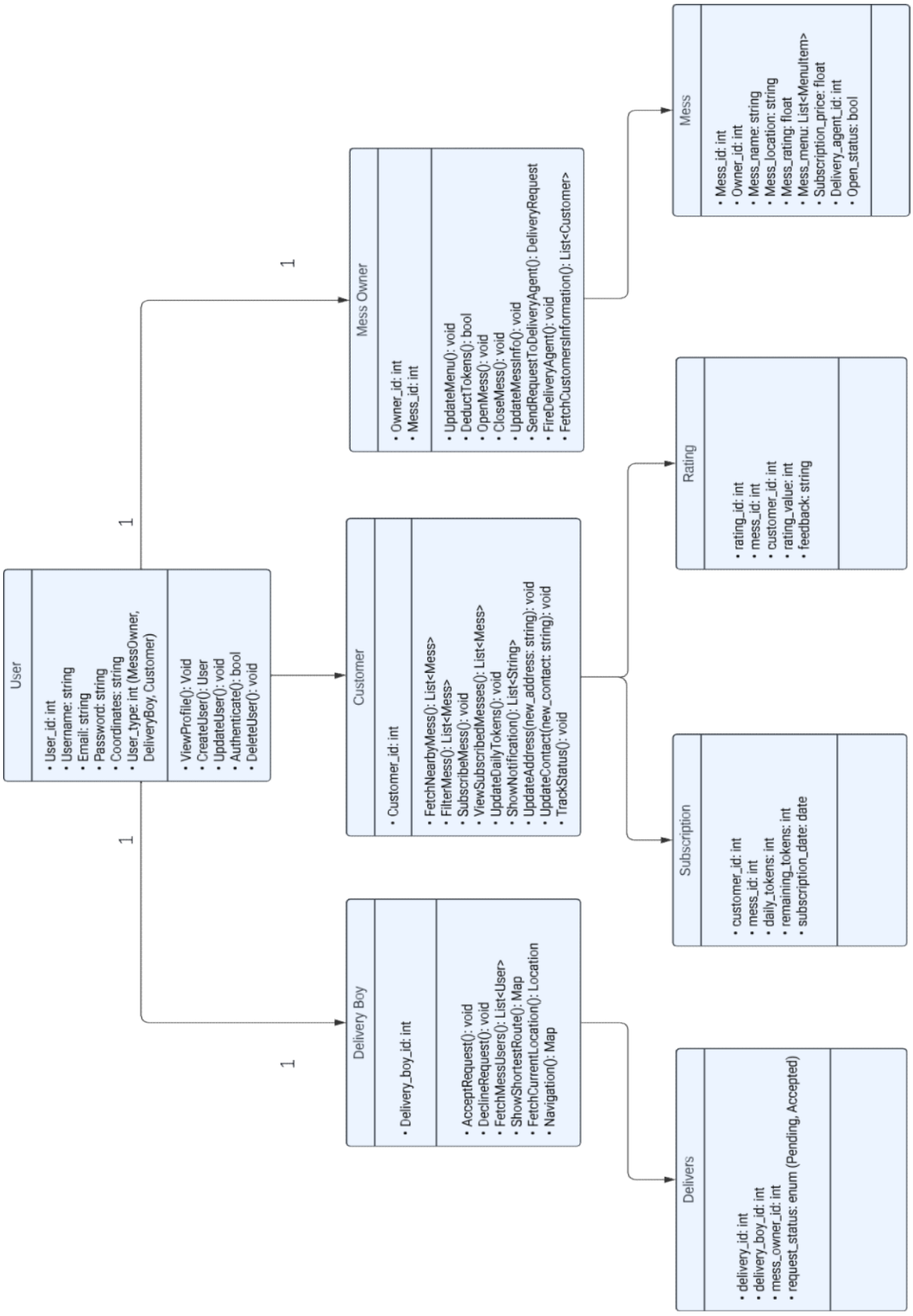


Figure 5.2: Activity Diagram of System

5.3 ER Diagram



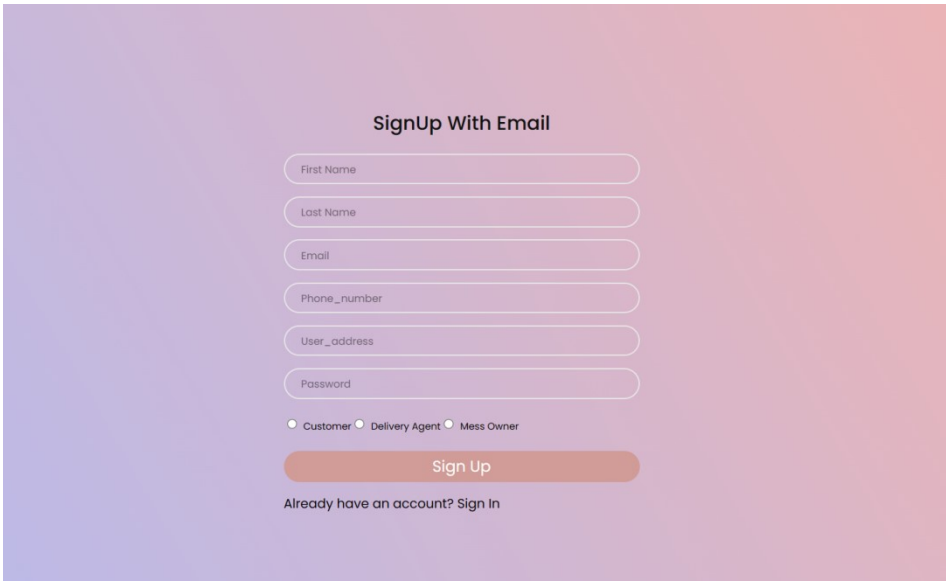
5.4 Class Diagram



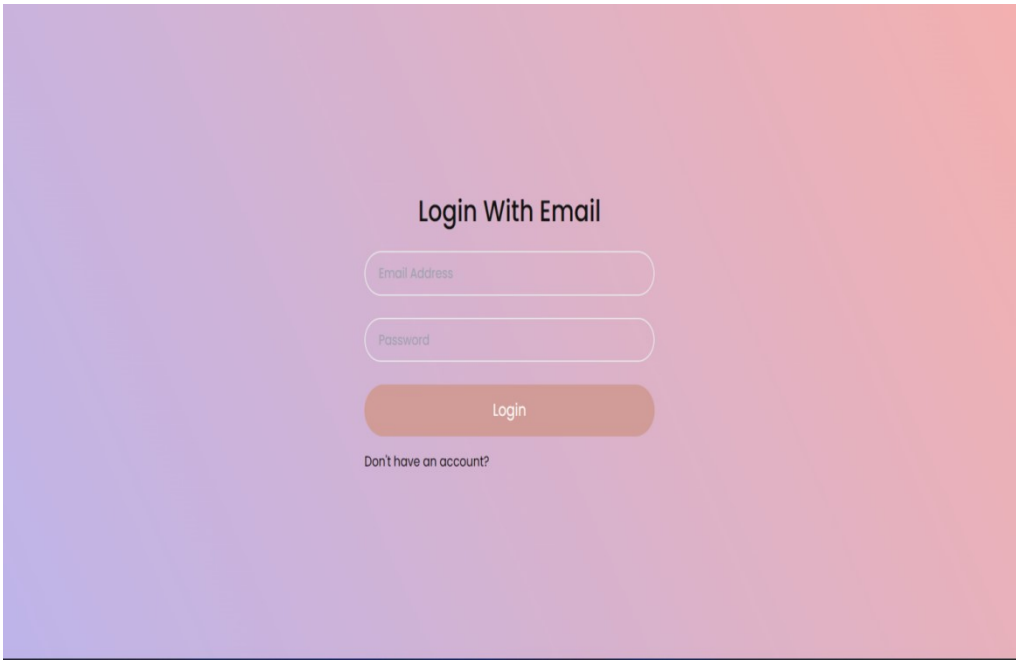
Chapter 6

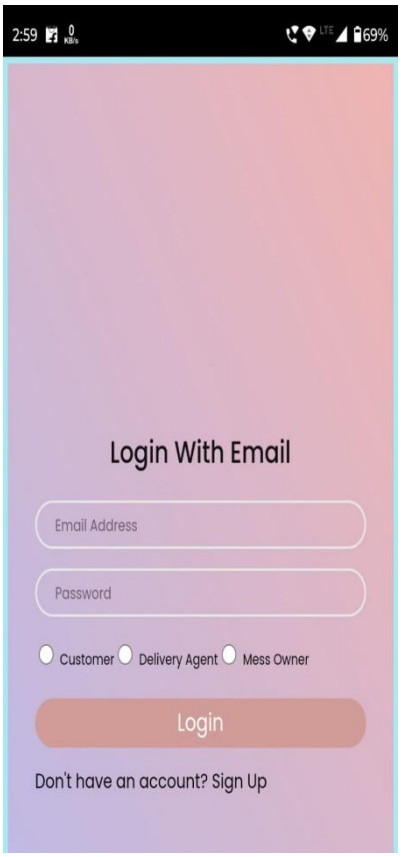
IMPLEMENTATION

6.1 Implementation Details

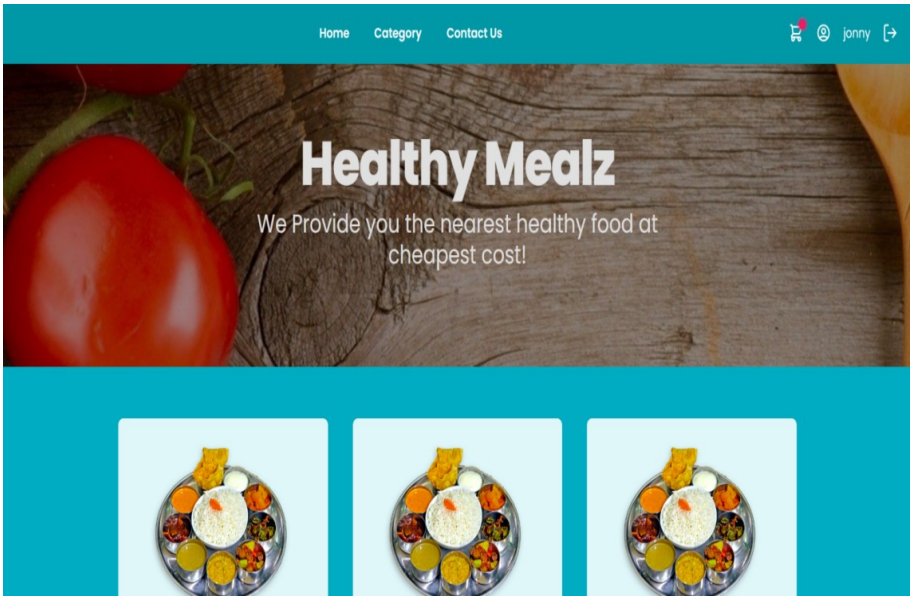


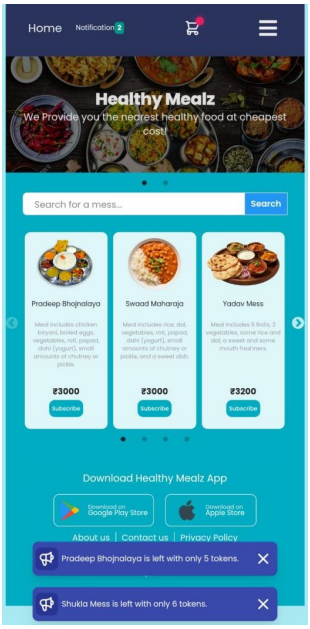
This is the Healthy Mealz Login entry point, if a customer is not yet registered then he/she can register on the moving signup page.



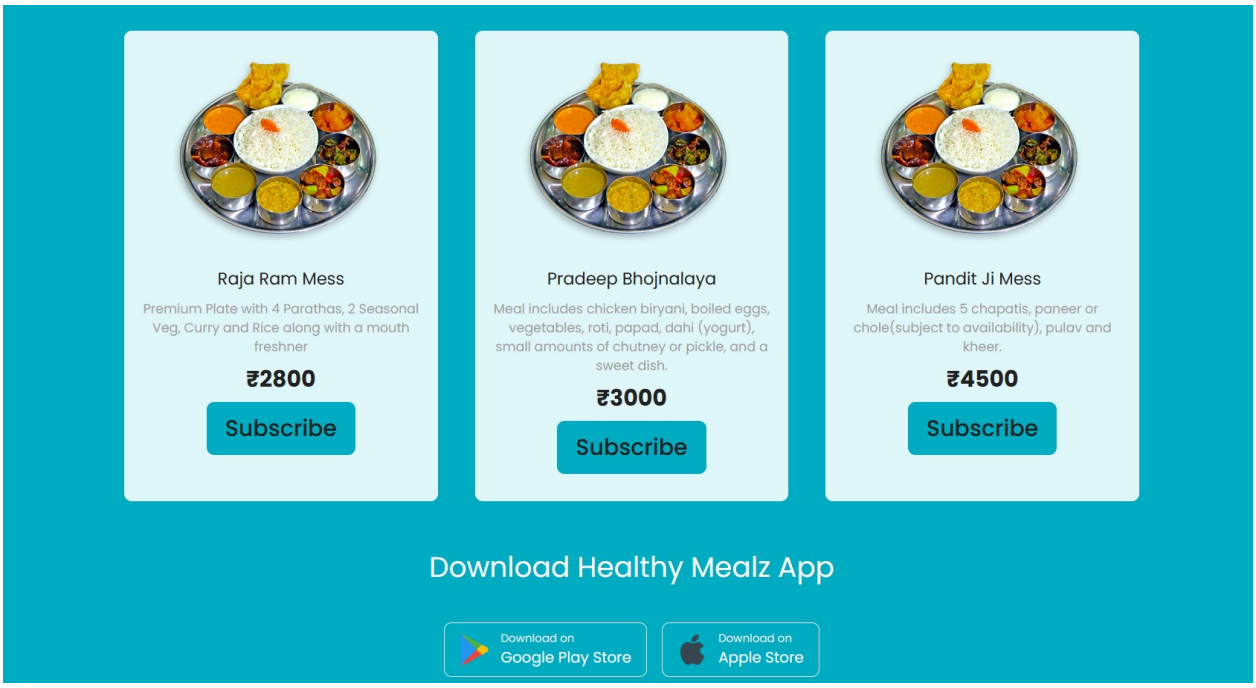


If user is registered, now user can login from the login page.

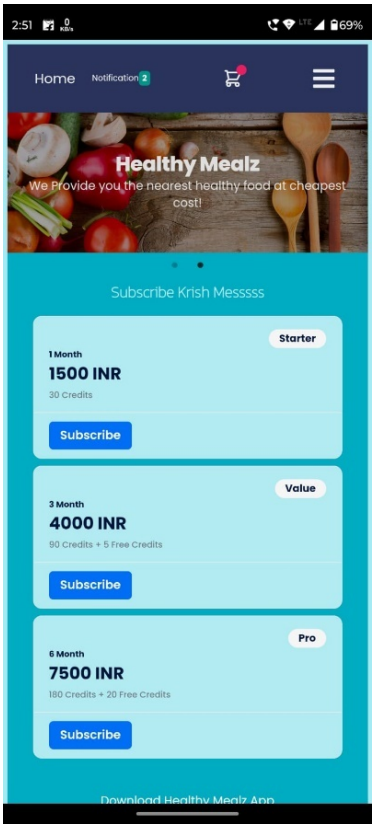


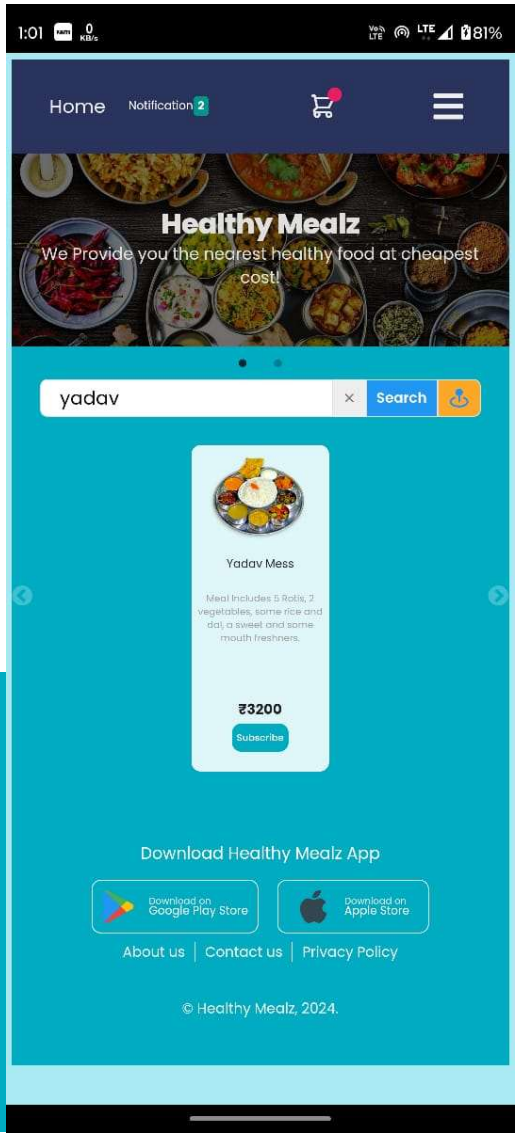


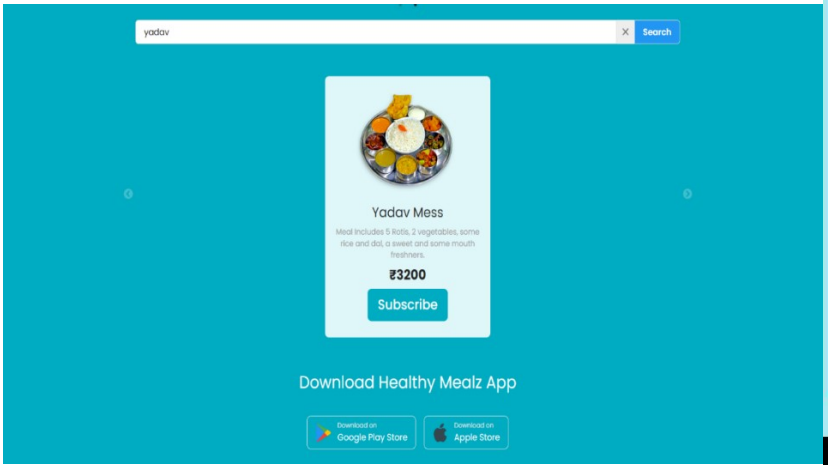
This is the home page of our application where users can select different messes.

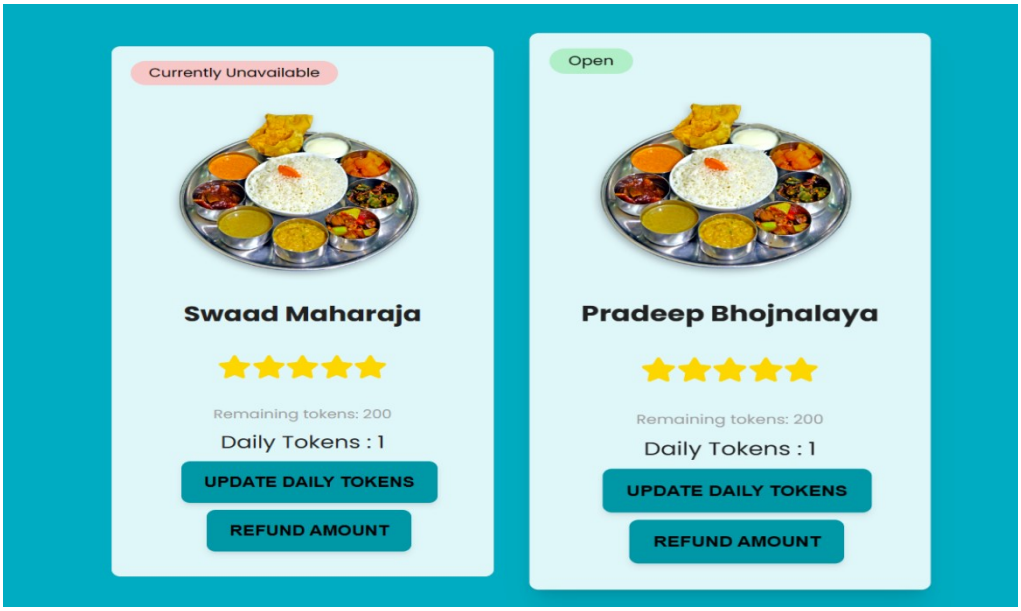


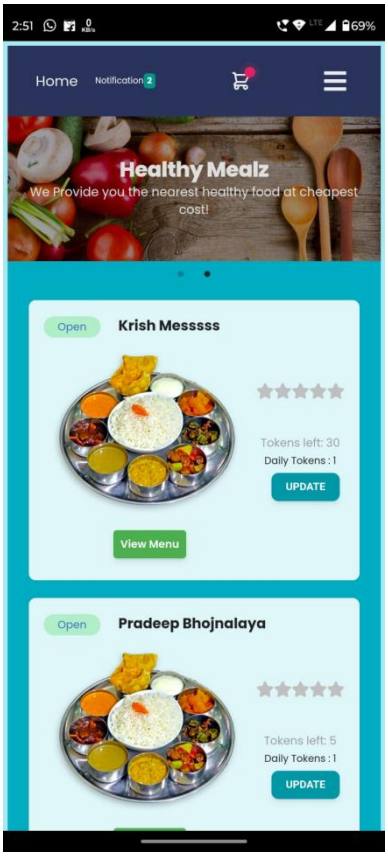
Here, users can subscribe for different messes.


2:51
69%
Home
Notification 2
Healthy Mealz
We Provide you the nearest healthy food at cheapest cost!
Subscribe Krish Messsss
Starter
1 Month
1500 INR
30 Credits
Subscribe
Value
3 Month
4000 INR
90 Credits + 5 Free Credits
Subscribe
Pro
6 Month
7500 INR
180 Credits + 20 Free Credits
Subscribe
Download Healthy Mealz App

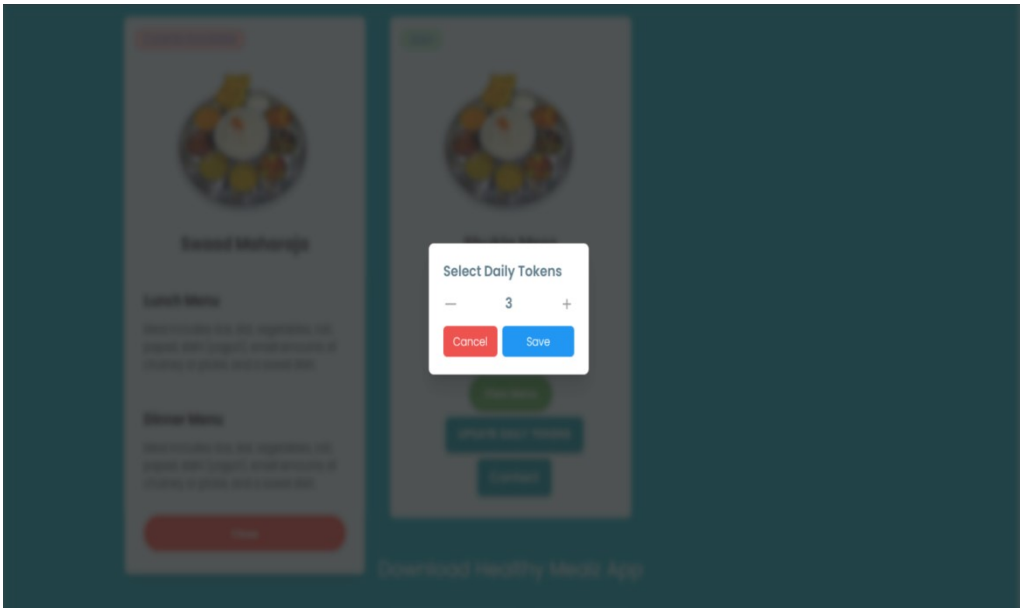

1:01
81%
Home
Notification 2
Healthy Mealz
We Provide you the nearest healthy food at cheapest cost!
yadav
Search
Yadav Mess
Meal includes 5 Rotis, 2 vegetables, some rice and dal, a sweet and some mouth freshners.
₹3200
Subscribe
Download Healthy Mealz App
Download on Google Play Store
Download on Apple Store
About us | Contact us | Privacy Policy
© Healthy Mealz, 2024.

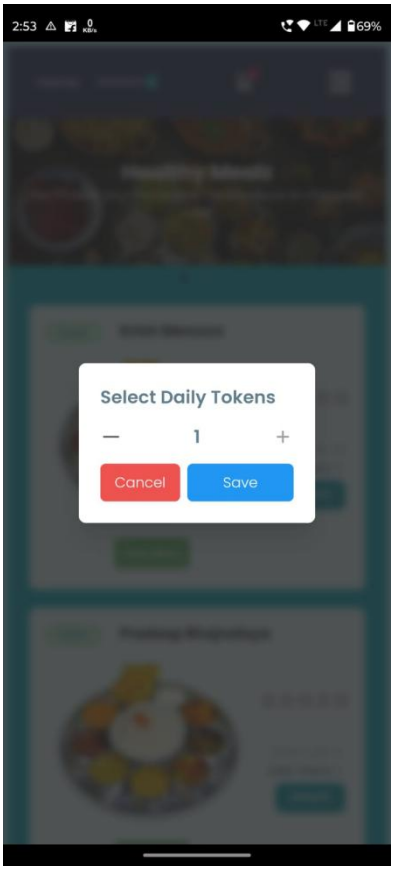

yadav
Search
Yadav Mess
Meal includes 5 Rotis, 2 vegetables, some rice and dal, a sweet and some mouth freshners.
₹3200
Subscribe
Download Healthy Mealz App
Download on Google Play Store
Download on Apple Store



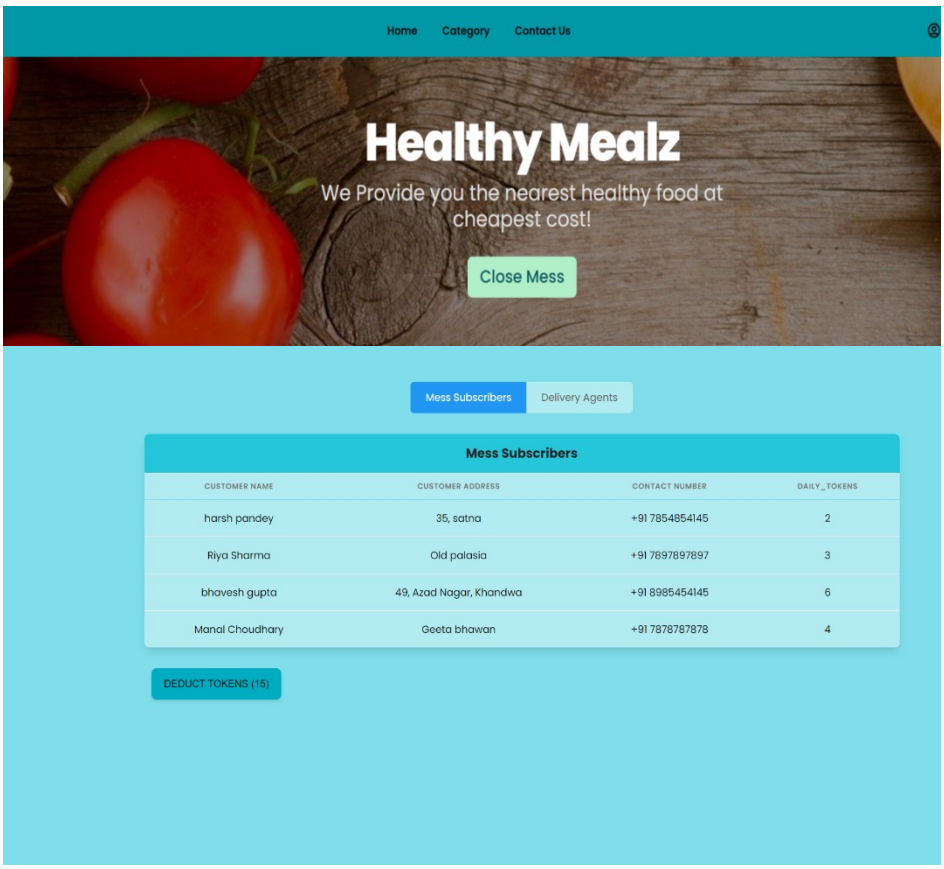


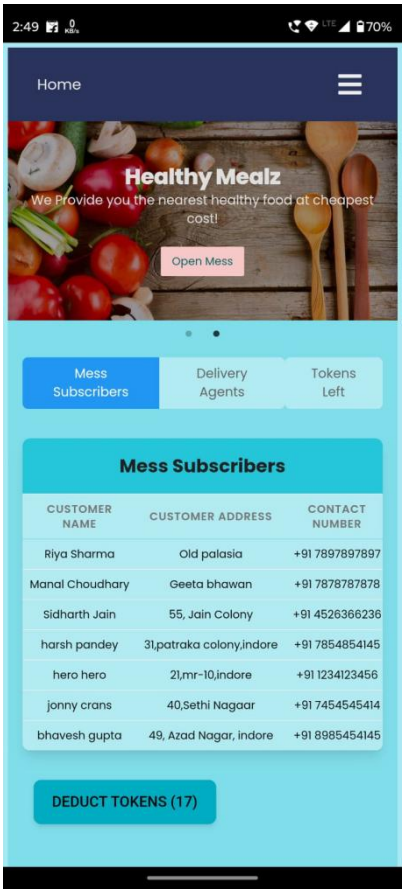
Users can see the daily tokens and update the number of tokens used per day.



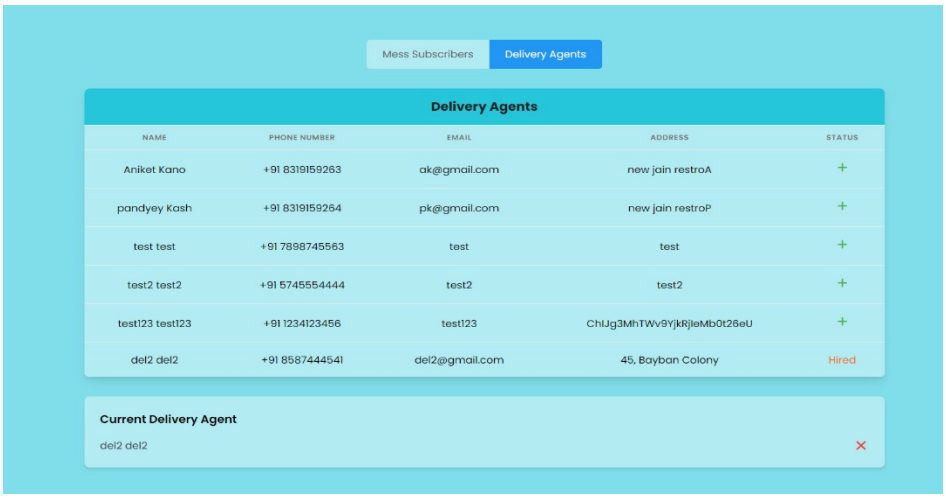


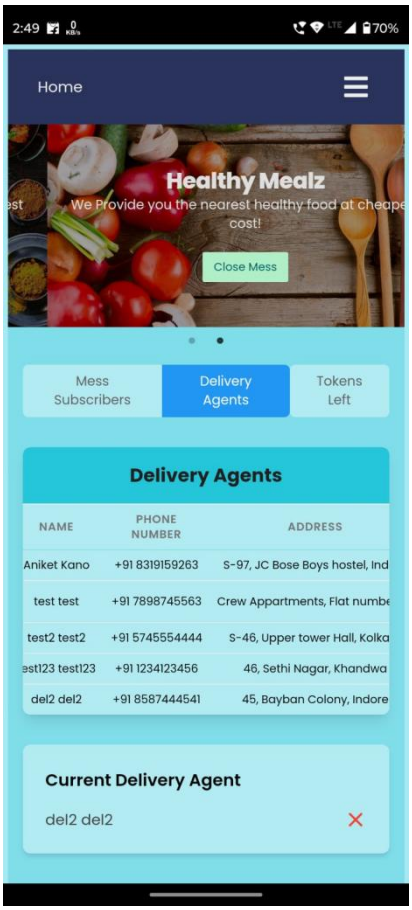
Users can see Menus of every mess.



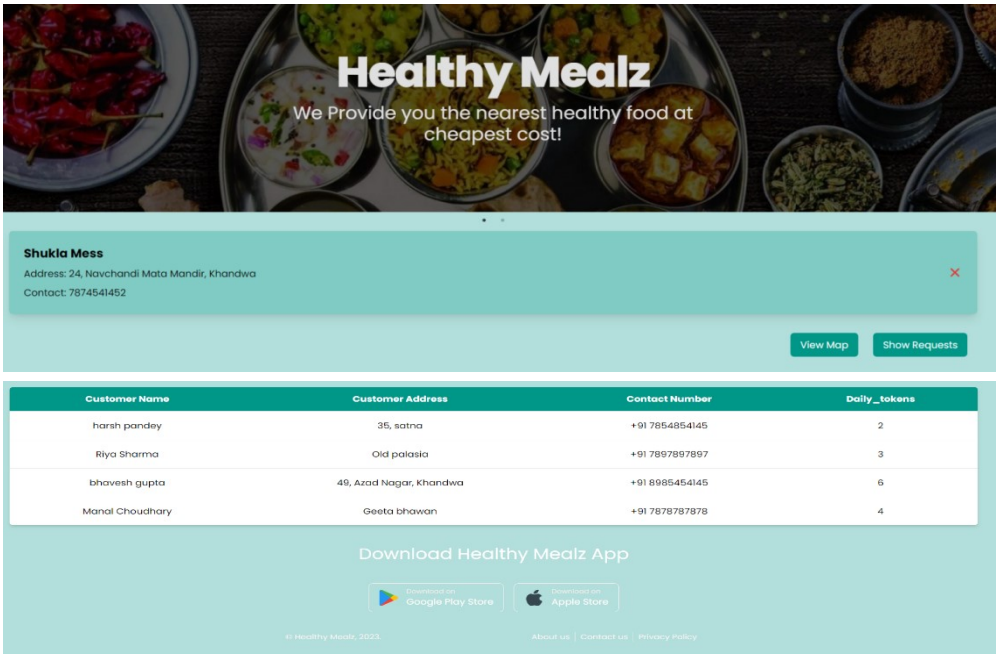


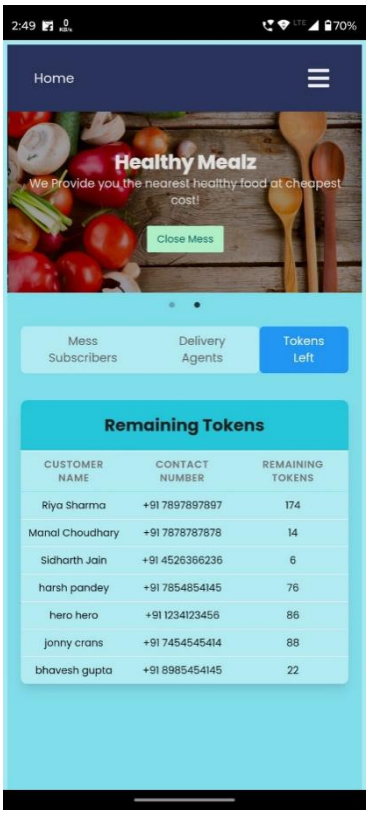
Mess owner can open and close their mess on their Dashboard, as well as view current mess subscribers' basic personal details.



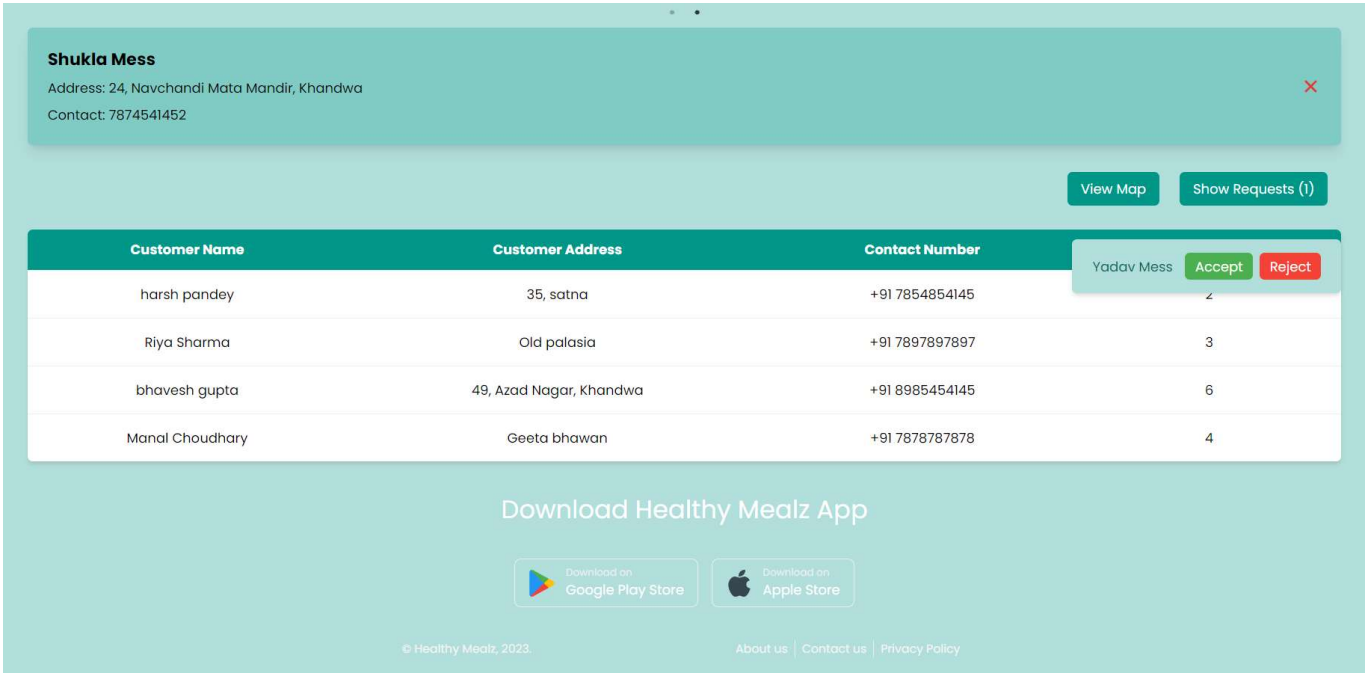


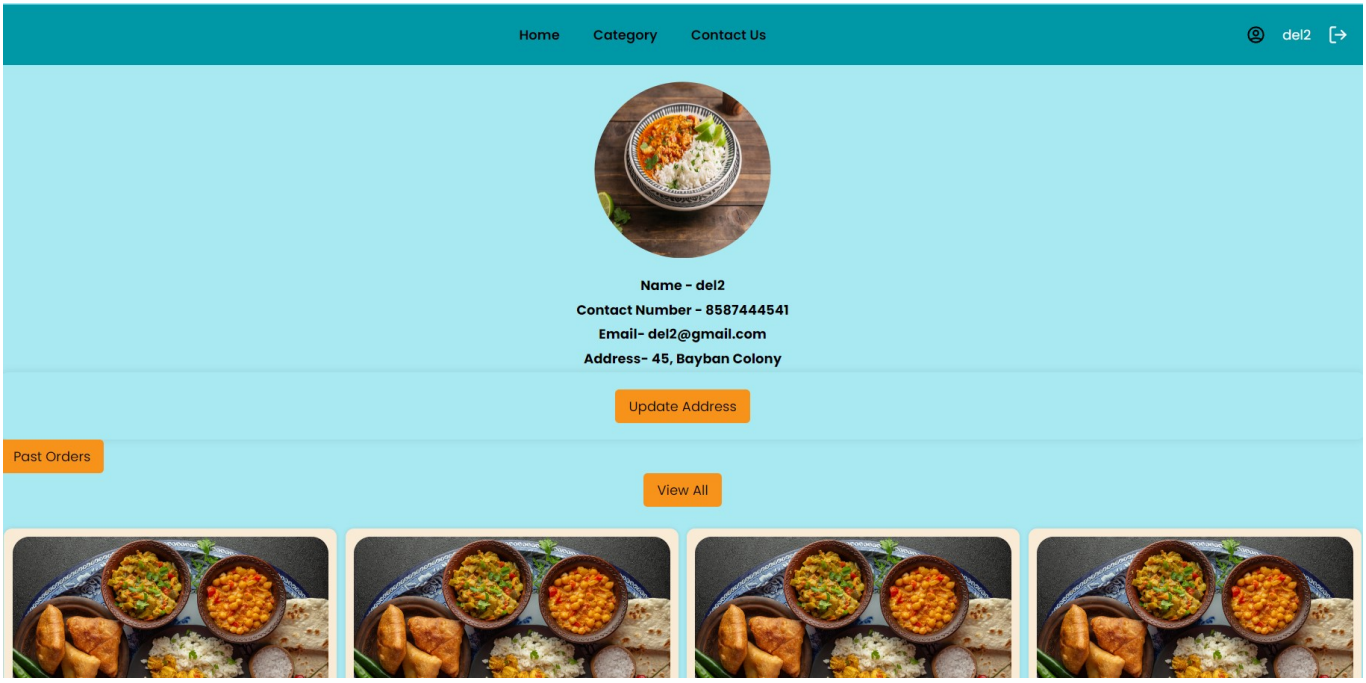
Mess owners can have different delivery agents ready to perform their duty.

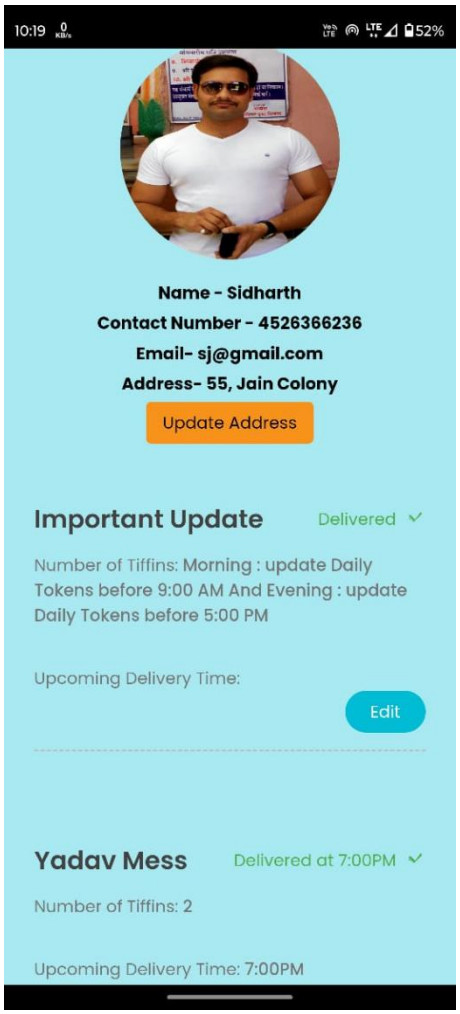




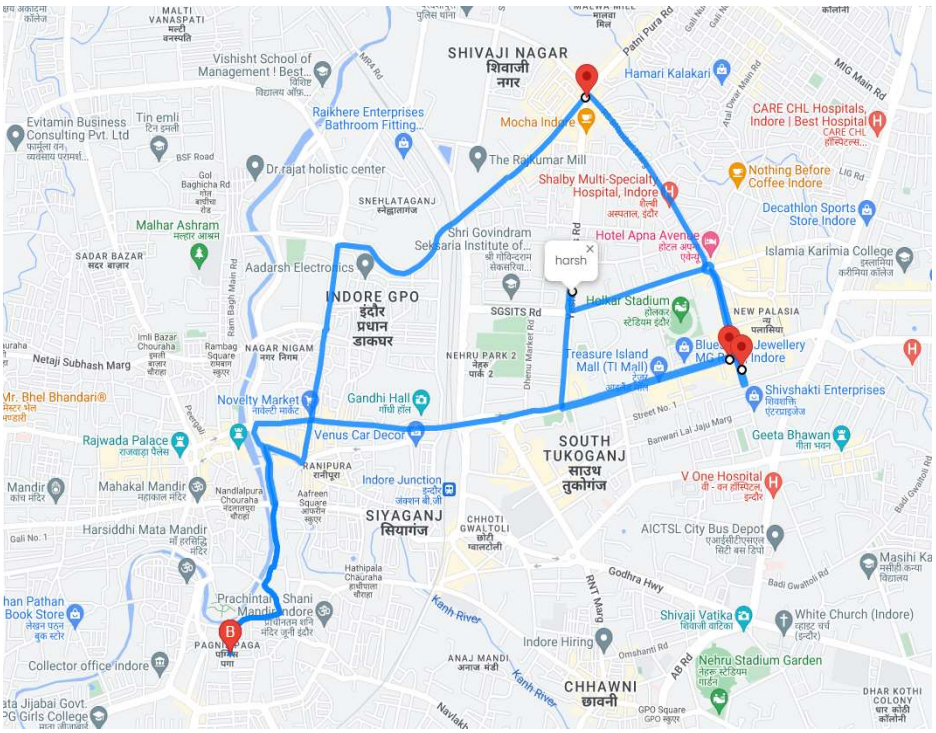
The Delivery agent can work for multiple mess, by accepting their request to work as a delivery agent.

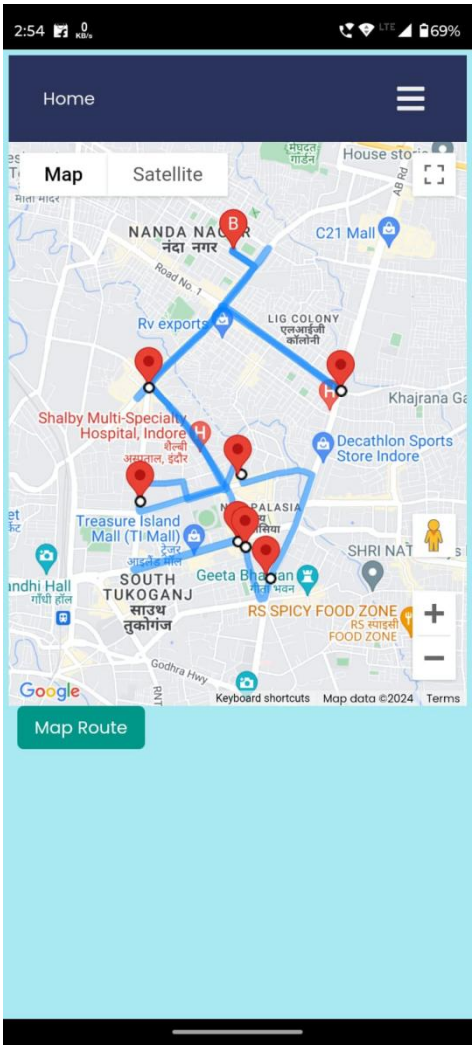






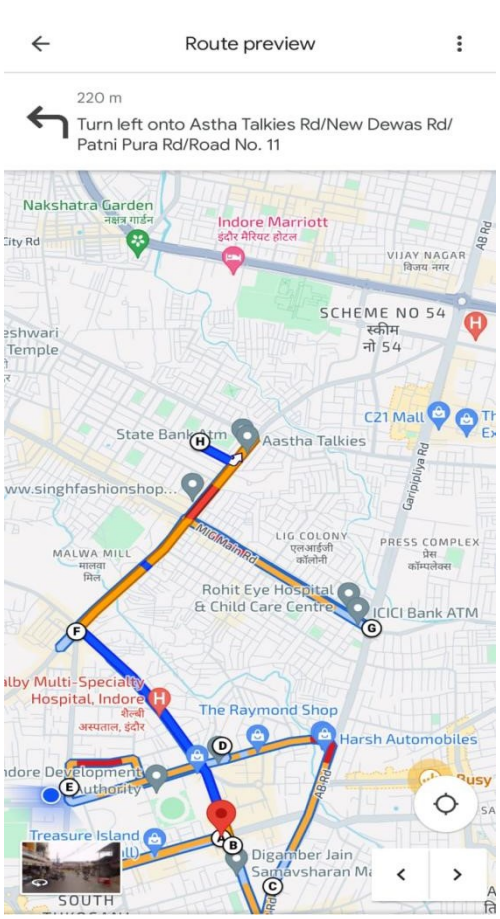
This is the customer's personal dashboard which shows basic personal details and their past orders





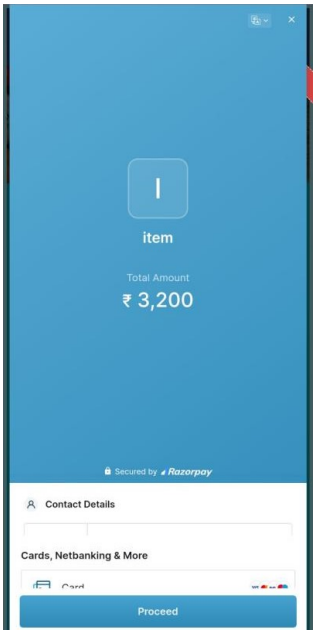
Map will be visible to the delivery agent.

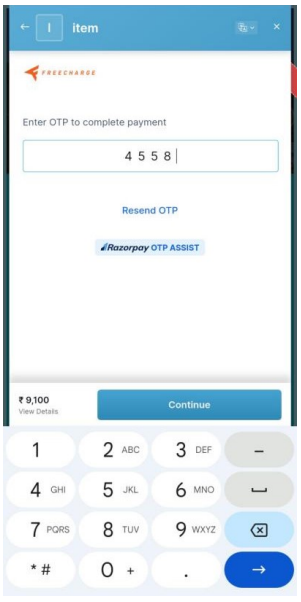
This is the route that the delivery boy will follow to deliver the order.



In this Customer can navigate with google maps.

Payment Gateway:





SHORTEST PATH ALGORITHM IMPLEMENTATION

```
function sortWaypoints(waypoints) {
    const sortedWaypoints = [waypoints[0]];
    const remainingWaypoints = waypoints.slice(1);

    while (remainingWaypoints.length > 0) {
      let closestWaypointIndex;
      let closestWaypointDistance = Infinity;

      for (let i = 0; i < remainingWaypoints.length; i++) {
        const distance = getDistance(
          sortedWaypoints[sortedWaypoints.length - 1].location,
          remainingWaypoints[i].location
        );

        if (distance < closestWaypointDistance) {
          closestWaypointIndex = i;
          closestWaypointDistance = distance;
        }
      }

      sortedWaypoints.push(remainingWaypoints[closestWaypointIndex]);
      remainingWaypoints.splice(closestWaypointIndex, 1);
    }

    return sortedWaypoints.map(({ location, stopover }) => ({ location, stopover }));
  }

  function getDistance(location1, location2) {
    const latDiff = location1.lat - location2.lat;
    const lngDiff = location1.lng - location2.lng;
    return Math.sqrt(latDiff * latDiff + lngDiff * lngDiff);
  }
```

This code defines two functions: sortWaypoints and getDistance, aimed at sorting a list of waypoints based on their proximity to each other.

1. sortWaypoints(waypoints): This function takes an array of waypoints as input and sorts them based on their proximity to each other. It starts by initializing an array called sortedWaypoints with the first waypoint from the input array. Then, it creates a copy of the input array called remainingWaypoints,

excluding the first element.It enters a while loop that continues until all waypoints are processed. Within the loop:

- It initializes variables closestWaypointIndex and closestWaypointDistance to keep track of the index and distance of the closest waypoint.
- It iterates over the remainingWaypoints array using a for loop.
- For each remaining waypoint, it calculates the distance from the last sorted waypoint using the getDistance function.
- If the calculated distance is smaller than the current closest distance, it updates closestWaypointIndex and closestWaypointDistance with the new values.

After finding the closest waypoint, it adds it to the sortedWaypoints array and removes it from the remainingWaypoints array using splice.Finally, it returns an array of sorted waypoints with only the location and stopover properties.

2. getDistance(location1, location2): This function calculates the distance between two locations represented by latitude and longitude coordinates. It takes two location objects as input, each containing lat and lng properties.It calculates the difference in latitude and longitude between the two locations and uses the Pythagorean theorem to compute the Euclidean distance between them. Finally, it returns the calculated distance.

Overall, the sortWaypoints function iterates through all the waypoints, finding the closest one to the last sorted waypoint and adding it to the sorted list. It repeats this process until all waypoints are sorted. The getDistance function is used to calculate the distance between two locations based on their latitude and longitude coordinates.

DEEP LINKING WITH GOOGLE MAP FOR NAVIGATION

```
const openGoogleMaps = () => {
    const origin = ${mess_loc.lat},${mess_loc.log};
    const destination = ${mess_loc.lat},${mess_loc.log};

    const waypointsString = waypoints.map(waypoint =>
${waypoint.location.lat},${waypoint.location.lng}).join('|');

    const url = https://www.google.com/maps/dir/?
api=1&origin=${origin}&destination=${destination}&waypoints=${waypointsString}&travelmode=driving;

    // Open the URL in a new tab or window
    window.open(url, '_blank');
  };
```

This code defines a function **openGoogleMaps** that generates a URL to open Google Maps with specified origin, destination, and intermediate waypoints for driving directions.

1. **const openGoogleMaps = () => { ... }**: This line declares a function named **openGoogleMaps** using arrow function syntax.
2. **const origin = ${mess_loc.lat}, ${mess_loc.log};**: This line creates a constant named **origin** containing the latitude and longitude coordinates of the origin location. It uses template literals to insert the latitude (**mess_loc.lat**) and longitude (**mess_loc.log**) values.
3. **const destination = ${mess_loc.lat},${mess_loc.log};**: This line creates a constant named **destination** containing the latitude and longitude coordinates of the destination location, following the same template literal approach as above.
4. **const waypointsString = waypoints.map(waypoint => ${waypoint.location.lat}, ${waypoint.location.lng}).join('|');**: This line constructs a string representing the intermediate waypoints for the journey. It uses the **map** function to iterate over the **waypoints** array, extracting the latitude and longitude coordinates of each waypoint. These coordinates are then concatenated into a string separated by | symbols, indicating the delimiter for waypoints in the Google Maps URL.
5. **const url = https://www.google.com/maps/dir/?api=1&origin=${origin}&destination=${destination}&waypoints=${waypointsString}&travelmode=driving;**: This line constructs the Google Maps URL for obtaining driving directions. It concatenates various parameters:
    - **origin**: The origin location.
    - **destination**: The destination location.
    - **waypoints**: The intermediate waypoints string.
    - **travelmode**: Specifies the mode of travel, in this case, "driving".
    - Other parameters like **api=1** are standard parameters required for using the Google Maps API.

GOOGLE MAP API CALLING

```
import { GoogleMap, LoadScript, DirectionsService, DirectionsRenderer, Marker, InfoWindow }
from '@react-google-maps/api';

<DirectionsService
              options={{
                destination: { lat: parseFloat(mess_loc.lat), lng: parseFloat(mess_loc.log) },
                origin: { lat: parseFloat(mess_loc.lat), lng: parseFloat(mess_loc.log) },
                travelMode: 'DRIVING',
                waypoints: waypoints
              }}
              callback={directionsCallback}
            />
```

Using the **@react-google-maps/api** library to integrate Google Maps functionality into a React component.

1. Import Statements:

import { GoogleMap, LoadScript, DirectionsService, DirectionsRenderer, Marker, InfoWindow } from '@react-google-maps/api';

- This line imports several components from the **@react-google-maps/api** library, including **GoogleMap**, **LoadScript**, **DirectionsService**, **DirectionsRenderer**, **Marker**, and **InfoWindow**. These components enable various functionalities related to Google Maps integration within a React application.

2. DirectionsService Component:

<DirectionsService options={{ destination: { lat: parseFloat(mess_loc.lat), lng: parseFloat(mess_loc.log) }, origin: { lat: parseFloat(mess_loc.lat), lng: parseFloat(mess_loc.log) }, travelMode: 'DRIVING', waypoints: waypoints }} callback={directionsCallback} />

- This code represents the **DirectionsService** component, which is part of the **@react-google-maps/api** library.
- The **options** prop specifies various parameters for obtaining directions. These include the **destination** (latitude and longitude of the destination), **origin** (latitude and longitude of the origin), **travelMode** (mode of travel, in this case, 'DRIVING'), and **waypoints** (an array of intermediate waypoints).

- **parseFloat()** is used to convert the latitude and longitude coordinates from strings to floating-point numbers.
- The **callback** prop specifies a function (**directionsCallback**) that will be called once the directions are retrieved from the DirectionsService.

3. Explanation-

- The **DirectionsService** component is used to fetch directions (e.g., driving directions) from Google Maps API based on the provided options.
- In this case, it's fetching driving directions from a specified origin (likely the same as the destination for some reason) to a destination location (**mess_loc.lat** and **mess_loc.log**), with optional intermediate waypoints provided in the **waypoints** array.
- Once the directions are retrieved, the **callback** function (**directionsCallback**) will be invoked to handle the directions data. This callback function involve updating state or rendering the directions on the map.

RAZORPAY PAYMENT API CALLING

```
const initPayment = (data,months) => {
    const options = {
      key: "rzp_test_7XkaKD6uJ4qEsm",
      amount: data.amount,
      currency: data.currency,
      name: "item",
      description: "Test Transaction",
      // image: book.img,
      order_id: data.id,
      handler: async (response) => {
        try {
          const verifyUrl = "http://localhost:5000/Customer/verify";
          await axios.post(verifyUrl,
{response,months,mess_id,user_id:cookies.get("User").User_id})
                    .then((res) => {
                         alert("Successfully Subscribed to Mess!");
                         navigate("../tiffin"); });

        } catch (err) {
          if (err.response && err.response.data) {
                    alert(err.response.data);}
        }
      },
      theme: {
        color: "#3399cc",
      },
    };
    const rzp1 = new window.Razorpay(options);
    rzp1.open();
  };
```

This code defines a function **initPayment** intended to initiate a payment transaction using the Razorpay payment gateway.

1. Function Definition:

const initPayment = (data, months) => {

- This line defines a function named **initPayment** that takes two parameters: **data** (an object containing payment-related information) and **months** (presumably the number of months for the subscription).

2. Payment Options:

```
const options = { key: "rzp_test_7XkaKD6uJ4qEsm", amount: data.amount, currency: data.currency, name: "item", description: "Test Transaction", // image: book.img, order_id: data.id, handler: async (response) => { try { const verifyUrl = "http://localhost:5000/Customer/verify"; await axios.post(verifyUrl, { response, months, mess_id, user_id: cookies.get("User").User_id }) .then((res) => { alert("Successfully Subscribed to Mess!"); navigate("../tiffin"); }); } catch (err) { if (err.response && err.response.data) { alert(err.response.data); } } }, theme: { color: "#3399cc", }, };
```

- This block defines an **options** object containing various configuration parameters for the Razorpay payment gateway.
- **key**: The public API key provided by Razorpay.
- **amount**: The amount to be charged for the transaction, likely retrieved from the **data** object.
- **currency**: The currency of the transaction, also retrieved from the **data** object.
- **name**: The name of the item being purchased.
- **description**: A description for the transaction.
- **order_id**: The unique identifier for the order.
- **handler**: An async function that will be called when the payment is successfully completed. It sends a POST request to a verification endpoint (**verifyUrl**) with the payment response data, number of months, **mess_id**, and **user_id**.
- **theme**: An object defining the theme for the payment UI, including color.

3. Razorpay Initialization:

```
const rzp1 = new window.Razorpay(options);
```

- This line initializes a new instance of Razorpay with the provided options.

4. Opening the Payment Modal:

```
rzp1.open();
```

- This line opens the Razorpay payment modal, allowing the user to complete the payment process.

5. Explanation:
   - The **initPayment** function initializes a payment transaction using Razorpay, configured with the provided options.
   - When the payment is successfully completed, it sends a POST request to a verification endpoint and alerts the user of a successful subscription.
   - If an error occurs during the payment process, it alerts the user with the error message provided by the server response.

Chapter 7

CONCLUSION

In conclusion, the project aims to revolutionize the way individuals access and enjoy food services by leveraging technology to address common challenges in navigating dining options, especially in new or bustling urban environments. By providing a centralized platform or application, users can effortlessly locate tiffin centres, mess facilities, and home-based kitchens tailored to their preferences and dietary needs.

The objectives outlined in the project serve as guiding principles towards achieving this goal. By prioritizing user convenience, implementing location-based services, and maintaining a comprehensive database, the project ensures users can easily find and access food options that meet their requirements. Furthermore, promoting user engagement, fostering connectivity with home-based kitchens, and upholding quality standards contribute to enhancing the overall dining experience and fostering a sense of community among users.

Through strategic promotion and growth initiatives, the project aims to expand its reach and impact, reaching users across different localities and demographics. Additionally, establishing a feedback mechanism and continuously iterating based on user input ensures the platform remains responsive to evolving needs and preferences.

Finally, developing a sustainable monetization strategy ensures the long-term viability and growth of the project, enabling it to continue providing valuable services to users while also supporting local businesses and entrepreneurs in the food industry.

In essence, the project represents a holistic approach to addressing the challenges of accessing affordable, nutritious, and convenient food options, while also fostering connections within local communities and promoting sustainability in food delivery practices. Through these efforts, the project endeavors to enrich the dining experience for users and contribute positively to their overall well-being and satisfaction.

Chapter 8

REFERENCES

[1] Postgres official documentation- https://www.postgresql.org/docs/

[2] Git official documentation - https://git-scm.com/doc

[3] Star UML and A guide to agile process blogs.

[4] Nodejs Documentation - https://nodejs.org/docs/latest/api/

[5] React Documentation - https://react.dev/reference/react

APPENDIX A

Glossary

1. **Google Maps API**

The Google Maps API is one of those clever bits of Google technology that helps you take the power of Google Maps and put it directly on your own site. It lets you add relevant content that is useful to your visitors and customize the look and feel of the map to fit with the style of your site.

2. **Capacitor**

A cross-platform native runtime for web apps. Capacitor is an open-source native runtime for building Web Native apps. Create cross-platform iOS, Android, and Progressive Web Apps with JavaScript, HTML, and CSS. Capacitor delivers the same cross-platform benefits, but with a more modern approach to app development, taking advantage of the latest Web APIs and native platform capabilities.

3. **Geolocation API**

The Geolocation API is a service that accepts an HTTPS request with the cell tower and Wifi access points that a mobile client can detect. It returns latitude/longitude coordinates and a radius indicating the accuracy of the result for each valid input.

4. **Code Editor**

A code editor is essential for writing and editing code. Popular choices include Android Studio, Visual Studio Code, Atom, Sublime Text, or WebStorm.

5. **Elephant SQL**

Elephant SQL is a PostgreSQL database hosting service. Elephant SQL will manage administrative tasks of PostgreSQL, such as installation, upgrades to latest stable version and backup handling. Elephant SQL is also integrated to several cloud application platforms (also known as PaaS). With a click of a button your database is provisioned in the same data centre as your application is hosted, and is ready to be used immediately.