**Learning Platform Web Application**

**By**

|  |  |
| --- | --- |
| **Sadia Shabbir** | **2019-GCUF-069488** |
| **Maryam Tariq** | **2019-GCUF-069475** |
| **Samia Shabbir** | **2019-GCUF-069444** |

Project submitted in partial fulfillment of the requirements for the degree of

**BACHELOR OF SCIENCE**

**IN**

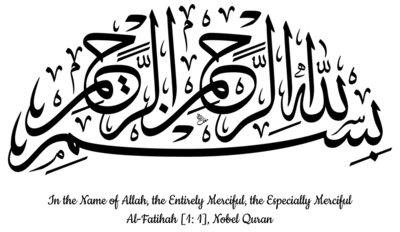
**COMPUTER SCIENCE**



**DEPARTMENT OF COMPUTER SCIENCE**

**Government College University Faisalabad**

# 2024



## DECLARATION

This project, a die-hard work, is productized out by Sadia Shabbir, Maryam Tariq and Samia Shabbir under the supervision of Mr. hamza (Lecturer, Computer Department) and Mr. **\_\_\_\_\_\_\_\_** (Hod, Computer Department) University Community College, GC University, Faisalabad, Pakistan. We feel please to declare that the project and contents of this project is the productive result of our hardworking, studies and research and no part of this is copied from any published source. This work has been conducted under the practical atmosphere of our studies not for the award of any other degree / diploma. The University may take action if the information provided is found guilty at any stage. Any external sources of information used in this project, including references, have been duly acknowledged through proper citations and bibliographical references. The project has not been previously submitted for any other degree or examination at any other institution. Any contributions made by others to this project, including guidance and support from faculty members, have been duly acknowledged. The software code, documentation, and any other materials presented as part of this project are the result of my own work, unless otherwise acknowledged. I take full responsibility for the authenticity and originality of the content presented in this project. I understand that any misrepresentation or falsification of information in this declaration will have serious consequences, including the possibility of disciplinary action by the institution.

**Signature of Student:**

Sadia Shabbir

Registration No. 2019-GCUF-069488

Maryam Tariq

Registration No. 2019-GCUF-069475

Samia Shabbir

Registration No. 2019-GCUF-069444

## Dedication

I dedicate this project to God Almighty my creator, my strong pillar, my source of inspiration, wisdom, knowledge and understanding. He has been the source of my strength throughout this program and on His wings only have I soared. I also dedicate this work to my Friends who has encouraged me all the way and whose encouragement has made sure that I give it all it takes to finish that which I have started. I dedicate this project, titled "Learning Platform Web Application" to the individuals who have been a source of inspiration and support throughout this journey. To my family, for their unwavering love, encouragement, and understanding. Your belief in me has been my driving force, and I am grateful for your constant support. To my project supervisor, [Mr. Hamza], for their guidance, expertise, and valuable insights. Their mentorship has been invaluable in shaping this project and my growth as a developer. To my friends and classmates, for their camaraderie, motivation, and the countless hours spent brainstorming ideas and discussing challenges. Your presence has made this project more enjoyable and fulfilling. To the staff and faculty members at [**Rise college** ], for providing a conducive learning environment and resources that have contributed to my education and the successful completion of this project. To all the users and stakeholders who participated in the testing and evaluation of the app, providing valuable feedback and helping me improve the functionality and user experience. Lastly, I would like to express my gratitude to everyone who has played a role, big or small, in this project. Your support and belief in my abilities have been instrumental in its completion. This project is dedicated to each and every one of you. Thank you for being a part of my journey and for making it a truly enriching experience.

.

## ACKNOWLEDGEMENT

I would like to express my sincere gratitude and appreciation to all those who have contributed to the successful completion of my Final Year Project titled "Learning Platform Web Application " This project has been an enriching and rewarding experience, and I would like to acknowledge the individuals and organizations for their support. My project supervisor, **Mr. Hamza,** for their invaluable guidance, expertise, and continuous support throughout the project. Their knowledge and insights have been instrumental in shaping the direction of this work. The faculty members of **Rise College,** for providing me with a conducive learning environment and valuable resources. I am grateful for their dedication to education and their efforts in imparting knowledge and skills. My family and friends, for their unwavering support, encouragement, and understanding. Their belief in me and their motivation have been a constant source of inspiration, driving me to overcome challenges and strive for excellence. The participants who willingly volunteered their time and provided feedback during the testing and evaluation phases of the project. Their insights and suggestions have been invaluable in refining the functionality and user experience of the app. The developers and contributors of open-source libraries, frameworks, and tools that were utilized in the development of this project. Their efforts have significantly expedited the development process and enhanced the overall quality of the application. I would also like to extend my gratitude to all the train passengers, restaurant owners, and delivery personnel who may benefit from this train food ordering app. Their needs and expectations have been a driving force behind the development of this solution. Lastly, I want to express my appreciation to all the individuals, both mentioned and unnamed, who have directly or indirectly contributed to this project. Your support and assistance have played an integral role in its successful completion. In conclusion, I am truly grateful to everyone who has been a part of this project. Your contributions, encouragement, and support have been invaluable, and I am honored to have had the opportunity to work on this project with such wonderful individuals.

**Sadia Shabbir (Roll No. 2260)**

### Maryam Tariq (Roll No. 2206)

### Samia Shabbir (Roll No. 2259)

To

The Controller of Examinations, Government College University,

Faisalabad Layyah Campus

I, the Supervisor, certifies that the contents and form of the project submitted by

**Sadia Shabbir (Roll No. 2260)**

### Maryam Tariq (Roll No. 2206)

### Samia Shabbir (Roll No. 2259)

Have been found satisfactory for the award of the degree.

## Internal Examiner

Name:

Signature:

**Lecturer:** University Community College (Government College University Faisalabad) Department of Computer Science

## External Examiner

Name:

Signature:

# Table of Contents

## Table of Contents ................................................................................................. XII Chapter-1 Introduction ............................................................................................ 1

1.1 Introduction ........................................................................................................................... 1

1.2 Background............................................................................................................................ 1

1.3 Purpose .................................................................................................................................. 1

1.4 Scope ..................................................................................................................................... 1

1.5 Objective ............................................................................................................................... 1

1.6 Intended Audience and Reading Suggestions........................................................................ 2

1.7 Process Model ....................................................................................................................... 2 1.8 Document Convention ........................................................................................................... 2

## Chapter-2 Software Requirement Specification .................................................... 3 2.1 Overall Description ......................................................................................... 3

2.1.1 Product Perspective ............................................................................................................ 3

2.1.2 Product Features ................................................................................................................. 3

2.1.3 Design and Implementation Constraints .............................................................................. 4

2.1.4 Assumptions and Dependencies .......................................................................................... 4

## 2.2 System Features .............................................................................................. 5

2.2.1 Login/Sing Up .................................................................................................................... 5

2.2.2 Add to Cart ......................................................................................................................... 5

2.2.3 Adding New Items .............................................................................................................. 5

2.2.4 Check Details ...................................................................................................................... 5

2.2.5 Order Details ...................................................................................................................... 5

## 2.3 External Interface Requirements .................................................................. 6

2.3.1 User Interfaces .................................................................................................................... 6

2.3.2 Hardware Interfaces ............................................................................................................ 7

2.3.3 Software Interfaces ............................................................................................................. 7

2.3.4 Communications Interfaces ................................................................................................ 7

## 2.4 Other Nonfunctional Requirements .............................................................. 7

2.4.1 Performance Requirements ................................................................................................. 7

2.4.2 Safety Requirements ........................................................................................................... 7

2.4.3 Security Requirements......................................................................................................... 8

## Chapter-3 Analysis ................................................................................................... 9

3.1 Identifying Actors and Use Cases using Textual Analysis .................................................... 9

3.1.1 Use Case Name: Restaurant Owner ............................................................................. 9

3.1.2 Use Case Name: Passenger ........................................................................................ 10

3.1.3 Use Case Name: Delivery Boy ................................................................................... 11

3.2 Forming Use Case Diagram with Candidate and Use Cases .............................................. 12

3.2.1 Use Case diagram: Restaurant Owner ........................................................................ 12

3.2.2 Use Case diagram: Passenger ..................................................................................... 12

3.2.3 Use Case diagram: Delivery Boy .............................................................................. 13

3.3 Describe the Events Flow for Use Case ............................................................................. 13

3.3.1 Addition of Restaurant Owner ................................................................................... 13

3.3.2 Addition of Passenger ................................................................................................ 13 3.3.3 Addition of Delivery Boy ........................................................................................... 14

## Chapter-4 Design .................................................................................................... 15

4.1 Architecture Diagram ......................................................................................................... 15

4.2 ERD Diagram with data dictionary ..................................................................................... 16

4.3 Class Diagram ..................................................................................................................... 17

4.4 Object Diagram ................................................................................................................... 18

4.5 Sequence Diagram ............................................................................................................... 19

4.6 Activity Diagram ................................................................................................................. 20

4.7 Collaboration Diagram ........................................................................................................ 21

4.8 State Transition Diagram ..................................................................................................... 22

## Chapter-5 Development ........................................................................................ 23

5.1 Operating System ................................................................................................................ 23

5.1.1 Available Operating System ....................................................................................... 23

5.1.2 Selected Operating System ........................................................................................ 23

5.2 Development Approach ....................................................................................................... 23

5.2.1 Available Development Approach ............................................................................. 24

5.2.2 Selected Development Approach ............................................................................... 24

5.3 Programming Language ...................................................................................................... 24

5.3.1 Available Programming Language ............................................................................. 25

5.3.2 Selected Programming Language .............................................................................. 25

5.4 Platform ............................................................................................................................... 26

5.3.1 Available Platform ..................................................................................................... 26

5.3.2 Selected Platform ....................................................................................................... 27

5.4 Case Tool ............................................................................................................................. 27

5.4.1 Available Case Tools .................................................................................................. 27

5.4.2 Selected Case Tool ..................................................................................................... 28

5.5 Database .............................................................................................................................. 28

5.5.1 Available Database ..................................................................................................... 28 5.1.2 Selected Database ...................................................................................................... 29

## Chapter-6 Testing ........................................................................................................ 30

6.1 Test Case Specification ....................................................................................................... 30

6.2 Black Box Test Case........................................................................................................... 30

6.2 1 BVA or Boundary Value Analysis ............................................................................. 30

6.2.2 Equivalence Class Partitioning ................................................................................... 30

6.2.3 State Transition Testing ............................................................................................. 30

6.2.4 Decision Table Testing ............................................................................................... 30

6.2.5 Graph Base Testing .................................................................................................... 30

6.3 White Box Testing .............................................................................................................. 31

6.3 1 Statement Coverage ................................................................................................... 31

6.3.2 Branch Coverage ........................................................................................................ 31

6.3.3 Path Coverage ............................................................................................................ 31 6. 4 Test Cases ........................................................................................................................... 32

## Chapter-7 Implementation ................................................................................... 36

7.1 Component Diagram ........................................................................................................... 36

7.2 Deployment Diagram ......................................................................................................... 37 7.3 Database Architecture (1- Tier, 2-Tier, 3- Tier Architecture) ............................................ 38

## Chapter-8 Tools And Technologies .................................................................... 40

8.1 Programming Language ...................................................................................................... 40 8.2 Operating Enjoinment ......................................................................................................... 40

## Appendix A: User Documentation ........................................................................ 41

User Documentation For Restaurant Owner ............................................................................. 41

User Documentation For Passenger .......................................................................................... 42

User Documentation For Delivery Boy ..................................................................................... 43

**Appendix B: Source Code ...................................................................................... 44 Appendix C: References ........................................................................................ 66**

# List of Figures

1. Main Interface ................................................................................................................. 6
2. Login/Register Screen ...................................................................................................... 6
3. Use Case Instructor........................................................................................... 12
4. Use Case Student ........................................................................................................ 12
5. Use Case Admin .................................................................................................. 13
6. Architecture Diagram ..................................................................................................... 15
7. ERD ................................................................................................................................ 16
8. Class Diagram ................................................................................................................ 17
9. Object Diagram .............................................................................................................. 18
10. Sequence Diagram ......................................................................................................... 19
11. Activity Diagram ............................................................................................................ 20
12. Collaboration Diagram ................................................................................................... 21
13. State Transmission Diagram .......................................................................................... 22
14. Component Diagram ...................................................................................................... 23
15. Deployment Diagram ..................................................................................................... 37
16. 1-Tier Architecture ......................................................................................................... 38
17. 2-Tier Architecture ......................................................................................................... 39
18. 3-Tier Architecture ......................................................................................................... 39

## XI

**Abstract:**

## The popularity of online education is increasing rapidly. This project focuses on developing a comprehensive learning platform designed to cater to both children and adults through various courses and blog articles. The platform supports roles such as Lecturer/Tutor, Student, and Admin, with features tailored to different user categories: Kids and Adults. Users can engage in actions including accessing articles, enrolling in courses, and tracking their progress. Interaction with content is enhanced through comments, replies, views, and likes. In this project, technologies such as React, MongoDB, Node.js, and Express.js are used to create a responsive and user-friendly learning platform that facilitates effective online education for users of all ages. The platform ensures content is appropriate through age-based filtering and promotes engagement through interactive features like comments, likes, and progress tracking.

## Keywords

Here are some keywords related to a train food ordering app: Online Learning Platform, E-Learning, Online Courses, Education Technology, Adult Learning, Children's Education, Free Courses, Paid Courses, Online Tutoring and Student Progress Tracking

XII

**Chapter-1 Introduction to the Problem**

## 1.1 Introduction

## Learning Platform Web Application is a comprehensive online learning platform designed for both children and adults, offering diverse courses and educational blogs. Users can easily explore, enroll in courses, and track their progress with features like comments, likes, and views. With age-based content filtering and a ranking system, Learn Together ensures relevant, high-quality educational experiences for all users.

## 1.2 Background

## Existing educational platforms often lack personalized features and age-based content filtering, leading to a fragmented learning experience. Learning Platform Web Application addresses these issues with interactive course management, role-specific, and a progress tracking system to enhance the educational journey for all users

## 1.3 Purpose

The purpose of the Learning Platform Web Application is to create an engaging and accessible online learning environment for both children and adults. This platform enables users to explore diverse educational content, participate in interactive courses, and track their progress through a user-friendly interface. Learning Platform Web Application aims to enhance the learning experience by providing role-specific features for lecturers, students, and admins, ensuring that all users receive high-quality, age-appropriate educational resources.

## 1.4 Scope

## The scope of the Learning Platform Web Application encompasses the development of a comprehensive online learning system. It includes features for user registration, role-based access (teachers, students, admins), course management (creation, editing, deletion), blog management (creation, editing, deletion), interactive content (likes, comments, views), and a ranking system based on user engagement metrics. The platform will support age-based content filtering to ensure appropriate educational material for users under 18 and provide a responsive, mobile-friendly interface using technologies such as React, MongoDB, Node.js, and Express.js. The project aims to enhance the learning experience through seamless user interaction and personalized educational content delivery.

## 1.5 Objective

The objective is to develop an online food ordering service Android app, to enable the user to place their order at any time during traveling in train.

1. The convenience of ordering from any place at any time.

1. It is way easier to save time and effort to order your favorite food with just a few clicks.

1. Mobile apps provide the freedom to order from any place during traveling in train.

1. Passengers can easily order fresh and tasty food either online or through their mobile phone.

1. Provide passengers with a convenient and hassle-free way to order meals and snacks during their train journey. Eliminate the need for passengers to carry their own food or rely on limited onboard dining options.

XIII1

## 1.6 Intended Audience and Reading Suggestions

This application can be used by passengers. This is only for those who want food while traveling. This app is used when traveling by train.

## 1.6 Process Model

The process model for developing a train food ordering app involves several key steps. First, requirements gathering is conducted to understand the needs of train passengers and food vendors. This information guides the design of the system architecture, including client-side and server-side components. Food vendors are then onboarded onto the platform by verifying credentials and establishing partnerships. Vendor profiles and menus are set up on the app, allowing passengers to browse and order meals. Secure payment processing is implemented for seamless transactions. Real-time order tracking and notifications are integrated to keep passengers and vendors updated. Compatibility with multiple train routes and schedules is ensured. Rigorous testing is performed to ensure proper functionality and usability. Finally, the app is deployed on relevant platforms, and ongoing support and updates are provided to maintain its performance and meet user needs.

## 1.8 Document Conventions

When documenting a Train Food Ordering app, there are several conventions that can be used to ensure clarity and consistency. Here are some possible conventions:

**1.7.1 Title:** The document should have a clear and concise title that reflects the purpose of the app.

**1.7.2 Introduction:** The document should begin with an introduction that provides a brief overview of the app, its features, and its benefits.

**1.7.3 Getting started:** This section should provide step-by-step instructions for downloading and installing the app, as well as any initial setup steps required.

**1.7.4 User interface:** The document should include screenshots or illustrations of the app's user interface to help users navigate the app.

**1.7.5 Features:** This section should provide a detailed description of the app's features, including what they do and how to use them.

**1.7.6 Troubleshooting:** The document should include a section on troubleshooting common issues that users may encounter, along with solutions or workarounds.

**1.7.7 Frequently Asked Questions (FAQs):** This section should include a list of frequently asked questions about the app, along with their answers.

**1.7.8 Glossary:** If the app uses technical or specialized terminology, a glossary of terms can be included to help users understand the app's functionality.

**Chapter-2**

**Software Requirement Specification**

## 2.1 Overall Description

### 2.1.1 Product Perspective

"Train Food Delivery" is a brand new application. There are few similar applications abroad, but they serve a different purpose. Food Panda is a similar app but it's delivery only with no dine in option.

### 2.1.2 Product Features

The system comprises 3 major modules with their sub-modules as follows:

#### RESTAURANT

* Login
* Manage Food
  + Add/update/delete/view
  + Enable/disable food
  + Food preparation time
* Orders o Pending Orders:
  + List of orders which are yet to be delivered
  + Approve/Reject/Cancel
  + Assign Delivery Person
  + Update status
  + Today’s Orders:
  + List of all Today’s orders irrespective of status o All Orders:
  + List of all the orders
* Filter by date
* Manage Delivery Person o Add/Update/Delete/View
  + Manage Trains
  + Add/Update/Delete/View
  + Manage Stops & delivery & arrival time details (Add/ Update/ Delete/ View stops)

#### DELIVERY BOY

* Login
* Orders
  + List of Pending Orders
  + Order details
  + Passenger/Train details
  + Drop details
  + Update Status

#### PASSENGER

* Register
* Login
* Profile
* Change Password
* Home
  + Check Train & Drop Station Availability
  + If yes you will be able to see the Menu
* Foods
  + Cart
  + View/Remove
  + Proceed to Payment
* Orders
  + List of orders
  + Track/Cancel
* Transactions
  + Debit/Credit

### 2.1.3 Design and Implementation Constraints

* Java Programming Language
* Internet enabled computers are required.
* Customers will be required internet

### 2.1.4 Assumptions and Dependencies

Following are the assumptions:

* **AS-1** Internet-enabled computers will be available in the food court at each food stall.

* **AS-2** Customer will be required internet access to order.

* **AS-3** Riders will be available to deliver the food for nearby offices.

The app will be dependent on certain external factors such as:

* **DE-1** The app would be dependent on rules and regulations by government authorities.

* **DE-2** The system would require cloud for data storage

## 2.2 System Features

The System Feature are Following:

### 2.2.1 Login/Sing UP

#### Description and Priority

The signup process allows new users to create an account within the app. Users provide their necessary information, such as name, email address, and password. The login feature enables users to access their accounts by entering their registered email address and password. This authentication process verifies their identity and grants access to personalized features and functionalities.

#### Functional Requirements

* Name
* Password
* Email

### 2.2.2 Add to Cart

The "Add to Cart" functionality in a train food ordering app allows users to select and accumulate food items for purchase before proceeding to checkout. It provides a convenient way for users to browse through the available menu, make their selections, and keep track of their chosen items.

### 2.2.3 Adding New Items

The feature of adding new food items by a restaurant owner in a train food ordering app allows restaurant owners to expand their menu and offer additional food options to train passengers.

### 2.2.4 Check Details

The "Check Details" functionality in a train food ordering app allows users to review and verify the details of their food orders before finalizing the purchase. It provides an opportunity for users to ensure that their order is accurate and meets their requirements.

### 2.2.6 Order details

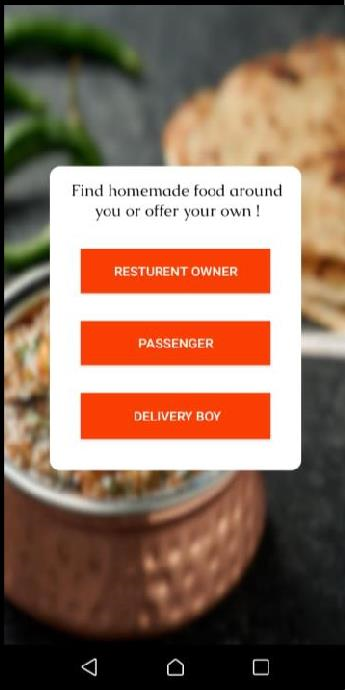
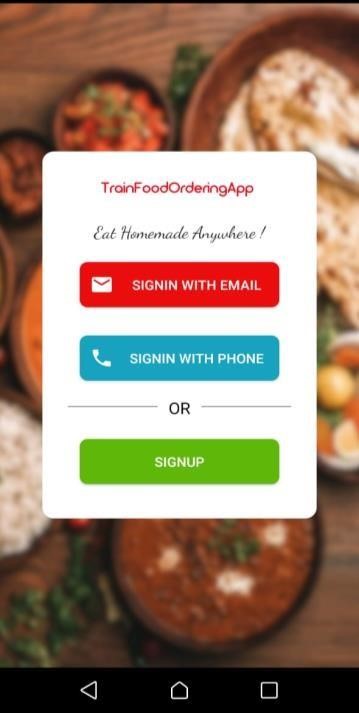
The "Order Details" functionality in a train food ordering app allows users to view and review the specific details of their placed orders. It provides users with comprehensive information about their selected food items, delivery preferences, payment details, and order status.

## 2.3 External Interface Requirements

### 2.3.1 User Interfaces

The system will use following User interfaces:

* Main Interface

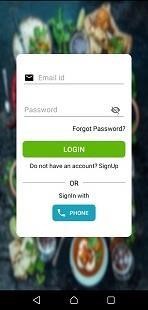
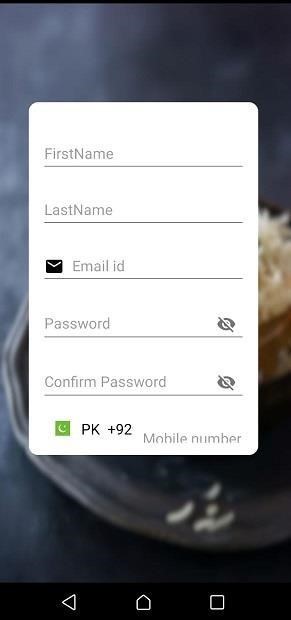


***Fig.1 Main Interface***

* Passenger Login/Register

* Restaurant Login/Register

* Delivery Boy Login/Register



***Fig.2 Login/Register Screen***

### 2.3.2 Hardware Interfaces

The system will use following communication interfaces:

* Android devices.
* RAM 8 GB
* Minimum Space Required 128GB
* Display 16 bit color

### 2.3.3 Software Interfaces

The system will use following Software interfaces:

* Operating system(Android)
* Android Studio
* Programming language (Java)

### 2.3.4 Communications Interfaces

The system will use following communication interfaces:

* This application must need an Internet Connection
* Emails
* Google Map

## 2.4 Other Nonfunctional Requirements

### 2.4.1 Performance Requirements

The performance requirements of this application are as following:

* This system will enable customers to place their order in a fast manner.
* This system will be available 24/7 every day.
* 92% of the queries shall be completed in approximately 3.5-4 seconds.
* Easy to use
* Error free

### 2.4.2 Safety Requirements

The safety requirements of this application are as following:

* Sensitive data isn't distributed among third party mediators.
* Sensitive data is not stored outside the app's storage system.
* The system cannot afford loss of data of its customers because it provides analysis on basis of it.
* The app should employ robust security measures to protect user data, including personal information, login credentials, and payment details. This involves implementing encryption techniques, secure data storage, and adhering to industry-standard security practices.
* The app should use secure authentication mechanisms to verify user identities during the login process. This can include password hashing, two-factor authentication (2FA), or biometric authentication for enhanced security.
* The app should comply with relevant privacy regulations and have a clear privacy policy that outlines how user data is collected, used, and protected. Users should have control over their data and be able to manage their privacy settings within the app.

### 2.4.3 Security Requirements

The security requirements of this application are as following:

* Every user must change his initial password after first successful login.
* Personal Data (Personal Information) of user is protected.
* Payment Methods or safe.
* Ensure compliance with relevant data protection laws in Pakistan, such as the Pakistan Personal Data Protection Act (PDPA) or any other applicable regulations. Understand the legal obligations regarding data collection, storage, and processing.
* Implement secure authentication mechanisms, such as strong password policies, password hashing, and encryption, to protect user login credentials and prevent unauthorized access.
* Utilize secure communication protocols, such as HTTPS, to encrypt data transmission between the app and backend servers. This protects user data from interception or unauthorized access.
* Implement secure data storage practices, including encryption of sensitive user data at rest. Employ appropriate access controls to prevent unauthorized access to the database or data storage systems.
* Clearly communicate the app's privacy policy to users, outlining how their data will be collected, used, and protected. Obtain user consent for data processing and provide options for users to manage their privacy settings.
* Develop an incident response plan to effectively handle security incidents or breaches. Define protocols for incident detection, containment, investigation, and notification, ensuring a timely and appropriate response.

## Chapter-3

**Analysis (Use case Model)**

**3.1 Identifying Actors and Use Cases using Textual Analysis**

### 3.1.1 Use Case Name: Restaurant Owner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DC#** | **1** | **Ref:** | | **Req#1** |
| **UC Name** | Signup as Restaurant Owner. | | | |
| **Level** | (Abstract or Detailed) | | | |
| **Description** | Create restaurant owner profile. | | | |
| **Actor** | Restaurant Owner | | | |
| **Stakeholders** | Restaurant Owner | | | |
| **Preconditions** | New account is created. | | | |
| **Success Guarantee** | Signup successfully**.** | | | |
| **Main Success Scenario** | **Action** | | **Response** | |
| 1. Enter the basic details. | |  | |
|  | | 2. Account created successfully. | |
| **Extensions** | If something went wrong, system respond accordingly | | | |
| **Special requirements** | Application should be running | | | |
| **Frequency of Occurrence** | Login Any Time | | | |

**3.1.2 Use Case Name: Passenger**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DC#** | **2** | **Ref:** | | **Req#2** |
| **UC Name** | Signup as Passenger. | | | |
| **Level** | (Abstract or Detailed) | | | |
| **Description** | Create Passenger profile. | | | |
| **Actor** | Passenger | | | |
| **Stakeholders** | Passenger | | | |
| **Preconditions** | New account is created. | | | |
| **Success Guarantee** | Signup successfully**.** | | | |
| **Main Success Scenario** | **Action** | | **Response** | |
| 1. Enter the basic details. | |  | |
|  | | 2. Account created successfully. | |
| **Extensions** | If something went wrong, system respond accordingly | | | |
| **Special requirements** | Application should be running | | | |
| **Frequency of Occurrence** | Login Any Time | | | |

**3.1.3 Use Case Name: Delivery Boy**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **`** | **3** | **Ref:** | | **Req#3** |
| **UC Name** | Signup as Delivery Boy. | | | |
| **Level** | (Abstract or Detailed) | | | |
| **Description** | Create Delivery Boy profile. | | | |
| **Actor** | Delivery Boy | | | |
| **Stakeholders** | Delivery Boy | | | |
| **Preconditions** | New account is created. | | | |
| **Success Guarantee** | Signup successfully**.** | | | |
| **Main Success Scenario** | **Action** | | **Response** | |
| 1. Enter the basic details. | |  | |
|  | | 2. Account created successfully. | |
| **Extensions** | If something went wrong, system respond accordingly | | | |
| **Special requirements** | Application should be running | | | |
| **Frequency of Occurrence** | Login Any Time | | | |

## 3.2 Forming Use Case Diagram with Candidate and Use Cases

**3.2.1**

**Use**

**Case**

**diagram:**

**Restaurant**

**Owner**

Sign

Up

Login

Restaurant

Owner

Add

Food

Confirm

Order

Manage

Delivery

Boy

***Fig.3***

***Use***

***case***

***Restaurant***

***Owner***

**3.2.2**

**Use**

**Case**

**diagram:**

**Passenger**



Sign

Up

Login

Check

Food

Passenger

Place

Order

Cancel

Order

***Fig.4***

***Use case***

***Passenger***



**3.2.3**

**Use**

**Case**

**diagram:**

**Delivery**

**Boy**

Delivery

Boy

***Fig.5***

***Use***

***case***

***Delivery***

***Boy***



Sign

Up



Login



Check

order



Pick

Order

## 3.3 Describe the Events Flow for Use Case

An outsider should be able to quickly comprehend the interactions between the actor(s) and the application from the sequence of events. The sequence of events should show what the system really does, not how it is intended to carry out the needed action.

**3.3.1 Addition of Restaurant Owner**

### • Precondition: Noun

* **Main Flow:**

Admin have to enter these formalities for the addition of new employee:

* Add first name
* Add last name
* Add data of birth
* Add mobile number
* Add email
* Add post detail and salary
* Add weekly working days.

**3.3.2 Addition of Passenger**

### • Precondition: Noun

* **Main Flow:**

Admin have to enter these formalities for the addition of new employee:

* Add first name
* Add last name
* Add data of birth
* Add mobile number
* Add email
* Add post detail and salary
* Add weekly working days.

**3.3.1 Addition of Delivery Boy**

### • Precondition: Noun

* **Main Flow:**

Admin have to enter these formalities for the addition of new employee:

* Add first name
* Add last name
* Add data of birth
* Add mobile number
* Add email
* Add post detail and salary
* Add weekly working days.

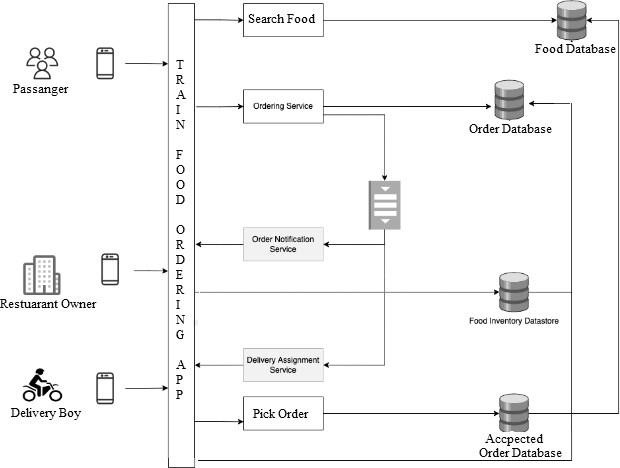
## Chapter-4 Design

**4.1 Architecture Diagram**

**4.1.1 Definition:**

An architecture diagram is a visual representation of all the elements that make up part, or all, of a system. Above all, it helps the engineers, designers, stakeholders — and anyone else involved in the project — understand a system or app’s layout.

**4.1.2 Diagram:**



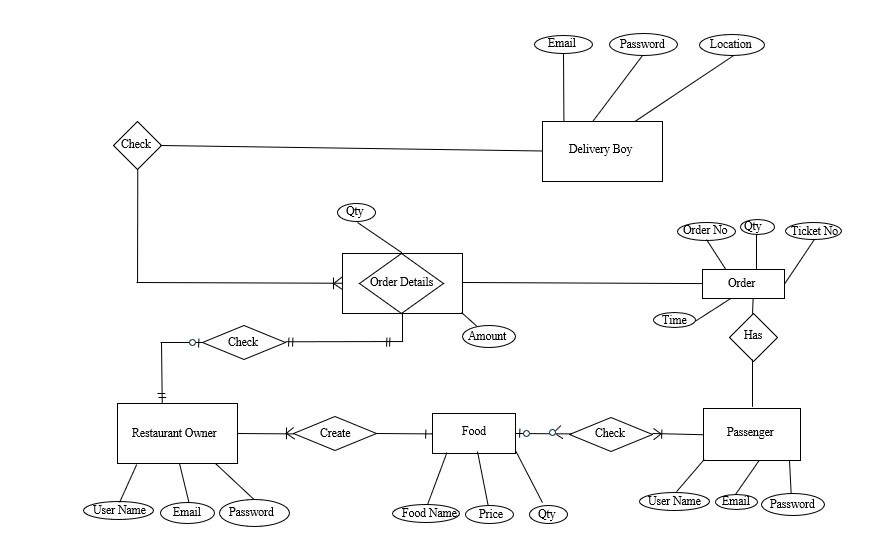
***Fig.6 Architecture Diagram***

## 4.2 ERD Diagram with data dictionary

**4.2.1 Definition:**

An Entity Relationship (ER) Diagram is a form of flowchart that shows the relationships between "entities" like people, things, or concepts inside a system. ER Diagrams are most frequently used in the disciplines of software engineering, business information systems, education, and research to build or troubleshoot relational databases. They are also known as ERDs or ER Models, and they employ a predetermined collection of symbols to represent the interconnectivity of entities, connections, and their qualities. These symbols include rectangles, diamonds, ovals, and connecting lines.

There are three main models and they have different attributes and different relationships. The restaurant owner creates the food. Food is a sub entity that has its own attributes. The passenger can also check the food. A passenger places an order where the order is a sub-entity that has its own attributes and is associated with order details that has its own attributes. the delivery person can check the order and pick up the order and deliver the order to the passenger **4.2.2 Diagram:**



***Fig.7 ERD***

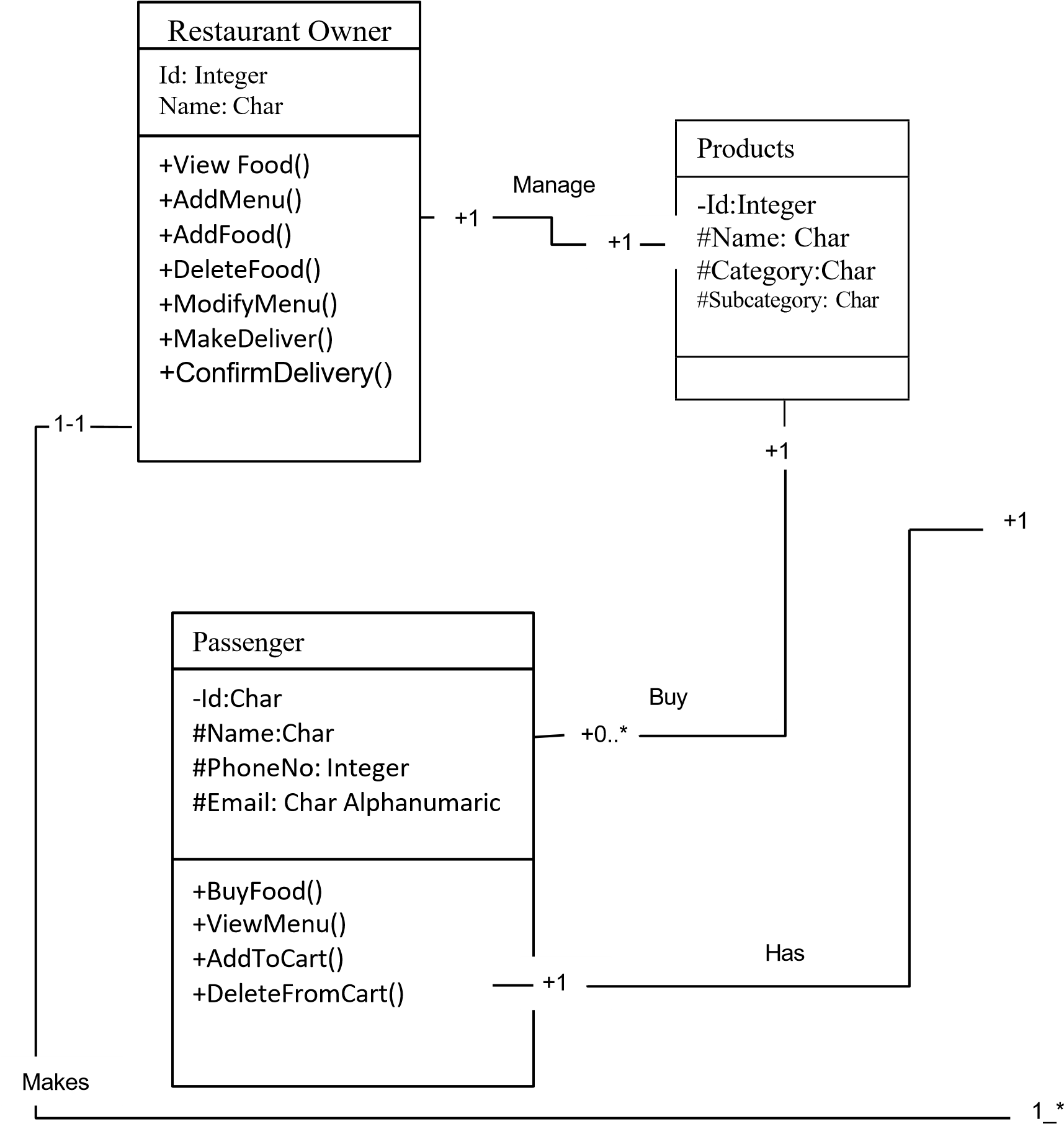
## 4.3 Class Diagram

**4.3.1 Definition:**

A class diagram in the Unified Modeling Language (UML) is a sort of static structural diagram used in software engineering that illustrates the classes, attributes, operations (or methods), and interactions between objects in a system to explain the system’s structure. There are three main classes restaurant owner, passenger and delivery boy they have different characteristics and perform different functionality. The passenger has two sub classes one is cart and buy product which is also sub class of restaurant. **4.3.2 Diagram:**

|  |  |
| --- | --- |
|  | Cart |
| -Id: Integar  #NumberOfProducts:  Integer  #Product1: Char  #Product2: Char  #Product3: Char  #Price: Float  #Total: Float |
|  |
|  |

|  |  |
| --- | --- |
|  | Delivery Boy: |
| -Id:Integer -Name: Char |
| +ViewOrder()  +PickOrder() |
|  |



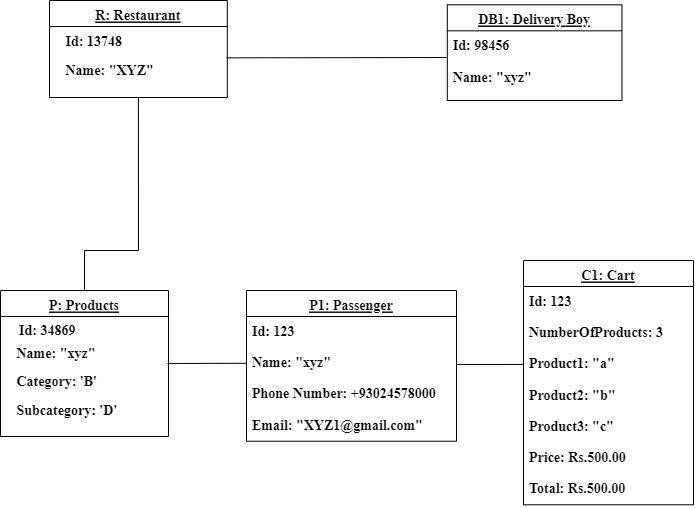
***Fig.8 Class Diagram***

## 4.4 Object Diagram

**4.4.1 Definition:**

A runtime instance of a class that can have its own state and data values is referred to as an object. An object diagram includes objects and their relationships, which may be regarded a specific case of a class diagram or a communication diagram. A static UML object diagram is an example of a class diagram; it displays a snapshot of the detailed state of a system at a moment in time.

**4.4.2 Diagram:**



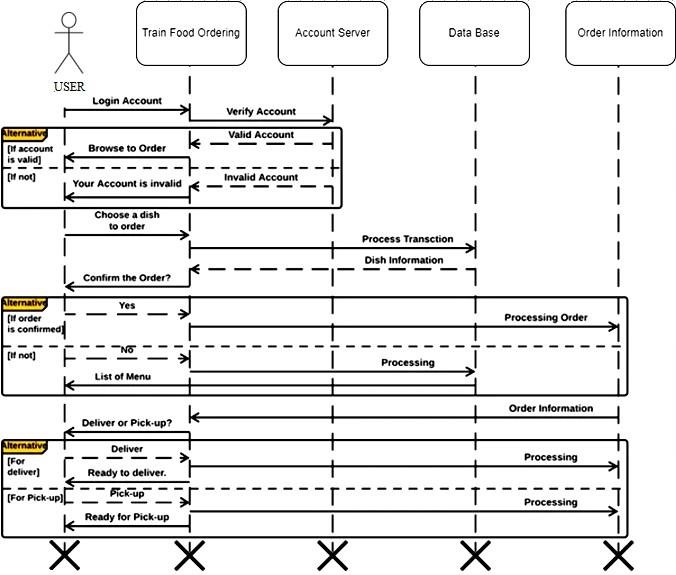
***Fig.9 Object Diagram***

## 4.5 Sequence Diagram

**4.5.1 Definition:**

A sequence diagram is a diagram created using the Unified Modeling Language (UML) that shows the flow of messages sent and received by objects during an interaction. A set of objects that are represented by lifelines and the messages they exchange over the course of an interaction make up a sequence diagram. First of all user login to the application and server give permission to login. If he/she is valid user then passenger chose food. For choosing food passenger interact with database and return different categories of food. The delivery boy can see the location of the restaurant and can pick up the order. After picking order check the location of passenger and deliver the food.

**4.5.2 Diagram:**



***Fig.10 Sequence Diagram***

## 4.6 Activity Diagram

**4.6.1 Definition:**

An activity diagram displays business and software processes as a succession of actions. These functions can be carried out by individuals, hardware, or software. Activity diagrams are used to define business processes and use cases, as well as to track the execution of system operations.

**4.6.2 Diagrams:**



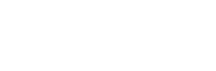
No



Accepted

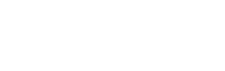


Yes



See

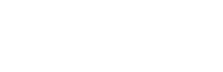
Order



Add

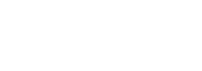
to

Card



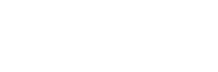
See

Order



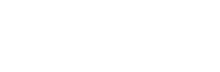
Place

Order



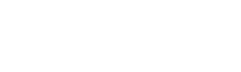
Create

Food

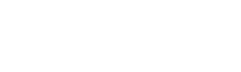


Manage

Order

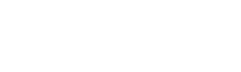


Exit

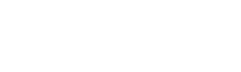


Delivery

Boy

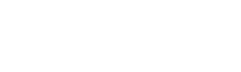


Passenger

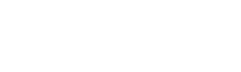


Restaurant

Owner



Server



Login

***Fig.11 Activity Diagram***

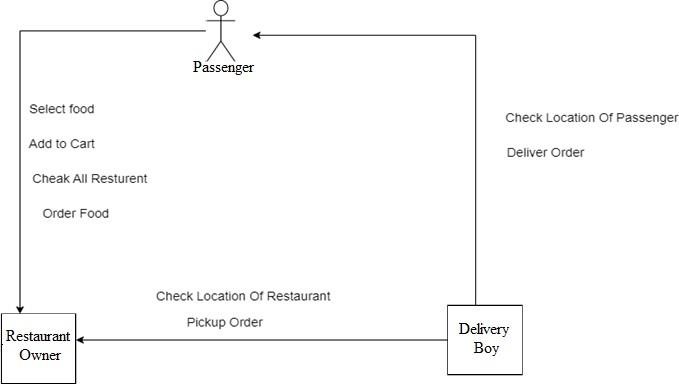
## 4.7 Collaboration Diagram

**4.7.1 Definition:**

In the Unified Modeling Language, a collaboration diagram—also called a communication diagram— illustrates the connections and interactions between software elements. These diagrams may be used to define each object's function as well as the dynamic behavior of a specific use case.

The passenger can choose food, add food to the basket, check restaurants and order food to the restaurant. The delivery driver checks the passenger's location and delivers the food to the passenger and also checks the restaurant's location and selects the order form restaurant and delivers to the passenger.

**4.7.2 Diagram:**



***Fig.12 Collaboration Diagram***

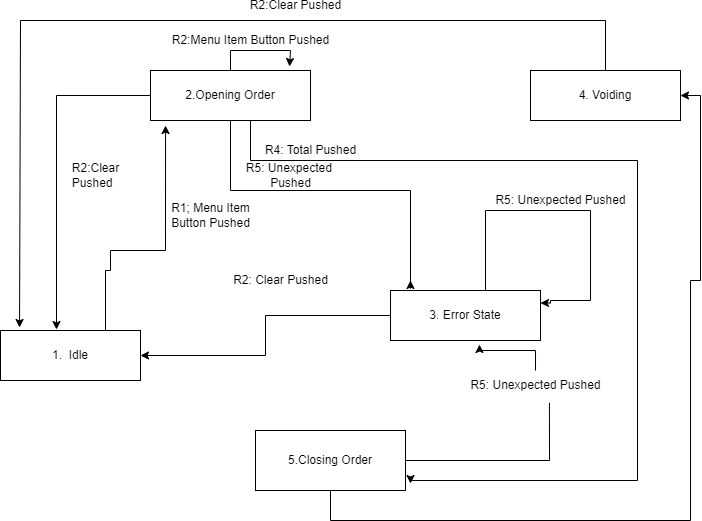
## 4.8 State Transition Diagram

**4.8.1 Definition:**

State-transition diagrams show the states that an object can be in, the circumstances under which it can change states, the guards that must be in place before the transition can occur, and the activities that are performed during an object's existence (actions). State-transition diagrams are incredibly useful for describing how various things behave throughout the whole spectrum of use cases that affect those objects. State-transition diagrams are useless for demonstrating the collaboration between elements that causes the transitions.

The first time system will be ideal. The menu item button pressed the open command, if an unexpected button was pressed, then it is in an error state, and if it was pressed, cancel the idle transition. If unexpected, automatic pushed cancel then canceled, the order was placed if not any error.

**4.8.2 Diagram:**



***Fig.13 State Transmission Diagram***

## Chapter-5 Development

**5.1 Operating System**

It is software that manages computer hardware and software resources, provides common services for computer programs, and enables users to interact with the computer system. The operating system acts as an intermediary between computer hardware and software applications, allowing them to communicate and function. Operating systems (OS) are used in computing devices to provide an essential layer of software that manages and coordinates the hardware and software components of the system.

### 5.1.1 Available Operating System

Here are some commonly used OS options for developing a train food ordering app:

* **Android:** Android is an open-source mobile OS developed by Google. It is widely used for developing apps for smart phones and tablets. If your train food ordering app primarily targets Android devices, developing it on the Android OS would be a suitable choice.
* **iOS:** iOS is the operating system developed by Apple for its mobile devices, including iPhones and iPads. If you want to target Apple users specifically, developing the app on the iOS platform is necessary. However, keep in mind that iOS development requires using Apple's programming language, Swift, and adherence to Apple's guidelines.
* **Web-based:** Instead of developing a native app for a specific OS, you can opt for a web-based approach. In this case, the app can be accessed through a web browser, making it platform- independent. Users can access the app from any device with a web browser, including smartphones, tablets, and computers.
* **Cross-platform frameworks:** If you aim to develop the app for multiple platforms simultaneously, you can consider using cross-platform frameworks such as React Native, Flutter, or Xamarin. These frameworks allow you to write code once and deploy it on multiple platforms, including Android and iOS.

### 5.1.2 Selected Operating System

• **Android:** It is currently used in various devices such as mobiles, tablets, televisions, etc. Android provides a rich application framework that allows us to build innovative apps and games for mobile devices in a Java language environment. Every train passenger has android devices and can easily order the food by using mobile.

## 5.2 Development Approach

Development approach refers to the methodology or strategy followed during the software development process. Different development approaches provide guidelines on how to plan, design, develop, and deliver software projects.

### 5.2.1 Available Development Approach

Some common development approaches include:

* **Waterfall:** The waterfall approach follows a sequential and linear development process, with distinct phases such as requirements gathering, design, development, testing, and deployment. Each phase is completed before moving on to the next, and changes made in earlier phases may be challenging to accommodate later.
* **Agile:** Agile is an iterative and flexible development approach that emphasizes collaboration, adaptability, and customer feedback. It involves breaking down the project into small increments called sprints, where each sprint delivers a functional piece of software. Agile methodologies include Scrum, Kanban, and Extreme Programming (XP).
* **Spiral:** The spiral approach combines elements of the waterfall and iterative approaches. It involves multiple iterations of development, where each iteration builds upon the previous one. The spiral approach focuses on risk management and incorporates prototyping and customer feedback.
* **Rapid Application Development (RAD):** RAD emphasizes rapid prototyping and quick iterations to develop software. It involves close collaboration between developers and users to gather requirements, build prototypes, receive feedback, and iterate on the solution.
* **DevOps:** DevOps is an approach that emphasizes collaboration and integration between development and operations teams. It involves continuous integration, delivery, and deployment to streamline the software development lifecycle and ensure faster and more reliable releases.
* **Lean:** The lean approach aims to eliminate waste and focus on delivering value to the customer. It emphasizes efficiency, continuous improvement, and reducing unnecessary steps or activities in the development process.
* **Incremental:** The incremental approach involves dividing the project into smaller increments or modules, each of which is developed and delivered independently. This approach allows for early delivery of functioning features and facilitates ongoing customer feedback and requirements refinement

### 5.2.2 Selected Development Approach

We select waterfall approach for developing a train food ordering app:

**Requirements Gathering:** In this initial phase, gather all the necessary requirements for your train food ordering app. This includes understanding the functionalities, features, user roles, and any specific constraints or preferences.

**System Design:** Once the requirements are defined, proceed to design the overall system architecture and user interface of the app. Create detailed design specifications, including database structure, screen layouts, navigation flow, and interactions.

**Implementation:** With the design specifications in hand, start implementing the train food ordering app based on the defined requirements and system design. This involves writing code, developing the user interface, integrating external services (such as payment gateways or notification systems), and implementing the app's business logic.

**Testing:** Once the implementation is complete, thoroughly test the app to ensure it functions as intended.

Conduct different types of testing, including unit testing (testing individual components), integration testing (testing interactions between components), and system testing (testing the entire app's functionality).

Identify and fix any issues or bugs encountered during testing.

**Deployment:** After successfully testing the app and ensuring its stability, package the app for deployment. This may involve generating installation files or publishing the app on relevant app stores, such as the Google Play Store for Android devices.

**Operation and Maintenance:** Once the app is deployed, monitor its performance, gather user feedback, and address any reported issues or bugs. Perform regular maintenance tasks, including updates, security patches, and bug fixes.

## 5.3 Programming Language

A programming language is a formal language that provides a set of instructions for a computer to perform specific tasks. It allows programmers to write code and communicate with computers to develop software applications, scripts, or algorithms. Programming languages serve as a means for humans to convey their instructions to computers effectively.

### 5.3.1 Available Programming Language

* **Java:** Java is the official programming language for Android app development. It has a large community and extensive documentation, making it a popular choice. Java is known for its platform compatibility, object-oriented nature, and support for various libraries and frameworks.
* **Kotlin:** Kotlin is a modern programming language developed by JetBrains and officially supported by Google for Android app development. Kotlin offers concise syntax, improved safety features, and enhanced interoperability with Java, making it a preferred choice for many Android developers.

### 5.3.2 Selected Programming Language

* **Java:** Using Java for Android development offers several benefits:
* **Wide Adoption:** Java has been the primary language for Android app development for many years. As a result, there is a vast community of developers and extensive resources available, including libraries, frameworks, and documentation. This makes it easier to find support, solutions, and examples when working with Java for Android.
* **Platform Compatibility:** Java is known for its platform independence, and Android is no exception. Android's core libraries are built using Java, and the Android Runtime (ART) translates Java bytecode into a format that can run on Android devices. This means that Java-based Android apps can run on a wide range of Android devices, providing broad compatibility.
* **Mature Ecosystem:** Java has a mature ecosystem with a rich set of tools, libraries, and frameworks specifically tailored for Android development. Android Studio, the official integrated development environment (IDE) for Android, provides excellent support for Java. Additionally, libraries like Retrofit, Gson, and Apache HTTP Client make it easier to handle network requests, data serialization, and other common tasks.
* **Performance**: Java is known for its efficient performance, making it well-suited for resource- constrained environments like mobile devices. The Android runtime optimizes Java bytecode execution, and developers can optimize their code further by following best practices and leveraging performance optimization techniques.

## 5.4 Platform

Android is a popular operating system for mobile devices, and it continues to be a leading platform for app development. According to Market Research Future (MRFR), the market for mobile app development platforms is expected to reach USD 70.59 billion by 2030 at a CAGR of 26.46%. There are many different platforms for developing Android apps, and it can be difficult to know the best choice for your project.

### 5.1.1 Available Platform

Here are some commonly used platform options for developing a train food ordering app:

* **Android Studio:** Android Studio is the official integrated development environment (IDE) for Android app development, and it is widely considered the best choice for most developers. It is developed by Google and offers many features that make it easy to create high-quality apps.
* **Xamarin:** Xamarin is a cross-platform app development platform that allows you to create apps for Android, Windows, and iOS using a single codebase. It is based on the C# programming language and uses the .NET framework, which is popular among developers familiar with Windows development.
* **Flutter:** Flutter is an open-source app development framework created by Google. It is based on Dart, a programming language that uses a declarative style, making it easy to build high-performance apps.
* **PhoneGap:** PhoneGap is a cross-platform app development platform that allows you to create apps using HTML, CSS, and JavaScript. It is based on the Apache Cordova framework and uses a WebView to render the app, which means that you can write your app using web technologies and then package it as a native app for Android and other platforms.

### 5.1.2 Selected Platform

• **Android Studio:** Some features of Android Studio include:

* A visual layout editor that allows you to drag and drop UI elements to create the user interface for your app.
* A range of tools for debugging, testing, and optimizing your app.

o Support for a variety of programming languages, including Java, Kotlin, and C++.

* Integration with Google Play services, allowing you to easily implement features such as maps, ads, and in-app payments.
* One of the main advantages of Android Studio is that it is constantly being updated and improved by Google, ensuring that it stays updated with the latest technologies and trends.

## 5.4 Case Tool

CASE (Computer-Aided Software Engineering) tools, also known as CASE tools or CASE environments, are software applications that provide support for various activities involved in the software development process. These tools are designed to assist software developers, analysts, and project managers in tasks such as analysis, design, coding, testing, and maintenance of software systems.

### 5.4.1 Available Case tools

There are several CASE (Computer-Aided Software Engineering) tools available that can aid in the development of a train food ordering app on the Android platform. Here are some commonly used CASE tools that can be helpful in different stages of the software development lifecycle:

* **Unified Modeling Language (UML) Tools:** UML tools provide support for creating visual models and diagrams to represent the structure, behavior, and relationships of a software system. They help in designing the app's architecture, data flow, and user interface. Some popular UML tools include Enterprise Architect, Lucid chart, and Visual Paradigm.
* **Integrated Development Environments (IDEs):** IDEs are comprehensive software development environments that offer a range of tools and features to support coding, debugging, and testing. Android Studio is the official IDE for Android app development, providing essential features like code editing, compilation, debugging, and deployment.
* **Version Control Systems (VCS):** Version control systems are used to manage source code and track changes made during development. They allow developers to collaborate, manage different versions of the codebase, and handle conflicts efficiently. Git, which is widely used in conjunction with platforms like GitHub and Bitbucket, is a popular version control system.
* **Issue Tracking and Project Management Tools:** These tools help manage tasks, track issues, and facilitate collaboration within development teams. They allow you to assign tasks, set priorities, track progress, and communicate with team members. Examples of such tools include Jira, Trello, and Asana.

### 5.4.2 Selected Case Tools

• **Unified Modeling Language (UML) Tools:** Unified Modeling Language (UML) tools offer several benefits in the software development process. One major advantage is their ability to provide a visual representation of the system being developed. UML diagrams allow developers to create models and diagrams that illustrate the structure, behavior, and relationships of the software. This visual representation makes it easier for team members, stakeholders, and clients to understand and communicate complex concepts and design decisions. UML tools also provide a standardized notation, ensuring consistency and clarity in communication across the development team. Another benefit is the documentation aspect.

## 5.5 Data Base

A database is an organized collection of structured data that is stored and accessed electronically. It is designed to efficiently store, retrieve, and manage large amounts of data. Databases are widely used in various applications and industries to store and organize data in a structured manner.

### 5.5.1 Available Data Base

Here are some commonly used databases that can be suitable for a train food ordering app:

* **Relational Databases:** 
  + **MySQL:** MySQL is a popular open-source relational database management system (RDBMS) known for its scalability, performance, and wide community support. It is well-suited for applications with structured data and complex querying needs.
  + **PostgreSQL:** PostgreSQL is another powerful open-source RDBMS that offers advanced features, extensibility, and strong data integrity. It provides support for complex queries, JSON data, and spatial data, making it suitable for applications requiring advanced data manipulation capabilities.
* **NoSQL Databases:** 
  + **MongoDB:** MongoDB is a popular open-source document-oriented NoSQL database. It offers flexible schema design, scalability, and high performance for handling unstructured or semi-structured data.
  + **Firebase Real-time Database:** Firebase is a cloud-based platform that provides a NoSQL database, among other services. Firebase Realtime Database offers real-time data synchronization and offline support, making it suitable for real-time collaboration and synchronization in the train food ordering app.

### 5.5.2 Selected Database

**Firebase Real-time Database:** It offers features for backend infrastructure, real-time database, user authentication, cloud messaging, analytics, and more. This integrated approach simplifies the development process by providing a unified platform to handle multiple aspects of app functionality. Secondly, Firebase is known for its real-time database capabilities. The Firebase Real-time Database enables real-time data synchronization across clients, allowing instant updates and changes to be propagated in real-time. This is particularly useful for applications that require real-time collaboration or instant data updates, such as chat apps or collaborative document editing.

## Chapter-6

**Testing (Software Quality Attributes) 6.1 Test Case Specification**

In this portion, we will discuss the testing phase of developed application **“Train Food Ordering App”** in different manner to know that how much efficient and effective application is?

## 6.2 Black Box Test Case

**6.2.1 BVA or Boundary Value Analysis:**

BVA is Black Box Test Design Technique, which is used to find the errors at boundaries of input domain (tests the behavior of a program at the input boundaries) rather than finding those errors in the center of input. So, the basic idea in boundary value testing is to select input variable values at their: minimum, just above the minimum, just below the minimum, a nominal value, just below the maximum, maximum and just above the maximum. That is, for each range, there are two boundaries, the lower boundary (start of the range) and the upper boundary (end of the range) and the boundaries are the beginning and end of each valid partition. We should design test cases which exercise the program functionality at the boundaries, and with values just inside and outside the boundaries. Boundary value analysis is also a part of stress and negative testing.

### 6.2.2 Equivalence Class Partitioning

In this method, the input domain data is divided into different equivalence data classes – which are generally termed as ‘Valid’ and ‘Invalid’. The inputs to the software or system are divided into groups that are expected to exhibit similar behavior. Thus, it reduces the number of test cases to a finite list of testable test cases covering maximum possibilities.

### 6.2.3 State Transition Testing

This testing technique uses the inputs, outputs, and the state of the system during the testing phase. It checks the software against the sequence of transitions or events among the test data. Based on the type of software that is tested, it checks for the behavioral changes of a system in a particular state or another state while maintaining the same inputs.

### 6.2.4 Decision Table Testing

This approach creates test cases based on various possibilities. It considers multiple test cases in a decision table format where each condition is checked and fulfilled, to pass the test and provide accurate output. It is preferred in case of various input combinations and multiple possibilities.

### 6.2.5 Graph Base Testing

This technique of Black box testing involves a graph drawing that depicts the link between the causes (inputs) and the effects (output), which trigger the effects. This testing utilizes different combinations of output and inputs.

## 6.3 White Box Testing

### 6.3.1 Statement Coverage

Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code. It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed.

Using this technique we can check what the source code is expected to do and what it should not. It can also be used to check the quality of the code and the flow of different paths in the program. The main drawback of this technique is that we cannot test the false condition in it.

### 6.3.2 Branch Coverage

Branch Coverage is a white box testing method in which every outcome from a code module(statement or loop) is tested. The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once. It helps to measure fractions of independent code segments and to find out sections having no branches.

Branch Coverage testing is important in ensuring that the software is robust and that all possible paths through the application are thoroughly tested.

### 6.3.3 Path Coverage

Path coverage testing is a kind of software testing wherein the tester assesses each line of software code. There are four steps to path coverage testing—draw a control graph, calculate the program’s cyclomatic complexity, find a basis set of paths, and generate test cases to exercise each path. Path coverage testing reduces redundant tests, focuses on program logic, executes test cases at least once, and ensures the best design.

**6.4 Test Cases**

**Test Case 1:**

|  |  |
| --- | --- |
| Date: 1 Oct, 2023 |  |
| System: Android |  |
| Objective: Login as Restaurant Owner | Test ID:1 |
| Version:1 |  |
| Input:  Restaurant Owner Credentials. |  |
| Expected Result: Display Restaurant Owner dashboard |  |
| Actual Result: Passed |  |

**Test Case 2:**

|  |  |
| --- | --- |
| Date: 1 Oct, 2023 |  |
| System: Android |  |
| Objective: Login as Passenger | Test ID:2 |
| Version:1 |  |
| Input:  Passenger Credentials. |  |
| Expected Result: Display Passenger dashboard |  |
| Actual Result: Passed |  |

**Test Case 3:**

|  |  |
| --- | --- |
| Date: 1 Oct, 2023 |  |
| System: Android |  |
| Objective: Login as Delivery Boy | Test ID:3 |
| Version:1 |  |
| Input:  Delivery Boy Credentials. |  |
| Expected Result: Display Delivery Boy dashboard |  |
| Actual Result: Passed |  |

**4:**

|  |  |
| --- | --- |
| Date: 2 Oct 2023 |  |
| System: Android |  |
| Objective: Add Food As Restaurant Owner | Test ID:4 |
| Version:1 |  |
| Input:  Input food name, food description and price |  |
| Expected Result: Display Food |  |
| Actual Result: Passed |  |

**Test Case 5:**

|  |  |
| --- | --- |
| Date: 2 Oct, 2023 |  |
| System: Android |  |
| Objective: Add to cart | Test ID:5 |
| Version:1 |  |
| Input:  Click the add Cart button |  |
| Expected Result: food added to cart |  |
| Actual Result: Passed |  |

**Test Case 6:**

|  |  |
| --- | --- |
| Date: 3 Oct, 2023 |  |
| System: Android |  |
| Objective: Place Order | Test ID:6 |
| Version:1 |  |
| Input:  Click on order now button |  |
| Expected Result: order is added |  |
| Actual Result: Passed |  |

**7:**

|  |  |
| --- | --- |
| Date: 10 Oct, 2023 |  |
| System: Android |  |
| Objective: Show Order As Restaurant Owner | Test ID:7 |
| Version:1 |  |
| Input:  Click on show order |  |
| Expected Result: Display Orders |  |
| Actual Result: Passed |  |

**Test Case 8:**

|  |  |
| --- | --- |
| Date: 10 Oct , 2023 |  |
| System: Android |  |
| Objective: Show Order as Delivery Boy | Test ID:8 |
| Version:1 |  |
| Input:  Click on Show Order Button |  |
| Expected Result: Display Order |  |
| Actual Result: Passed |  |

**Test Case 9:**

|  |  |
| --- | --- |
| Date: 10 Oct, 2023 |  |
| System: Android |  |
| Objective: Show Pending Order As Delivery Boy | Test ID:9 |
| Version:1 |  |
| Input:  Click on pending order |  |
| Expected Result: Show Pending Order |  |
| Actual Result: Passed |  |

**10**

|  |  |
| --- | --- |
| Date: 11 Oct, 2023 |  |
| System: Android |  |
| Objective: Show Track Order as Passenger | Test ID:10 |
| Version:1 |  |
| Input:  Click on track order. |  |
| Expected Result: Display track orders. |  |
| Actual Result: Passed |  |

**Test Case 11**

|  |  |
| --- | --- |
| Date: 11 Oct, 2023 |  |
| System: Android |  |
| Objective: Show pending order as Restaurant Owner | Test ID:11 |
| Version:1 |  |
| Input:  Click on pending order. |  |
| Expected Result: Display pending orders. |  |
| Actual Result: Passed |  |

**Test Case 12**

|  |  |
| --- | --- |
| Date: 11Oct, 2023 |  |
| System: Android |  |
| Objective: Show pending order as Passenger | Test ID:12 |
| Version:1 |  |
| Input:  Click on pending order |  |
| Expected Result: Display pending orders. |  |
| Actual Result: Passed |  |

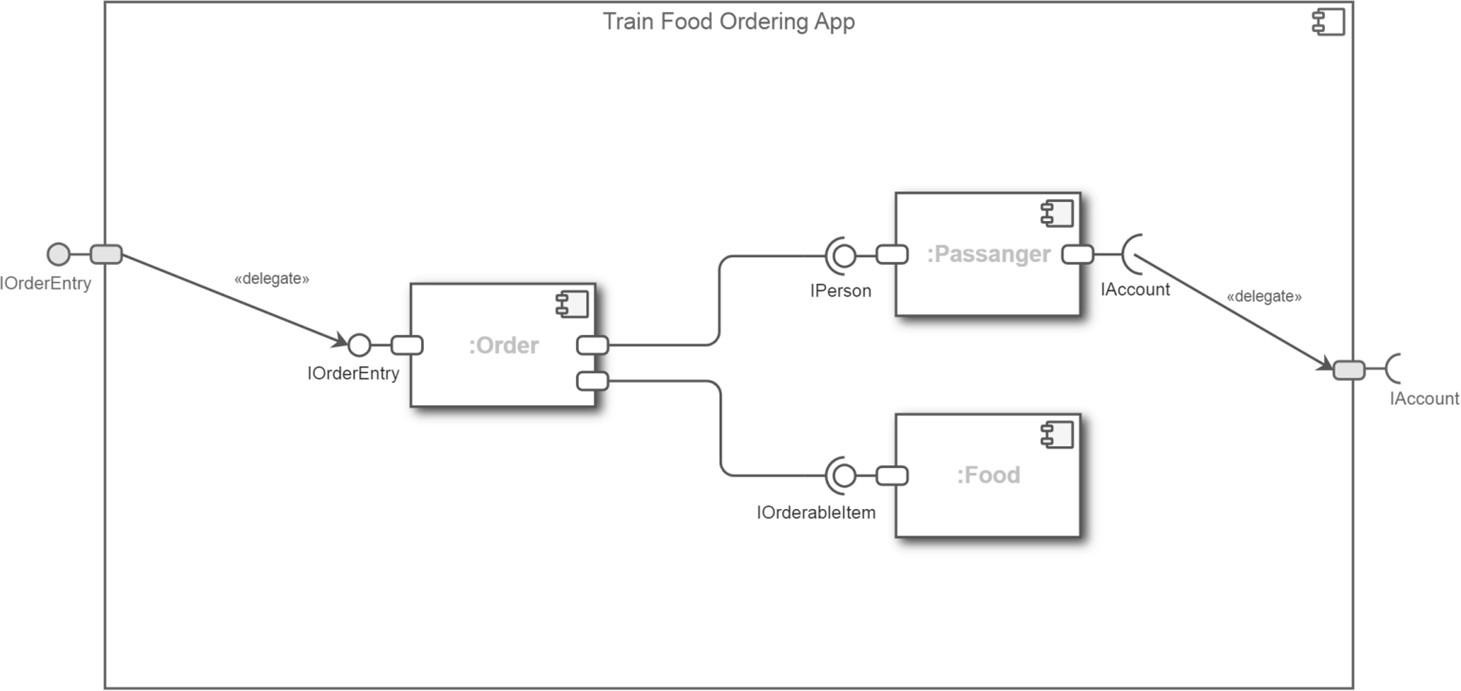
**Chapter-7 Implementation**

## 7.1 Component Diagram

### 7.1.1 Definition

A component diagram, often called a UML component diagram, shows how the physical parts of a system are wired up and organized. To represent implementation specifics and ensure that all necessary functionalities of the system are covered by planned development, component diagrams are frequently created.

### 7.1.2 Diagram



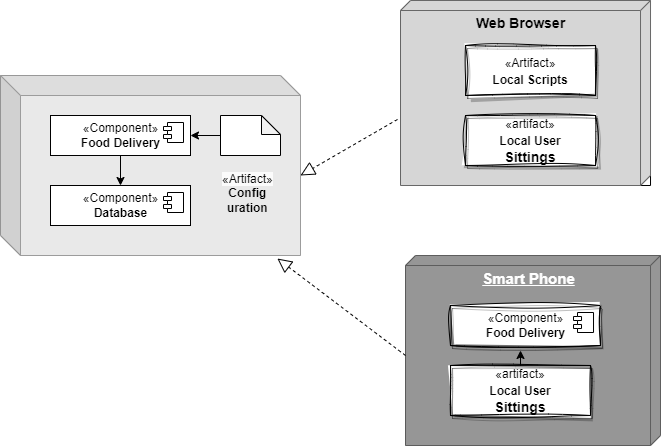
***Fig.14 Component Diagram***

## 7.2 Deployment Diagram

### 7.2.1 Definition

Deployment diagrams represent a system's physical architecture. The links between the system's hardware and software components as well as the physical distribution of the processing are displayed in deployment diagrams.

### 7.2.2 Diagram



***Fig.15 Deployment Diagram***

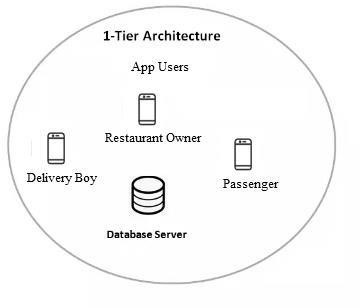
## 7.3 Database Architecture (1- Tier, 2-Tier, 3- Tier Architecture)

### 7.3.1 Definition

A DBMS design is represented by database architecture. It aids in the creation, growth, use, and maintenance of the database management system. The database system may be divided into separate parts that can be independently adjusted, changed, replaced, and altered thanks to DBMS design. Understanding a database's components is also beneficial.

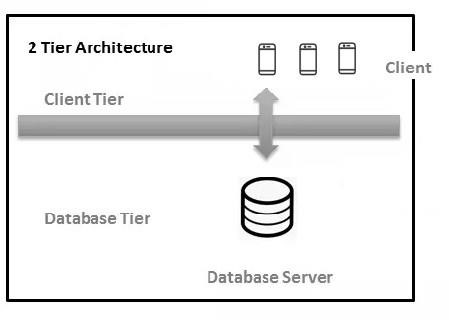
**7.3.2 Diagram**

### 1- Tier Architecture



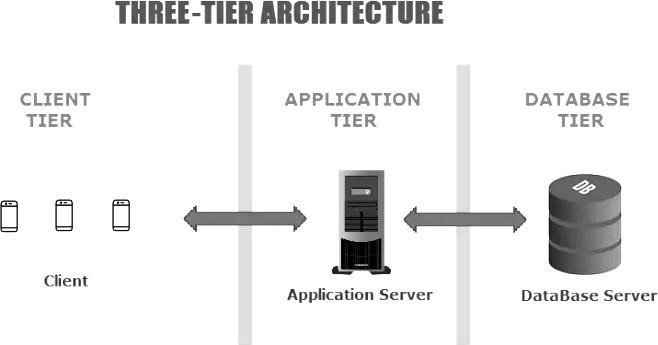
***Fig.16 1-Tier Architecture***

### 2- Tier Architecture



***Fig.17 2-Tier Architecture***

### 3- Tier Architecture



***Fig.18 3-Tier Architecture***

## Chapter-8 Tools and Technologies

**8.1 Programming Languages**

The programming language use in this application is:

* Java:

The most common language for mobile app development is Java, one of only a few that may be used to create applications for Android and iOS devices. Java's advantages over other programming languages include: It is focused on objects. Its syntax is comparable to C++, C#, and other languages.

* XML:

Extensible Markup Language (XML) is a markup language that provides rules to define any data. Unlike other programming languages, XML cannot perform computing operations by itself. Instead, any programming language or software can be implemented for structured data management.

## 8.2 Operating Environment

The operating environment for our application will be:

• Android System:

The Android operating system is most frequently used on different mobile platforms around the world. It is occupied approximately 75% of shares in the worldwide market by the end of 2020. A company like Open Handset Alliance has developed the first Android that depends on the customized version of the Linux kernel as well as other open-source software.

**APPENDIX A:**

**USER DOCUMENTATION**

## User Documentation for Restaurant Owner

### Introduction

* Welcome to the train food ordering app user documentation for restaurant owners.

* This guide will help you understand how to use the app to manage your restaurant and fulfill food orders for train passengers

### Getting Started

* Download and install the train food Ordering app from your respective app store.

* Launch the app and sign up for a new account or log in if you already have one.

* Provide your restaurant details, including name, location, contact information, and menu items.

### Menu Management

* Add, edit, or remove menu items available for train passengers.

* Include detailed descriptions, prices, and customization options for each menu item.

* Update menu availability based on stock or time restrictions.

### Order Management

* Receive and process incoming food orders from train passengers.

* Get notified of new orders in real-time and review order details including food items, quantities, and special instructions.
* Confirm or decline orders based on availability.

### Managing Special Offers and Discounts

* Create special offers or discounts for train passengers to attract more orders.

* Specify terms and conditions for promotional offers.

## User Documentation for Passenger

### Introduction

* Welcome to the train food ordering app user documentation.

* This guide will help you understand how to use the app as a passenger to order food during your train journey.

### Getting Started

* Download and install the train food Ordering app from your respective app store.

* Launch the app and sign up for a new account or log in if you already have one.

* Enter your personal details such as name, contact number, and email address.

### Menu Selection

* Explore the available restaurants and menus on the app.

* Browse through the menu items and select the desired food items by adding them to your cart.

### Order Tracking

* After placing the order, you can track its status in real-time.

* The app will provide updates on the preparation, dispatch, and estimated delivery time of your order.

### Delivery and Enjoying Your Meal

* Once the delivery personnel arrives at your train compartment, collect your order.

* Verify the contents of the order and enjoy your meal.

## User Documentation for Delivery Boy

### Introduction

* Welcome to the train food ordering app user documentation for delivery personnel.

* This guide will help you understand how to use the app to efficiently deliver food orders to train passengers.

### Getting Started

* Download and install the train food Ordering app from your respective app store.

* Launch the app and sign up for a new account or log in if you already have one.

* Provide your personal details, including name, contact information, and identification documents.

### Accepting Orders

* Receive notifications for new delivery orders in real-time.

* Review order details, including the pickup location, train number, and delivery instructions.

### Pickup and Delivery Process

* Arrive at the designated restaurant to pick up the food order.

* Verify the order details and ensure all items are packed correctly.

* Navigate to the assigned train and compartment using the provided directions and map integration.

### Delivery Confirmation

* Upon reaching the train compartment, confirm your arrival to the passenger through the app.

* Handover the order to the passenger, ensuring it matches the details mentioned in the app.

* Collect any payment due, if applicable, based on the app's instructions.

### Order Tracking and Updates

* Keep the app updated with your progress during the delivery process.

* Update the order status as "picked up," "en route," and "delivered" in real-time.

## APPENDIX B: Source code

**Code:**

packagecom.food\_on.app.ChefFoodPanel; public class ChefFinalOrders {

private String ChefId,DishId,DishName,DishPrice,DishQuantity,RandomUID,TotalPrice,UserId;

public ChefFinalOrders(String chefId, String dishId, String dishName, String dishPrice, String

dishQuantity, String randomUID, String totalPrice, String userId) {

ChefId = chefId;

DishId = dishId;

DishName = dishName;

DishPrice = dishPrice;

DishQuantity = dishQuantity;

RandomUID = randomUID;

TotalPrice = totalPrice;

UserId = userId;

}

public ChefFinalOrders()

{

}

public String getChefId() { return ChefId;

}

public void setChefId(String chefId) {

ChefId = chefId;

}

public String getDishId() { return DishId;

}

public void setDishId(String dishId) {

DishId = dishId;

}

public String getDishName() { return DishName;

}

public void setDishName(String dishName) {

DishName = dishName;

}

public String getDishPrice() { return DishPrice;

}

public void setDishPrice(String dishPrice) {

DishPrice = dishPrice;

}

public String getDishQuantity() { return DishQuantity;

}

public void setDishQuantity(String dishQuantity) {

DishQuantity = dishQuantity;

}

public String getRandomUID() { return RandomUID;

}

public void setRandomUID(String randomUID) {

RandomUID = randomUID;

}

public String getTotalPrice() { return TotalPrice;

}

public void setTotalPrice(String totalPrice) {

TotalPrice = totalPrice;

}

public String getUserId() { return UserId;

}

public void setUserId(String userId) {

UserId = userId;

}

}

package com.food\_on.app.CustomerFoodPanel;

import android.content.Context; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.cepheuen.elegantnumberbutton.view.ElegantNumberButton; import com.food\_on.app.R;

import com.google.firebase.auth.FirebaseAuth;

import com.google.firebase.database.FirebaseDatabase;

import java.util.HashMap; import java.util.List;

public class CustomerCartAdapter extends RecyclerView.Adapter<CustomerCartAdapter.ViewHolder> {

private Context mcontext; private List<Cart>cartModellist; static int total = 0;

public CustomerCartAdapter(Context context, List<Cart>cartModellist) { this.cartModellist = cartModellist; this.mcontext = context; total = 0;

}

@NonNull @Override

public ViewHolderonCreateViewHolder(@NonNullViewGroup parent, intviewType) {

View view = LayoutInflater.from(mcontext).inflate(R.layout.cart\_placeorder, parent, false); return new CustomerCartAdapter.ViewHolder(view);

}

@Override

public void onBindViewHolder(@NonNull final ViewHolder holder, int position) { final Cart cart = cartModellist.get(position); holder.dishname.setText(cart.getDishName());

holder.PriceRs.setText("Price: ₹ " + cart.getPrice()); holder.Qty.setText("× " + cart.getDishQuantity()); holder.Totalrs.setText("Total: ₹ " + cart.getTotalprice()); total += Integer.parseInt(cart.getTotalprice()); holder.elegantNumberButton.setNumber(cart.getDishQuantity()); final intdishprice = Integer.parseInt(cart.getPrice()); holder.elegantNumberButton.setOnValueChangeListener(new

ElegantNumberButton.OnValueChangeListener() {

@Override

public void onValueChange(ElegantNumberButton view, intoldValue, intnewValue) { intnum = newValue;

inttotalprice = num \* dishprice;

if (num != 0) {

HashMap<String, String>hashMap = new HashMap<>(); hashMap.put("DishID", cart.getDishID()); hashMap.put("DishName", cart.getDishName()); hashMap.put("DishQuantity", String.valueOf(num)); hashMap.put("Price", String.valueOf(dishprice)); hashMap.put("Totalprice", String.valueOf(totalprice)); hashMap.put("ChefId",cart.getChefId());

FirebaseDatabase.getInstance().getReference("Cart").child("CartItems").child(FirebaseAuth.getInstance().ge tCurrentUser().getUid()).child(cart.getDishID()).setValue(hashMap);

} else {

FirebaseDatabase.getInstance().getReference("Cart").child("CartItems").child(FirebaseAuth.getInstance().ge tCurrentUser().getUid()).child(cart.getDishID()).removeValue();

}

}

});

CustomerCartFragment.grandt.setText("Grand Total: ₹ " + total); FirebaseDatabase.getInstance().getReference("Cart").child("GrandTotal").child(FirebaseAuth.getInstance().

getCurrentUser().getUid()).child("GrandTotal").setValue(String.valueOf(total));

}

@Override

public intgetItemCount() { return cartModellist.size();

}

public class ViewHolder extends RecyclerView.ViewHolder {

TextViewdishname, PriceRs, Qty, Totalrs;

ElegantNumberButtonelegantNumberButton;

public ViewHolder(@NonNull View itemView) { super(itemView);

dishname = itemView.findViewById(R.id.Dishname);

PriceRs = itemView.findViewById(R.id.pricers);

Qty = itemView.findViewById(R.id.qty); Totalrs = itemView.findViewById(R.id.totalrs);

elegantNumberButton = itemView.findViewById(R.id.elegantbtn);

}

}

}

package com.food\_on.app;

public class Customer {

private String

City,ConfirmPassword,EmailID,FirstName,LastName,Mobileno,Password,State,Suburban,LocalAddress;

public Customer() {

}

public Customer(String City, String confirmPassword, String emailID, String firstName,StringlastName, String mobileno, String password, String state, String suburban,StringlocalAddress) { this.City = City;

ConfirmPassword=confirmPassword;

EmailID = emailID;

FirstName=firstName;

LastName=lastName;

Mobileno = mobileno;

Password = password; State = state;

Suburban = suburban;

LocalAddress=localAddress;

}

public String getCity() { return City;

}

public void setCity(String city) {

City = city;

}

public String getConfirmPassword() { return ConfirmPassword;

}

public void setConfirmPassword(String confirmPassword) {

ConfirmPassword = confirmPassword;

}

public String getEmailID() { return EmailID;

}

public void setEmailID(String emailID) {

EmailID = emailID;

}

public String getFirstName() { return FirstName;

}

public void setFirstName(String firstName) {

FirstName = firstName;

}

public String getLastName() { return LastName;

}

public void setLastName(String lastName) {

LastName = lastName;

}

public String getMobileno() { return Mobileno;

}

public void setMobileno(String mobileno) {

Mobileno = mobileno;

}

public String getPassword() { return Password;

}

public void setPassword(String password) {

Password = password;

}

public String getState() { return State;

}

public void setState(String state) {

State = state;

}

public String getSuburban() { return Suburban;

}

public void setSuburban(String suburban) {

Suburban = suburban;

}

public String getLocalAddress() { return LocalAddress;

}

public void setLocalAddress(String localAddress) {

LocalAddress = localAddress;

}

}

package com.food\_on.app;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AlertDialog;

import androidx.appcompat.app.AppCompatActivity;

import android.animation.Animator;

import android.animation.AnimatorListenerAdapter; import android.content.DialogInterface; import android.content.Intent; import android.os.Bundle; import android.os.Handler; import android.util.Log; import android.widget.ImageView; import android.widget.TextView; import android.widget.Toast;

import com.food\_on.app.R;

import com.google.android.gms.ads.AdError; import com.google.android.gms.ads.AdRequest; import com.google.android.gms.ads.FullScreenContentCallback; import com.google.android.gms.ads.LoadAdError;

import com.google.android.gms.ads.MobileAds;

import com.google.android.gms.ads.initialization.InitializationStatus;

import com.google.android.gms.ads.initialization.OnInitializationCompleteListener; import com.google.android.gms.ads.interstitial.InterstitialAd; import com.google.android.gms.ads.interstitial.InterstitialAdLoadCallback; import com.google.firebase.auth.FirebaseAuth; import com.google.firebase.database.DataSnapshot; import com.google.firebase.database.DatabaseError; import com.google.firebase.database.DatabaseReference; import com.google.firebase.database.FirebaseDatabase; import com.google.firebase.database.ValueEventListener;

public class MainActivity extends AppCompatActivity {

FirebaseAuthFauth;

FirebaseDatabasefirebaseDatabase;

DatabaseReferencedatabaseReference;

ImageViewimageVieww;

TextViewtextView;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main); imageVieww=(ImageView)findViewById(R.id.imageView); textView=(TextView)findViewById(R.id.textView7); imageVieww.animate().alpha(0f).setDuration(0); textView.animate().alpha(0f).setDuration(0); imageVieww.animate().alpha(1f).setDuration(1000).setListener(new AnimatorListenerAdapter() {

@Override public void onAnimationEnd(Animator animation) { textView.animate().alpha(1f).setDuration(800);

}

});

MobileAds.initialize(this, new OnInitializationCompleteListener() {

@Override

public void onInitializationComplete(InitializationStatusinitializationStatus) {

}

});

new Handler().postDelayed(new Runnable() {

@Override

public void run() {

Fauth = FirebaseAuth.getInstance();

if (Fauth.getCurrentUser() != null) {

if (Fauth.getCurrentUser().isEmailVerified()) {

Fauth = FirebaseAuth.getInstance();

databaseReference =

FirebaseDatabase.getInstance().getReference("User").child(FirebaseAuth.getInstance().getUid() + "/Role"); databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {

@Override

public void onDataChange(@NonNullDataSnapshotdataSnapshot) { String role = dataSnapshot.getValue(String.class);

if (role.equals("Customer")) {

Intent n = new Intent(MainActivity.this,

CustomerFoodPanel\_BottomNavigation.class); startActivity(n); finish();

}

if (role.equals("Chef")) {

Intent a = new Intent(MainActivity.this, ChefFoodPanel\_BottomNavigation.class);

startActivity(a); finish();

}

if (role.equals("DeliveryPerson")) {

Intent intent = new Intent(MainActivity.this,

Delivery\_FoodPanelBottomNavigation.class); startActivity(intent);

finish();

}

}

@Override

public void onCancelled(@NonNullDatabaseErrordatabaseError) {

Toast.makeText(MainActivity.this, databaseError.getMessage(), Toast.LENGTH\_LONG).show();

}

});

} else {

AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this); builder.setMessage("Check whether you have verified your details, Otherwise please verify"); builder.setCancelable(false); builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

dialog.dismiss();

Intent intent = new Intent(MainActivity.this, MainMenu.class);

startActivity(intent);

finish();

}

});

AlertDialog alert = builder.create(); alert.show();

Fauth.signOut();

}

} else {

Intent intent = new Intent(MainActivity.this, MainMenu.class);

startActivity(intent);

finish();

}

}

}, 3000);

}

}

package com.food\_on.app;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent; import android.os.Bundle; import android.util.Log; import android.view.View; import android.view.animation.Animation; import android.view.animation.AnimationUtils; import android.widget.Button; import android.widget.ImageView; import android.widget.Toast;

import com.food\_on.app.R;

import com.google.android.gms.ads.AdError; import com.google.android.gms.ads.AdRequest; import com.google.android.gms.ads.FullScreenContentCallback; import com.google.android.gms.ads.LoadAdError;

import com.google.android.gms.ads.interstitial.InterstitialAd;

import com.google.android.gms.ads.interstitial.InterstitialAdLoadCallback;

public class MainMenu extends AppCompatActivity {

Button signinemail, signinphone, signup;

ImageViewbgimage;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main\_menu);

final Animation zoomin = AnimationUtils.loadAnimation(this, R.anim.zoomin); final Animation zoomout = AnimationUtils.loadAnimation(this, R.anim.zoomout); bgimage = findViewById(R.id.back2); bgimage.setAnimation(zoomin); bgimage.setAnimation(zoomout);

// loadAd();

zoomout.setAnimationListener(new Animation.AnimationListener() {

@Override

public void onAnimationStart(Animation animation) {

}

@Override

public void onAnimationEnd(Animation animation) { bgimage.startAnimation(zoomin);

}

@Override

public void onAnimationRepeat(Animation animation) {

}

});

zoomin.setAnimationListener(new Animation.AnimationListener() {

@Override public void onAnimationStart(Animation animation) {

}

@Override

public void onAnimationEnd(Animation animation) { bgimage.startAnimation(zoomout);

}

@Override

public void onAnimationRepeat(Animation animation) {

}

});

signinemail = (Button) findViewById(R.id.SignwithEmail); signinphone = (Button) findViewById(R.id.SignwithPhone);

signup = (Button) findViewById(R.id.SignUp);

signinemail.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent signemail = new Intent(MainMenu.this, ChooseOne.class); signemail.putExtra("Home", "Email");

startActivity(signemail); finish();

}

});

signinphone.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent signphone = new Intent(MainMenu.this, ChooseOne.class); signphone.putExtra("Home", "Phone"); startActivity(signphone); finish();

}

});

signup.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent signup = new Intent(MainMenu.this, ChooseOne.class); signup.putExtra("Home", "SignUp"); startActivity(signup); finish();

}

});

}

@Override

protected void onDestroy() {

super.onDestroy();

System.gc();

}

}

## APPENDIX C: References

1. "IRCTC Food on Track" App: This is an official Indian Railways Catering and Tourism Corporation (IRCTC) app that allows passengers to order food for train journeys in India. The app is available on both Android and iOS platforms.

1. "Travel Khana" App: travel Khana is another popular train food ordering app in India that offers a wide range of food options for train travelers. The app covers a large number of train stations across the country.

1. "RailRestro" App: RailRestro is a food delivery platform designed specifically for train passengers in India. It allows users to order food from their favorite restaurants and get it delivered to their train seat.

1. "SnackLok" App: SnackLok is a train food delivery service in the United States that enables passengers to order food and beverages from local restaurants for delivery to their train seat.

1. "Caterwings" App: Caterwings is a food delivery app that partners with various railway companies and offers catering services for train journeys in Europe.

1. "Eat2Save" App: Eat2Save is a UK-based food delivery platform that provides food delivery services for passengers on selected train routes.

1. "The Food Sharing Revolution: How Start-Ups, Pop-Ups, and Co-Ops are Changing the Way We Eat" by Michael S. Carolan

1. "Delivering the Goods: A History of the Pizza Pioneers Who Invented the Delivery Business" by Richard B. Newman