ARBA MINCH UNIVERSITY

ARBA MINCH INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND SOFTWARE ENGINEERING

Cloud Based Food Ordering and Delivery System

Group Members

No.	Name	ID
1	Abraham Mebrate	Eitm/ur158942/11
2	H/Mariam Assefa	Eitm/ur156064/11
3	Mered Gugsa	Eitm/ur156207/11
4	Nahusenay Tadesse	Eitm/ur158884/11
5	Yared Tadesse	Eitm/ur158391/11

A senior project

Submitted to Faculty of Computing and Software Engineering, AMIT, Arba Minch University, in Partial fulfillment for the requirement of the Degree of Bachelor Science in (Software Engineering)

Advisor's name*:** Mr. Nathnael Tefera ***Signature: ___________

Mar 2023

Arba Minch, Ethiopia

Submited by:

1. Abraham Mebrate	________________	________________
	Signature	Date
2. H/Mariam Assefa	________________	________________
	Signature	Date
3. Mered Gugsa	________________	________________
	Signature	Date
4. Nahusenay Tadesse	________________	________________
	Signature	Date
5. Yared Tadesse	________________	________________
	Signature	Date

Approved by

1. Advisor Name: Mr. Nathnael Tefera ________________ ________________

Signature Date

Declaration

We the undergraduates declare that this project is our work and has not been presented for a degree program in any other university. In addition to this we also declare that all sources of materials used for this thesis have been duly acknowledged.

Declared by:

1. Abraham Mebrate	__________________	__________________
	Signature	Date
2. H/Mariam Assefa	__________________	__________________
	Signature	Date
3. Mered Gugsa	__________________	__________________
	Signature	Date
4. Nahusenay Tadesse	__________________	__________________
	Signature	Date
5. Yared Tadesse	__________________	__________________
	Signature	Date

Confirmed by advisor:

Name: Mr. Nathnael Tefera

Signature: _______________________

Date: ____________________________

Acknowledgment

First of all, we would like to thank the almighty God for helping in every step of our journey and hearing our prayers. Secondly, we would like to thank Arba Minch University Institute of Technology Faculty of Computer Science and Software Engineering for giving the chance to do our senior project. We would also like to thank our wonderful advisor Mr. Nathnael Tefera for giving us priceless advice that helped us superbly in every end of the project. Lastly, we would like to thank Miss Frehiwot Niguse from beU food delivering service for helping us immensely in gathering data on our subject.

Table of Contents

List of tables viii
List of figures ix
Abstract xi
List of Abbrivations xii
Chapter 1 1
1. Introduction 1
1.1 Background 1
1.2 Statement of the Problem 1
1.3 Purpose of the project 2
1.4 Objective 2
1.5 Feasibility Study 3
1.6 Scope and Limitation 4
1.7 Significance of the project 5
1.8 Target beneficiaries of the project 6
1.9 Requirement specification 7
1.10 Methodology 8
1.11 Testing plan 9
1.12 Task and schedule 10
1.13 Team Composition 11
Chapter 2 12
2. Description of the Existing System 12
2.1 Introduction of Existing System 12
2.2 Players in the Existing System 13
2.3 Major Functions/Activities in the Existing System 14

2.4 Business Rules 14
2.5 Bottlenecks of the Existing System 15
2.6 Forms and Other Documents of the Existing Systems 16
2.7 Practices to be preserved 19
Chapter 3 20
3. Proposed System 20
3.1 Overview 20
3.2 Functional requirements 20
3.3 Non-Functional Requirement 21
3.4 Performance Requirements 22
3.5 Identification of actors and use cases 23
3.6 Use Case Model 23
3.7 Class Diagram 38
3.8 Sequence Diagram 39
3.9 Activity Diagram 47
3.10 User Interface Prototype 53
Chapter 4 56
4. System Design 56
4.1 Introduction 56
4.2 Class type architecture 56
4.3. Class modeling 58
4.4. State chart diagram 58
4.5. Collaboration Diagram 63
4.6. Component diagram 65
4.7. Deployment diagram 66

4.8. Persistence modeling 67
4.9. User Interface design 68
Chapter 5 76
5. Implementation and Testing 76
5.1 Introduction 76
5.2 Implementation 76
5.2.1 Hardware and Software Acquisitions 77
5.2.2 Code Snippets 79
5.3 Testing 82
Chapter 6 102
6. Conclusions and Recommendation 102
6.1. Conclusion 102
6.2 .Recommendation 103
References 104

List of tables

Table 1: Team Composition....11
Table 2: Register Use Case Description....25
Table 3: Log in Use Case Description....25
Table 4: Manage Profile Use Case Description....26
Table 5: Accept Order Use Case Description....27
Table 6: Decline Order Use Case Description....27
Table 7: Change Order Status Use Case Description....28
Table 8: View Order History Use Case Description....29
Table 9: Search Dish Use Case Description....30
Table 10 Select Dish Use Case Description....31
Table 11: Manage Basket Use Case Description....31
Table 12: Add Dish Use Case Description....32
Table 13: Remove Dish Use Case Description....32
Table 14: Place Order Use Case Description....33
Table 15: Make Payment Use Case Description....34
Table 16: Track Order Use Case Description....34
Table 17: Add Dish Use Case Description....35
Table 18: Remove Dish Use Case Description....36
Table 19: Manage Restaurant Use Case Description....36
Table 20: Manage Courier Use Case Description....37
Table 21 Login testcase....85
Table 22 Sign-up testcase....86
Table 23 Search dish testcase....88
Table 24 Seach Dish Testcase....89
Table 25 Manage Basket Testcase....91
Table 26 Place Order Testcase....93
Table 27 Track Order Testcase....96

List of figures

Figure 1 Gantt Chart 10
Figure 2: beU customer receipt 19
Figure 3 Use Case Diagram 24
Figure 4 Class diagram of the food delivery system 38
Figure 5: Log in Sequence Diagram 39
Figure 6: Manage Restaurant Sequence Diagram 40
Figure 7: Manage Basket Sequence Diagram 41
Figure 8: Add Dish Sequence Diagram 42
Figure 9: Remove Dish Sequence Diagram 43
Figure 10: Place Order Sequence Diagram 44
Figure 11 Change Order Status Sequence Diagram 45
Figure 12: Track order Sequence Diagram 46
Figure 13: Register Activity Diagram 47
Figure 14: Log in Activity Diagram 48
Figure 15: Manage Restaurant Activity Diagram 49
Figure 16: Manage Courier Activity Diagram 50
Figure 17: Accept or Decline Order Activity Diagram 51
Figure 18: Change Status Activity Diagram 52
Figure 19 UI prototyping for customer app 53
Figure 20: UI prototyping for restaurant 54
Figure 21 UI Prototyping for Courier 54
Figure 22 UI Prototyping for Administrator 55
Figure 23: System architecture 57
Figure 24: Class diagram 58
Figure 25: State chart diagram for login 59
Figure 26: State chart diagram for add dish 60
Figure 27: State chart diagram for place order 61
Figure 28: State chart diagram for track order 62
Figure 29: Collaboration Diagram of the System 64
Figure 30: Collaboration Diagram for login 64

Figure 31: Component diagram 65
Figure 32:- Deployment diagram for Websites 66
Figure 33 Persistence Diagram 67
Figure 34: Home screen for the user app 68
Figure 35 Orders page for the user app 69
Figure 36 Profile screen for the courier app 70
Figure 37 Order Detail....71
Figure 38 Profile screen for user app 72
Figure 39 Order Details screen of Restaurant web app....73
Figure 40 Manage User payment screen of Admin Web app....74
Figure 41 Manage Restaurant screen of admin web app 75
Figure 42 Order list screen of the restaurant web app....75
Figure 43 Profile Screen....79
Figure 44 Payment Screen 80
Figure 45 Home Screen....81

Abstract

Unfortunate circumstances like disability, old age, pandemics and busyness, prevent many from going out and getting food. An online food ordering and delivering system can help those who find themselves in those unfortunate situations. Cloud Based Food Ordering and Delivery System is a project that aims to make online food ordering and delivering systems more efficient and more modernized. The system serves as a cloud based communication platform for customers, food vendors and food couriers to order and deliver food quickly and effectively. The project is implemented by using the mobile applications and the web applications used to build the necessary structures that make up the system. The mobile applications are developed using React Native. The web applications are developed using React. The database system and overall backend are developed using Amazon Web Services (AWS). For location services the Google Console Directions API is used.

Keyword: Cloud Based, Mobile Application, Web Application

List of Abbrivations

API	Application Programming Interface
AWS	Amazon Web Service
GPS	Global Positioning System
UI	User Interface
UML	United Modeling Language

Chapter 1

1. Introduction

1.1 Background

The first food order and delivery were recorded was in 1889 when the visiting royal couple of Savoy wanted to try the food of the people of Naples. [1] Since then it has evolved, making use of new technologies. From orders being taken by phone calls to now through the internet.

Nowadays food is ordered online and using mobile and web applications. Companies like Uber Eats and Grubhub operate in many major cities making it possible for users to choose, order and track food from wherever they are. These companies even managed to become the major source of food for many during the Covid 19 pandemic. [2]

These online food ordering and delivering companies have also been popping up in Addis Ababa in recent years. beU, Moteregnaw and Deliver Addis being one of the many that has sprung up around the city. The number of companies in the industry are rapidly rising in Addis Ababa. [3] This project aims to stand out amongst them.

By using a cloud-based system and GPS tracking we plan to make a food ordering and delivery system that connects hotels, restaurants, bakeries and any other institution that offers food with couriers and customers.

The system acts as a mediator between the food service providers, the couriers and the customers to create a cohesive environment that brings these entities together where every party is satisfied. The grand aim of the whole project is to make a comprehensive system that will revolutionize food delivery in Addis Ababa.

1.2 Statement of the Problem

Modern cities like New York, London, Beijing and other major cities of the world have become more and more modernized [4]. Their economy is shifting away from physical location and toward the digital and internet. Ethiopian cities, on the other hand, lack any form of delivery service, much alone a cloud-based food delivery system. It's reasonable to state that cloud-based food delivery services do not exist in Ethiopia's capital. The few food delivery applications and systems that exist in our nation, which can be counted on one hand, are not cloud-based and are reliant on hired people tasked with delivering the food, limiting the system's potential to the employees' ability.

It's tough for an Addis Ababa resident to order food from their favorite restaurant using an app and have it delivered to their front door. There is no comprehensive list or catalog of restaurants from which to locate and select the preferred eatery. Customers can't track their food while it's being delivered to their house using existing applications and systems, and there are no payment options available that assure a safe and secure online transaction.

1.3 Purpose of the project

The main purpose is to increase efficiency, make the ordering process a reliable and convenient transaction for both the customer and the service provider, minimize the manual data entry and ensure data accuracy and security during order placement process.

1.4 Objective

1.4.1 General Objective

The general objective of the project is to develop a cloud-based food delivery System

1.4.2 Specific Objectives

To accomplish the above general objective, we have applied the following specific objectives:

- Gather data from potential users to identify the requirements
- Examine current systems to see how they can be improved
- Design a system that absolves the shortcomings found in examining the current system
- Test component in the system during step of development

1.5 Feasibility Study

1.5.1 Technical feasibility

Most people these days have a smart phone and have a knowhow about using them, like any other system. Since we plan to design a user interface that is easy to learn and use for everybody as far as possible by following best UI design principles and guidelines that are widely used by industry professionals. In order to build this application, we are using AWS (Amazon Web Service) for the back-end and React and React native for the front end. These technologies are the latest available tools that are efficient and the most fitting to our vision. We chose React Native because React Native is great for mobile apps. It provides a slick, smooth and responsive user interface, while significantly reducing load time. It's also much faster and cheaper to build apps in React Native as opposed to building native ones, without the need to compromise on quality and functionality. [5] And for the back-end we chose AWS because our system is cloud based, it is available in 44 different zones in 16 geographical locations. You can access the servers from any country you want. AWS, as a cloud service, offers enhanced security to protect your virtual organization. The data centers and servers have multiple layers of operational security. Auto Scaling and Elastic Load Balancing are two AWS tools that make scaling easier while delivering high performance. You can scale up or down based on your requirements. Finally, one of the most important AWS benefits is its flexibility. It enables you to select your operating system, programming language, web application platform, database, and other services that you may need. [6]

1.5.2 Operational feasibility

The system is implemented in a smart phone application form for all entities involved except for administrators and restaurant. It has a common, easy to use and effortlessly understandable interface for any user that has had any experience with other smartphone applications or web application before. It incorporates a log in page that the user needs to sign up with a password and a preferred account (phone number or email address), when the user starts off. After the first log in, the user has no need to log in again on the same device unless specific situations relating to the device arise. The user would also need to log in if devices are changed. The account is be valid in infinite number of devices.

1.5.3 Economic Feasibility

To check if our system is economically feasible first, we have to consider the cost and benefit ratio. The cost that we provide for the system are minimum because most of the service we used are either free or have a student account or their paying system is with "Pay as you go" approach which allows you to easily adapt to changing business needs without over-committing budgets and improving your responsiveness to changes. With a pay-as-you-go model, you can adapt your business depending on need and not on forecasts, reducing the risk of over provisioning or missing capacity. Which is a great condition for us because we built this system for study purposes but if we are going to deploy this system to a real world, we won't have to spend a lot of money from investor for the system deployment because the system will make us the money as it progresses and we will get charged based on our number of customers. One last thing is that we are using AWS which makes our system a server-less system which will decrease our total cost because we don't have to have our local server that manages and gives services.

After Deploying the system our main job is completed. The only thing that is left to do is administrator the system which requires a computer and an office or a customer service center which does fraud detection, order issues, payment fraud, incentive abuse by courier or customer, collect feedback and rating.

1.6 Scope and Limitation

1.6.1 Scope

The scope of this system includes allowing customers to request food delivery, ordering food from nearby hotels, and delivering the food to the customer who requested the service. This system is being introduced in Addis Ababa, Ethiopia's capital city. This is because a third-party firm based solely in Addis Ababa provides the system with numerically encoded positional data, such as street and address house numbers. In addition to this 3rd party organization, we also employ Amazon's web service to deploy cloud computing in the system.

1.6.2 Limitation of the project

Some limitations of the system are:

- The service is only available in Addis Ababa because the positioning system is only available there.
- Because the system is a cloud-based system offered by Amazon Web Service, a fee will be calculated.
- Risk of power outages and loss of internet connection.

1.7 Significance of the project

The internet and technological advancements are having a great impact on people spending a big part of their day on the internet, it provides a huge market potential for restaurants.

As the research shows in the year 2020/21, 20% of Ethiopian people use internet [7], and these behaviors continue to affect how businesses operate in the restaurant industry. It provides a huge market potential for restaurants. In fact, the demand for online restaurant ordering continues to grow among restaurant consumers and restaurant owners consistently look for solutions that allow customers to place orders online and have food delivered fast. An online ordering and delivery system is crucial for the modern restaurant, it contributes to higher sales volumes and greater customer satisfaction.

Creating and maintaining a great-looking menu that entices clients to purchase from you every time they see it is far easier and much less expensive (or even free). Restaurants not only free themselves from the hassle of printing and publishing expenses, but they also gain a great deal of freedom in modifying the menu whenever they choose. Furthermore, with a superb online menu, up-selling is automatic, which means that if customers see the dessert page, there is a higher probability they’ll order it along with their main meal.

In any restaurant, whether an order is accepted over the phone or in person, miscommunications can occur. Honest errors can result in spoiled food and, more crucially, irritated customers who may not return. For restaurants, one of the advantages of cloud-based ordering is that the consumer has more control. When the client is in charge, they feel more in control of their activities and have a better grasp of what they are doing. There's no risk of a mix-up because everything is written down.

Lastly cloud-based food ordering and delivery systems would greatly reduce the time and effort required to find foods at a reasonable price and in a convenient location, as well as provide work for bicycle, bike and car owners looking for delivery services.

1.8 Target beneficiaries of the project

The project benefits everyone in the entire supply chain. Restaurants get to get to sell more and courier who deliver the food will have a cut from every delivery.

Restaurants get to present their products to a much larger customer base than if they only use manual system. Online customers get much more convenience, access to a larger selection of foods, and more competitive prices.

Targeted Beneficiaries

- **Food vendors/Restaurants:**

Restaurants are the main targeted entities that are expected to benefit most through this project.

- **Couriers**

Couriers are also a major beneficiary of the project, they'll receive money for the service that they offer, and the tariffing system is in accordance to the kilometers they've covered in order to deliver the food.

- **Customer:**

The customers gets to choose foods from diverse choice of food vendors which in turn save inappropriate wastage of time that's caused by manual searching of food.

1.9 Requirement specification

1.9.1 Hardware specification

We built two mobile applications and two web application. The mobile applications are for the customer and courier while the web application are for the restaurant and the administrator.

For restaurant dashboard system and Administrator

The system can be accessed from a mobile, tablet or computer (recommended) because there may be an ordering traffic the device should be high end.

- Processor: Minimum 1 GHz; Recommended 2GHz or more
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- Hard Drive: Minimum 32 GB (for tablet or computer); Recommended 64 GB or more
- Memory (RAM): Minimum 1 GB; Recommended 4 GB or above

For the Customer app

- An android or an IOS integrated mobile devices can be used to use our services.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi) is also needed.
- The customer device must have a GPS module.

There is no absolute hardware requirement that must be included for the customer app because most of the things are done in the back end the customer app only use to render the services and take an input from the user.

For the Courier app

- An android or an IOS integrated mobile devices can be used to use our services.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi) is also needed.
- The deliverer device must have a GPS module as most mobile devices these days have.

1.9.2 Software specification

The tools and software's used in this project are: -

- Lucid charts: To make all the UML Diagrams and charts.
- Microsoft word: To make the documentation of the system.
- React Native: React Native is an open-source JavaScript framework, designed for building apps on multiple platforms like iOS, Android, and also web applications, utilizing the very same code base. It is based on React, and it brings all its glory to mobile app development.
- React: React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application.
- Visual Studio Code: Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.
- Amazon Web Service (AWS): In particular we use AWS Amplify to have the tools and features that lets frontend web and mobile developers quickly and easily build full-stack applications on AWS, with the flexibility to leverage the breadth of AWS services as our use cases evolve.

1.10 Methodology

Choosing the right software development methodology depends on your team structure, experience, project requirements, goals, budget, and other underlying factors. After considering this we have chosen Scrum Development Methodology as our methodology because Scrum is arguably one of the most flexible software development methodologies available. It is based on the Agile Development Methodology philosophy and is favored for its incremental and iterative approaches. The Scrum methodology involves the Product Owner, Scrum Master, and the Development Team. The product owner takes input from the client and ensures that the team is on track in fulfilling the client's requirements. Meanwhile, the Scrum Master acts as a facilitator and ensures that team members are familiar with the Scrum process. The team takes charge of executing the development. What makes Scrum an ideal methodology in a fast-paced environment is how tasks are executed in sprints. Each sprint takes up to 4 weeks. The speedy execution allows teams to identify issues, introduce solutions, test, and gather feedback in a short period. It makes tackling fast-paced projects much easier.

Some of the advantage of using Scrum as a methodology

- Short iterations allow quick resolutions to problems.
- Scrum is very responsive to changes as the process includes regular feedback.
- Scrum is economical and effective.
- Regular meetings ensure that team members are on the same page at all times.
- Contributions of individual members are noticed and appreciated through the Scrum meetings. [8]

1.11 Testing plan

1.11.1 Unit Testing

Every good software product in the world has one thing in common, testing. Nothing comes out perfect and any design needs polishing to the upmost capabilities of the developer. Unit testing is one of the main areas of this project's testing phase. The testing is divided into units that was tested individually. These units are but not limited to GPS tracking, payment system, authentication at log ins, rating system, order placing and verifying system and administrator management.

The testing process tests each individually testable component solitarily. The test restarts after every error fix. The test goes through each component's code path when possible. Special tests are designed to test specific scenarios to find bugs.

1.11.2 Integration Testing

After the unit testing is done, we must do an integration testing to see the camaraderie between the different units. The testing is done by using devices with IOS operating system and Android operating system. The administration management system are tasted in different browsers since it is going to be a web application. The question the test seek to answer is if the system can function as a complete and one entity. The installing process on devices are tested. The testing also include the state of functionality of the components when they interact with each other.

For example, after a customer place an order, and pays, the customer must be able to track the journey of the deliverer. The order placing and verifying unit must be able to work together with the payment and GPS tracking units smoothly. Smooth transactions are the backbone of our system. As such, transaction testing is crucial for our system to be efficient and functional. For this, we created test cases and use cases to track every step of the interaction. We run tests using the use cases, detect errors and finally do the tests again after the error has been fixed. The testing process is repetitive and consistent, beginning anew with every new feature added.

1.12 Task and schedule

Time management is critical in software development since it influences the project's outcome. Time management tactics can help you be more efficient and effective as a software developer. This is the schedule we've set for ourselves during the development process.

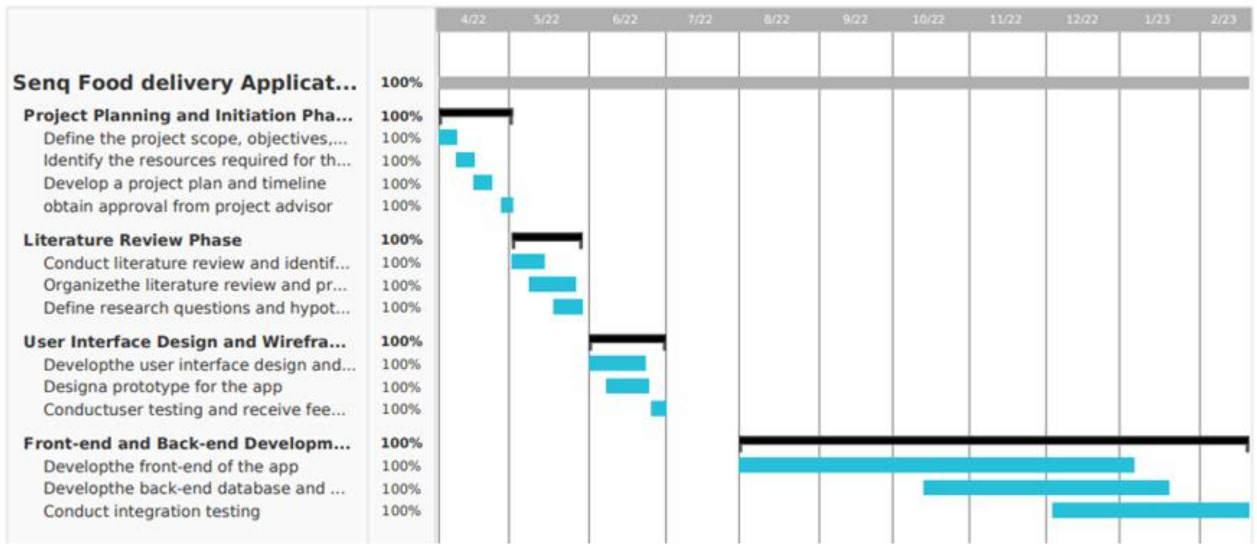


Figure 1 Gantt Chart

1.13 Team Composition

Here are the members of these team

Table 1: Team Composition

Project title	**Food delivery Application**				
Prepared by	No	Name	ID	Contact Address	Responsibility
	1	Abraham Mebrate	Eitm/ur158942/ 11	094127753 3	All
	2	H/Mariam Assefa	Eitm/ur156064/ 11	090825205 2	All
	3	Mered Gugsa	Eitm/ur156207/ 11	093615215 4	All
	4	Nahusenay Tadesse	Eitm/ur158884/ 11	094734060 2	All
	5	Yared Tadesse	Eitm/ur158391/ 11	093267544 0	All
Advisor	Mr Natnael				

Chapter 2

2. Description of the Existing System

2.1 Introduction of Existing System

As our current system there are few systems that we can take like Delivery Addis, Zmall Delivery But we decided to take beU Delivery as our main reference. beU is a successful food delivery service that mainly operates in Addis Ababa. It works with users ordering food through a mobile application from a restaurant found in a four-kilometer radius of their whereabouts. A biked deliverer employed by beU will take the order from the restaurant and deliver it to the user. The restaurant will be notified of the order via phone call from the beU office and the deliverer will pay cash when he/she takes the food from the restaurant.

BeU delivery is one of the pioneers in the online food delivering service in Ethiopia. It has laid the critical groundwork in its field that will be vital for future companies. However, its system is inefficient. Currently the mobile application is only used to notify the office what the user wants. The order tracking and paying system are all handled through phone calls and face to face. Since the number of orders that can be delivered through phone calls is limited by the number of operators tracking orders, growth in the number of orders will be debilitating for the company.

Even if beU has enough restaurants contracted to deliver a large amount of food every day, the system will be an anchor to its growth. The more foods and orders come, the more deliverers and more phone operators beU will have to hire, which will inevitably put a strain in the financial capacity of the company. These can be seen in the reviews left behind by frustrated users whose orders has been delivered late or not delivered at all. beU is increasing the workload on a manual force, when the technology exists for it to be automated. Relying on backward systems when better one exists might eventually lead to beU itself becoming outdated.

2.2 Players in the Existing System

Customer: the person that uses the system to order the food. The customers can be anyone within the range of 4km around the restaurant. Because beU deliver uses bicycles and motors for delivery system and their coverage area is small. And also, the customers should download the app and have an internet access. The customer doesn't need to create an account because the paying system is not native inside the app rather it's in cash, but if the customer wants to have a discount or coupons, they must create account.

Deliverer: in our current system which is beU delivery the deliverer is working under the company as a permanent worker. Not anyone can be a deliverer you should apply for a job in order to be a deliverer. The condition varies from company to company but most of them includes these preconditions. The deliverer must have the skill to use a bike or motorcycle. The deliverer must have an understanding of using the existing system. The deliverer must give personal information copy like driver license for motorcycle and ID or passport.

The service providing company: this are the companies that made the system and administer it along the way. They also provide maintenance and updates too. We used beU delivery for our main reference as existing system. This app was developed by binary technologies which is based in Addis Ababa. Their main job after building the app was compelling the hotel and the customers to get into the system and after that they manage orders and deliveries inconsistencies which may happen in the system.

Restaurant: they sign agreement with the service providing company and will register to the system as one stakeholder. In our existing system which is beU delivery the hotels agree with the binary software and after signing the agreement their food will be available for order in the app.in the existing system the hotels and the service providing company will be connected through some media. The company will call the hotel if there is any orders. Then after the food is ready the hotel will call back to the company and the company will assign the deliverer.

Operator: a person that works in the service providing company that will connect the customer, the hotel and the deliverer. First after receiving the order the operator will call the customer to verify the order after that the operator will call the hotel if there is any change and also manages orders in order to make sure that all orders are placed to the hotels. Operators are permanent workers at the service providing company.

2.3 Major Functions/Activities in the Existing System

The traditional version of ordering food for a takeaway service involves the customer physically going to the restaurant and placing an order. After ordering, the customer requests for a takeaway service. The food is packaged and given to the customer. Currently there are web pages and apps used to order food delivery. The following section summarizes the function of existing computerized system with their input, process and output.

Input:

From customer, the inputs to the system from customer are phone number, Name and location data. The customer phone number is used to contact the customer, the name for security purposes and location data for delivery. The customer enters this inputs in-order to acquire an account.

From restaurant, the inputs to the system from restaurants are phone number, menu list, and location data. The phone number is used to receive orders and communicate any issues that arise during production. A complete menu list containing pricing and dietary restrictions is required from the restaurant. The location data is used for delivery purposes.

2.4 Business Rules

The main business rules or principles for a restaurant are: -

- The restaurant is responsible to set-up phone line in order to receive orders phone.
- The restaurant should communicate any issues that will affect the delivery time using the phone to the admin
- The restaurant must complete orders and have them ready within 20 minutes after the order is confirmed by the admin
- All orders must be labeled and packaged in accordance with rules.
- To properly mark all menu items with the dietary restrictions that apply to them.

The main business rules or principles for Administrator are: -

- To collaborate on a meal plan with the restaurant.
- To provide delivery and customer service for orders to the restaurant
- To notify and confirm orders via phone call to the restaurant's phone number
- To provide delivery and customer services for orders to the restaurant
- To inform any type of changes that will affect the delivery time of the order

The following are shared business rules and principles by the restaurants and the admin: -

- The restaurant and the service provider should agree on an administrative service charge fee from each order to be paid to the service provider.
- The restaurant and the service provider should agree on the cost of correcting errors in accordance to where these errors originate from.

2.5 Bottlenecks of the Existing System

For reference of existing systems, we can take a look at both beU delivery and manual traditional food ordering. BeU delivery's flaw lies in the system being new, relies on traditional methods too much and lacks accurate locating systems. The traditional food ordering system in other hand has an obvious disadvantage concerning time management and efficiency.

2.5.1 Input (Inaccurate/Redundant/Flexible) and Output (Inaccurate)

The existing system takes inputs from the users. Some of these inputs are entered while creating an account such as phone number and emails which can be used to validate the account, this is a one of the good qualities about the existing system. The system also takes location of users by the users entering it rather than GPS which leads to inaccurate inputs. This will lead to order cancellation or inaccurate delivery.

2.5.2 Security and Control

In order to access the existing system, one must already have an account to login or create account; in both cases the system is secure. This way the users account is secured and can be only accessed by the user only.

2.5.3Efficiency

According to the comments of some users of the existing system on Google play store, most users agree the system is "inefficient. The orders get canceled out of nowhere and the users don't get any notification whatsoever. The system takes a lot of time to load and to display available foods. The users cannot track their food, and they do not have any idea where their food is, its status or whether or not it is being made.

2.6 Forms and Other Documents of the Existing Systems

2.6.1 Agreement form

This is a form which Binary Technologies PLC, BUE uses to make an agreement with restaurant.

Agreement Overview

1.1 Binary Technologies PLC hereby retains the restaurant to provide and perform catering services including any derivative, replacement, and follow-on services thereto (the "Services").

1.2 This agreement represents a service level agreement between Binary TechnologiesPLC and the restaurant for the provision of the services.

Pricing

3.1 The amount paid by Binary Technologies PLC to the restaurant for any order fromBinary Technologies PLC and the amount charged by Binary Technologies PLC .

(a) The restaurant agrees to pay Binary Technologies PLC an administrative Fee on everyorder, equal to______of the order price,

(b) For each order, a delivery fee is charged from the client. This pricing is

Service Agreement

1.3 The following detailed service parameters are the responsibility of the Restaurant inthe ongoing support of this agreement:

(a) To mutually agree with Binary Technologies PLC on a menu of foods or drinks (collectively "Meals") that includes meals descriptions, meals cost to Binary Technologies PLC, and meals pricing at the restaurant's physical location, all of whichmay be changed from time to time if mutually agreed upon by the parties.

(b) To work with Binary Technologies PLC to set up the ability to receive ordersvia phone.

(c) To notify Binary Technologies PLC immediately by phone call, if issuesarise that will affect or delay the order fulfillment.

(d) To fulfill and have complete orders ready within twenty (20) minutes afterorders are confirmed with Binary Technologies PLC call center.

(e) To label and package all orders according to the Binary Technologies PLCbrand guidelines.

(f) To clearly label all menu items with their corresponding dietary restrictions(gluten-free, vegetarian, vegan or dairy-free, etc.)

1.4 The following detailed service parameters are the responsibility of BinaryTechnologies PLC in the ongoing support of this Agreement:

(a) To work with restaurants on a meal menu.

(b) To provide delivery and customer service for orders to the restaurant from BinaryTechnologies PLC.

Merchant Food Health Standards

1.5 The merchant agrees that all orders must satisfy the local governing health regulationfor food preparation. The merchant agrees that violating these standards will be grounds for immediate termination of this agreement by Binary Technologies PLC, as well as for action byBinary Technologies PLC against the merchant for damages.

Miscellaneous

a. Nothing in this agreement will constitute a commitment by Binary Technologies PLC to send to the merchant any particular volume or Quantity of orders.

b. Nothing in this agreement shall preclude Binary Technologies PLC from obtainingservices similar to the Services from third parties.

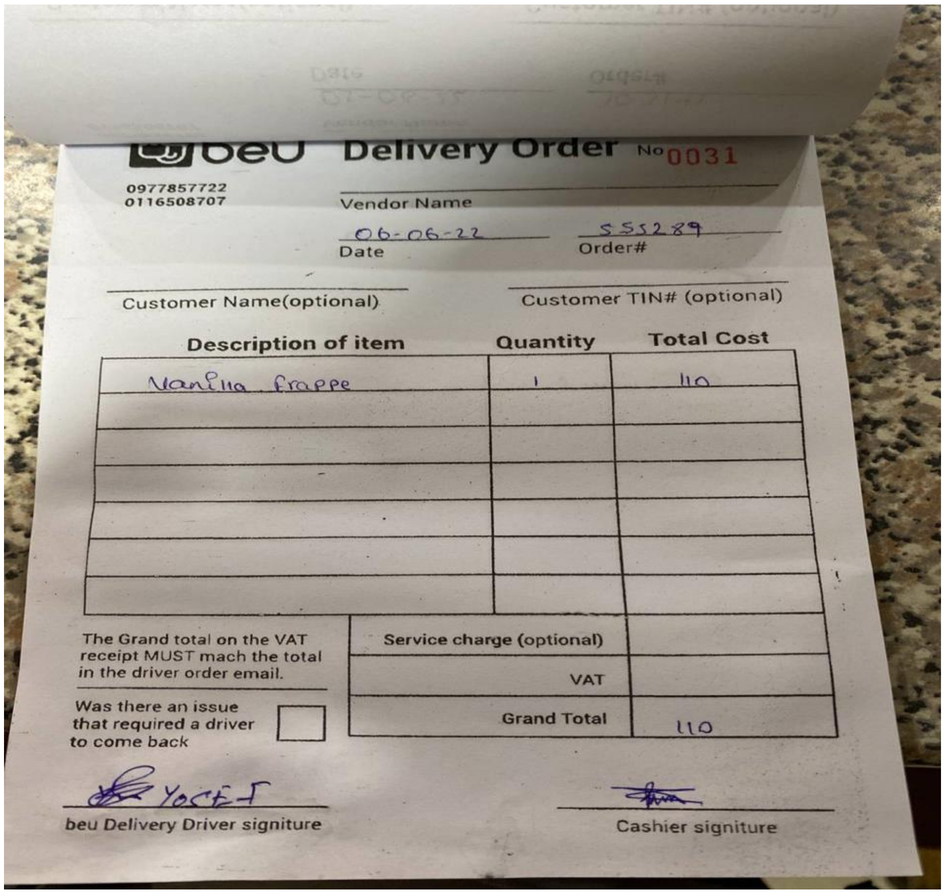
By: ___________________________________ Name:_______________

The Restaurant (Legal Name): _____________________________________

Name:__________________________________ Title:__________ Date: _______

2.6.2 Order pad form

This is a form in which deliverers of BUE or bicycle riders write on. It is permanently put in the restaurants and contains each and every food taken from the restaurant is recorded with its order id, primary key of the existing system given to every delivery order.



beU Delivery Order No 0031

0977857722
0116508707

Vendor Name

06-06-22
Date

555289
Order#

Customer Name(optional)

Customer TIN# (optional)

Description of item	Quantity	Total Cost
Vanilla frappe	1	110

The Grand total on the VAT receipt MUST mach the total in the driver order email.

Was there an issue that required a driver to come back ☐

Service charge (optional)	
VAT	
Grand Total	110

[illegible] Yosef
beu Delivery Driver signiture

[illegible]
Cashier signiture

Figure 2: beU customer receipt

2.7 Practices to be preserved

The existing system have some good qualities that worth preserving. The user interface is one of them, beU delivery mobile app have a great looking user interface that can attract a user and will make them visit the app again. We have preserved the following practices from in the existing system.

- The Authentication system.
- The flexibility and simplicity of the user interface.
- The identification of orders by unique order number.
- Using of GPS.

Chapter 3

3. Proposed System

3.1 Overview

This project uses an object-oriented development as a guideline. Object oriented programming aids the project by reducing the development time, reduces the resources required to maintain existing applications and increase code reuse.

In this chapter the relationship between the different entities and their function is laid out using UML diagram. These UML diagrams are divided into four parts, Class, Sequence, Activity and Use Case diagrams.

User Interface prototype diagrams are also included in this chapter. The prototypes are not real representation of the Cloud Based Food Ordering and Delivery system.

3.2 Functional requirements

1. The system lets the customer register or create a new account.
2. The system lets the customer login if they already have an account
3. The system gets the customer's location information.
4. The system lets all the users manage an account.
5. The system lets the customer navigate through the app to place an order.
6. The system lets the customer search any dish or restaurant.
7. The system lets the customer select a dish from the menu.
8. The system lets the customer add a dish from their order basket.
9. The system lets the customer remove a dish from their order basket.
10. The system lets the user place an order.
11. The system lets the user view an order.
12. The system lets the user pay in a different way.
13. The system lets the user track an order.

14. The system lets the restaurant create an account with a different registration requirement, like the physical address verification procedure, legal business information.
15. The system lets the restaurant add a new dish into the menu.
16. The system lets the restaurant delete a dish from the menu.
17. The system lets the restaurant manage the orders.
18. The system lets the restaurant view the orders.
19. The system lets the restaurant change the order status.
20. The system lets the courier register or create a new account.
21. The system lets the courier login if they already have an account
22. The system gets the courier location in a real-time manner.
23. The system notifies the courier if there is an order to a restaurant near them.
24. The system would give the courier all the needed order information including the customer address, delivery time and distance, order description, its specification.
25. The system would give the courier a choice to accept or decline the order.
26. The system lets the courier update the order status. Courier have to update the status at least twice when the order is picked and completed.
27. The system lets the courier view orders.
28. The system lets the admin login to the system
29. The system lets the admin Change the order status if any issue occurred.
30. The system lets the admin view orders.
31. The system lets the admin manage the restaurant account like validation of the documents and agreement.
32. The system lets the admin manage the courier accounts like validation of their documents, driving license and passport/ID

3.3 Non-Functional Requirement

1. There is a simple registration process that can be easily and efficiently done.
2. Users easily find the food they want easily. Without no multi step checkout like users must reach the “Add to basket” button in one step.

3. Only the system data administrator can assign roles and change access permissions to the system. The system is resilient to any kind of attacks.
4. The system loads in less than 4 seconds on Android and IOS 10+ on 3G/4G.
5. Because we are aiming to grow, the system removes all the back-end complexities for in-house engineers to make changes to the system in the future.

To properly judge the performance of Cloud-based Food Ordering and Delivery a criterion must be met. These criteria are listed above as Non-functional requirements.

3.4 Performance Requirements

The purpose of this section of the document is to outline the Software Performance requirements of our system. The following scales are used to measure the performance of the system: -

1. Response Time

Software:

The system developers warrant that in supporting 2,000,000 customers it shall ensure that performance shall not fall below the following level: 90% of ALL visible pages for "normal" customers respond in 3 seconds or less.

Measurement Points:

The response times are measured using AWS Analytics (or similar tool) in front of the web servers. The timer measures the time from the request for a page to when the last bit required to render the page is returned. Backend response times are measured using the application server log files.

2. Workload:

The software supports 30,000 customers which will on a busy day generates 7000 customers

3. Scalability:

The system developers warrant that the food delivery system is capable of supporting at least unlimited customers when implemented into a suitable production environment.

3.5 Identification of actors and use cases

Actor Identification:

An Actor in the Unified Modeling Language (UML) "specifies a role played by a user or any other system that interacts with the subject. The followings are a list of Actors in the Proposed System.

Customer: The person that uses our system to order a dish from a desired restaurant. The customer may be a new user or an existing user.

Restaurant: Is an organization in our system that provide the dishes to the customer. The restaurant needs to provide additional documentation and legal agreement document when registering.

Courier: The person that users our system to deliver the order to the customers. The Courier need to provide additional documentation and legal agreement document when registering.

Administrator: The person who manages the entire system. There may be an issue occur in the system in this case the administrator can take charge and fix the issue. The Registration process of the courier and the restaurant would be verified by the administrator.

3.6 Use Case Model

3.6.1 Use case diagram

A use case diagram is a way to summarize details of a system and the users within that system. It is generally shown as a graphic depiction. n of interactions among different elements in a system. [9]

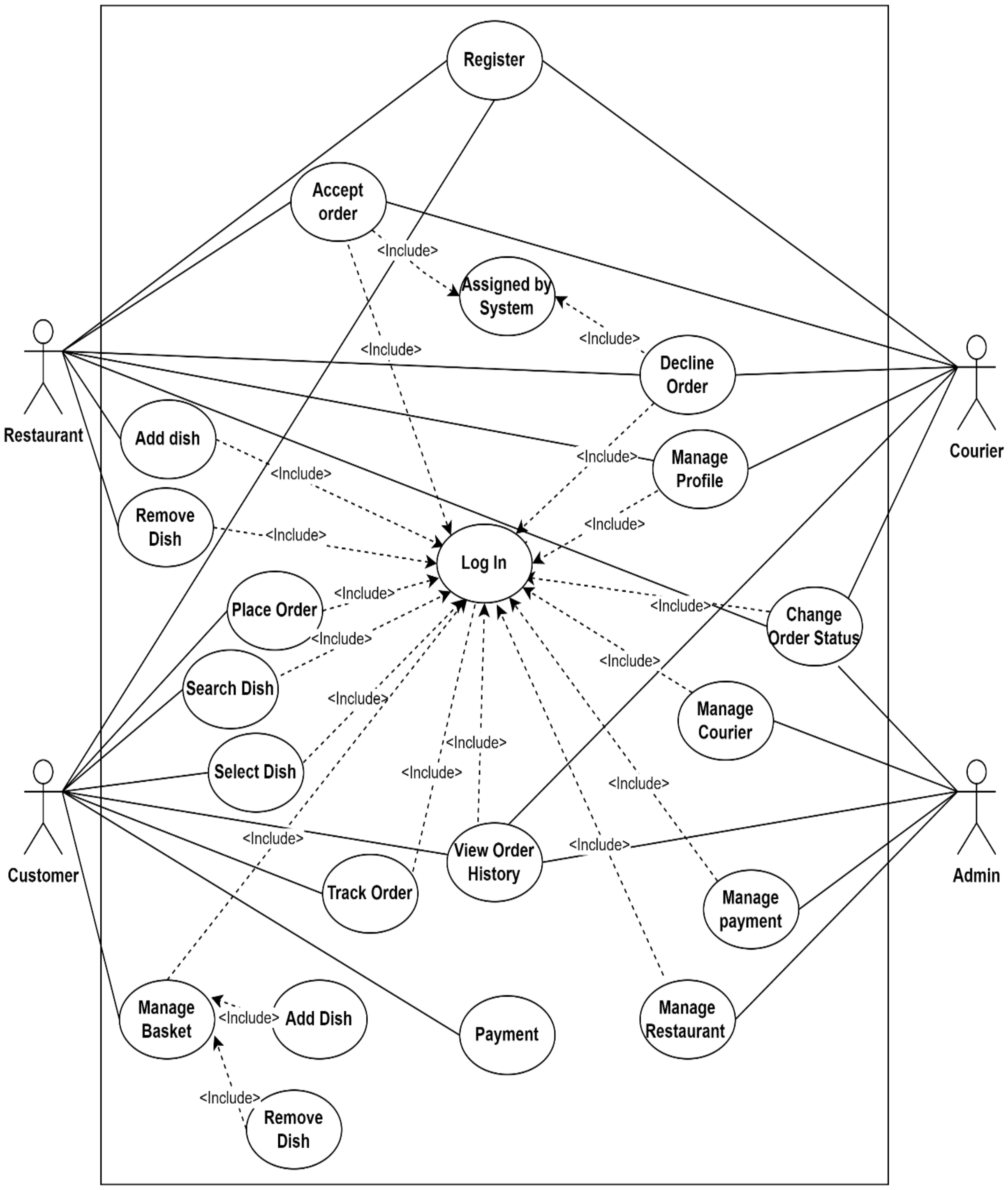


Figure 3 Use Case Diagram

3.6.2 Description of use case model

Table 2: Register Use Case Description

Use Case – UC-1	Register	
Related Requirement	Req-1, Req-14, Req-21	
Initiating Actor	Customer, Courier, Restaurant	
Actor's Goal	To create account	
Participating Actors	Customer, Courier, Restaurant, Admin	
Pre-conditions:	The Customer, Courier, Restaurant and Admin must have access to the system	
Post conditions	The customer, Courier and Restaurant have an account	
Basic course of action for Actor	**Actor Action** **1**. The Actor selects sign up option in the customer app **3.** The Actor fills all the information **5.** The Actor inputs the verification code	**System Response** **2**. The system displays registration form **4**. The system sends verification code **6.** The system creates a new account for the Actor and forwards the Actor to the main page
Alternative course of action for the Actor	**7.** If the Actor enters wrong information, **9.** If the Actor enters duplicate account	**8.** The system displays error message **10.** The system display "information already exists" message.

Table 3: Log in Use Case Description

Use Case – UC-2	Log in
Related Requirement	Req-2, Req-21, Req-28
Participating Actor	Customer, Courier, Restaurant, Admin
Actor's Goal	To gain access to their account

Pre-conditions:	The Customer, Courier, Restaurant and Admin must have an already existing account	
Post conditions	The Customer, Courier, Restaurant and Admin logins to the system	
Basic course of action for Actor	**<u>Actor Action</u>** **1**. The Actor selects log-in option in the Actor app **3.** The Actor fills all the information (user-name and password) **5.** The Actor is forwarded to the main page	**<u>System Response</u>** **2**. The system displays Actor log-in page **4**. The system verifies the user-name and password
Alternative course of action for Actor	**6.** If the customer enters wrong information (user-name and password)	**7.** The system display error message

Table 4: Manage Profile Use Case Description

Use Case – UC-3	Manage profile	
Related Requirement	Req-4	
Participating Actor	Customer, Courier, Restaurant	
Actor's Goal	To change user-name, email address or/and phone number information	
Pre-conditions:	The Customer, Courier, Restaurant must log in to the system or must already be logged in	
Post conditions	Changed profile information of customer, courier or restaurant	
Basic course of action for Actor	**<u>Actor Action</u>** **1**. The Actor selects edit profile option in the customer app **3.** The Actor edits their information	**<u>System Response</u>** **2**. The system displays Actor profile page **4**. The system confirms the edited information and displays a confirmation message
Alternative course of action for Actor	**5.** If the Actor enters invalid information (user-name and password)	**6.** The system displays error message

Table 5: Accept Order Use Case Description

Use Case – UC-4	Accept order	
Related Requirement	Req-17, Req-25	
Initiating Actor	Restaurant, Courier	
Actor's (Restaurant)Goal	To accept order from the customer	
Actor's (Courier)Goal	To receive the food from the restaurant	
Pre-conditions:	The Restaurant and the Courier must log in to the system or must already be logged in	
Post conditions	The restaurant changes the state of the process to "cooking" The courier changes the state of the process to "picked up"	
Basic course of action for the Restaurant	**<u>Actor Action</u>** **2**. The restaurant receives the order details **3.** The restaurant accepts the order **4**. The restaurant changes the state of the process to "cooking"	**<u>System Response</u>** **1**. The customer sends the order details to the restaurant
Basic course of action for the Courier	**<u>Actor Action</u>** **2**. The courier receives the order details **3.** The courier accepts the order **4**. The courier changes the state of the process to "picked up"	**<u>System Response</u>** **1**. The restaurant changes the order state to "ready for pickup"

Table 6: Decline Order Use Case Description

Use Case – UC-5	Decline order
Related Requirement	Req-17, Req-25
Initiating Actor	Restaurant, Courier
Actor's (Restaurant)Goal	To decline order from the customer
Actor's (Courier)Goal	To decline order from the restaurant
Pre-conditions:	The Restaurant and the Courier must log in to the system or must already be logged in
Post conditions:	The restaurant waits for another order The courier waits for another order

Basic course of action for the Restaurant	**Actor Action** **2**. The restaurant receives the order details **3.** The restaurant declines the order **4**. The restaurant changes the state of the process to "new"	**System Response** **1**. The customer sends the order details to the restaurant
Basic course of action for the Courier	**Actor Action** **2**. The courier receives the order details **3.** The courier rejects the order	**System Response** **1**. The restaurant changes the order state to "ready for pickup"

Table 7: Change Order Status Use Case Description

Use Case - UC6	**"Change order Status"**	
Related Requirement	Req-19, Req-26,Req-28	
Initiating Actor	Restaurant, Courier, Admin	
Actor's Goal	To Change The status of the order after some event occur.	
Participating Actors	Restaurant, Courier, Admin	
Pre-conditions:	The restaurant must receive the order or restaurant started the cooking process or the restaurant finished cooking the dish. The Distance between the courier and the restaurant must be less than 100m due to limitation in accuracy. The Distance between the courier and the customer must be less than 100m due to limitation in accuracy. The admin must have been notified of some issue.	
Post conditions	The status of the order changes to new status for the next events.	
Basic course of action for the Restaurant	**Actor Action** **1**. The restaurant manager clicks the orders menu. **3** The restaurant manager click on the current order status **5.** The Restaurant manager changes it to a new status.	**System Response** **2**. The system shows the orders. **4**. The system shows the current order status and the drop down menu to change the order status. **6.** The system updates to the new status

Alternative course of action for the Restaurant	**7.** If the restaurant manager enters wrong status to the order to fraud the customers or courier the admin will take action.	
Basic course of action for the courier	**<u>Actor Action</u>** **1**. The courier clicks the change the status button after going to restaurant. **3.** After the courier picked up the order the courier changes the status of the order from "Ready for pick up" to "picked up". **5.** Once again the change status button becomes clickable if the courier near Customer address. the courier can change the status of the order from" Picked up" to "Completed"	**<u>System Response</u>** **2**: The system shows the drop down menu to choose the status from **4**. The system updates to the new status and turn off the change status button again. **6**. The system updates to the new status
Alternative course of action for the courier	**7**. If the restaurant didn't serve the food on time the courier should wait.	
Basic course of action for the Admin	**<u>Actor Action</u>** **1**. The admin receives an issue from, the Customer, courier or restaurant. **2.** The admin clicks manage order button **4.** The admin clicks the order that need to be resolved **6.** The admin changes the status of the order from the drop-down menus	**<u>System Response</u>** **3**: The System shows the orders **4**. The system shows the details of that order **5.** The system shows the details of that order **7.** The system updates the status and displays the new status

Table 8: View Order History Use Case Description

Use Case – UC-7	View order history
Related Requirement	Req-19, Req-26, Req-27, Req-30
Initiating Actor	Restaurant, courier, admin, customer

Actor's Goal	To view order	
Participating Actors	Restaurant, courier, admin, customer	
Pre-conditions:	There should be an order placed	
Post conditions	Cancelation, reordering of viewed	
Basic course of action for the actors	**Actor Action** **Step 1**: The customer places an order **Step 3:** The actors inputs order ID **Step 4**: The actors click view order	**System Response** **Step 2**: The system gives ID to the order **Step 5:** The system displays the order
Alternative course of action for the actors	**Step 6**: If the actor enters wrong order ID	**Step 7:** The system display error message

Table 9: Search Dish Use Case Description

Use Case – UC-9	Search dish	
Related Requirement	Req-6	
Initiating Actor	Customer	
Actor's (Customer)Goal	To view order	
Participating Actors	Customer	
Pre-conditions:	No pre-conditions	
Post conditions	View the dish searched	
Basic course of action for the actors	**Actor Action** **Step 1**: The customer clicks search button **Step 3:** The customer search for the desired dish	**System Response** **Step 2**: The system enables the customers to search a dish in different orders **Step 4**: The system displays the searched dish
Alternative course of action for the actors	**Step 5**: If the customer search for unavailable dish	**Step 6:** The system display "No such dish "

Table 10 Select Dish Use Case Description

Use Case – UC-10	Select dish	
Related Requirement	Req-7	
Initiating Actor	Customer	
Actor's Goal	To select dish	
Participating Actors	Customer	
Pre-conditions:	Select restaurant	
Post conditions	Displays the description of the selected dish	
Basic course of action for the actors	**Actor Action** **Step 1**: The customer selects restaurant **Step 3:** The customer selects a dish from the display **Step 5:** The customer confirms the selected dish	**System Response** **Step 2**: The system shows all dishes available in the restaurant **Step 4**: The system asks for confirmation
Alternative course of action for the actors	**Step 5**: If the customer didn't select a restaurant	**Step 6:** The system display "No such dish "

Table 11: Manage Basket Use Case Description

Use Case – UC-11	Manage Basket	
Related Requirement	Req-4, Req-5, Req-6, Req-7, Req-8, Req-9	
Initiating Actor	Customer	
Actor's Goal	To manage multiple orders in a single order	
Participating Actors	Customer, Courier, Restaurant, Admin	
Pre-conditions:	The Customer must log in to the system or must already be logged in The Customer must select a single restaurant The Customer must select at least one dish	
Post conditions	The Customer had one or multiple dishes placed on the same order	
Basic course of action for the Actor	**Actor Action** **Step 1**: The Actor selects a restaurant	**System Response**

	Step 3: The Actor adds dishes from their order	**Step 2**: The system displays dishes the selected restaurant offers **Step 4**: The system put the selected dishes in the same order and make the place order button clickable

Table 12: Add Dish Use Case Description

Use Case – UC-12	Add Dish	
Related Requirement	Req-4, Req-5, Req-6, Req-7, Req-8	
Initiating Actor	Customer	
Actor's Goal	To add a dish in the order basket	
Participating Actors	Customer	
Pre-conditions:	The Customer must log in to the system or must already be logged in The Customer must select a single restaurant	
Post conditions	The Customer adds a dish to their basket	
Basic course of action for the customer	**Actor Action** **Step 1**: The Actor selects a dish	**System Response** **Step 2**: The system adds the dish in the basket

Table 13: Remove Dish Use Case Description

Use Case – UC-13	Remove Dish
Related Requirement	Req-4, Req-5, Req-6, Req-7, Req-8, Req-9
Initiating Actor	Customer
Actor's Goal	To remove a dish in the order basket
Participating Actors	Customer

Pre-conditions:	The Customer must log in to the system or must already be logged in The Customer must select a single restaurant The Customer must select at least a single dish	
Post conditions	The Customer removes a dish from their basket	
Basic course of action for the customer	<u>**Actor Action**</u> **Step 2**: The customer selects a dish to remove	<u>**System Response**</u> **Step 1**: The system make the remove dish button clickable after a dish has been added **Step 3**: The system removes the dish from the basket

Table 14: Place Order Use Case Description

Use Case – UC-14	Place Order	
Related Requirement	Req-10, Req-11	
Initiating Actor	Customer	
Actor's Goal	To place an order	
Actor's Goal	To approve or decline order	
Participating Actors	Customer, Restaurant	
Pre-conditions:	The Customer must log in to the system or must already be logged in The Customer must select a single restaurant The Customer must select at least one dish	
Post conditions	The Customer can place an order	
Basic course of action for the customer	<u>**Actor Action**</u> **Step 1:** The customer adds dishes **Step 4:** The customer selects the place order button **Step 6:** The restaurant either approves or rejects the order	<u>**System Response**</u> **Step 2**: The system put the added dishes in the same order **Step 3**: The system makes the place order button clickable **Step 5:** The system sends the order to the restaurant **Step 6:** The system sends the restaurant's reply to the customer

		Step 7: The system displays an approved or a rejected message
Alternative course of action for the customer	**Step 8:** If the customer presses the back button before pressing the place order button	**Step 9:** the system displays a prompt to let the customer know the order will be lost if they exit without selecting the place order button.

Table 15: Make Payment Use Case Description

Use Case – UC-16	**Make payment**	
Related Requirement	Req-12	
Initiating Actor	Customer	
Actor's Goal	To pay appropriate price for delivered food	
Participating Actors	Customer	
Pre-conditions:	The courier should have to deliver the food The state of the process should be "delivered"	
Post conditions	The customer can rate food	
Basic course of action for the customer	**Actor Action** **Step 2:** The customer selects payment options **Step 3:** The customer pays the price of the food	**System Response** **Step 1:** The system displays available payment options **Step 4:** The system displays payment verification text
Alternative course of action for the customer	**Step 5:** If the customer gives insufficient price	**Step 6:** The system displays error message

Table 16: Track Order Use Case Description

Use Case – UC-17	**Track Order**
Related Requirement	Req-13, Req-22
Initiating Actor	Customer

Actor's (Customer)Goal	To track the order in real time manner	
Participating Actors	Customer	
Pre-conditions:	The customer must place the order The state of the delivery process should be "picked up"	
Post conditions	The customer receives the food	
Basic course of action for the customer	<u>**Actor Action**</u> **Step 1:** The customer request the system to see the where the Courier is	<u>**System Response**</u> **Step 2:** The system displays the location of courier
Alternative course of action for the customer	**Step 3:** If there is mismatch to the exact location then the customer	**Step 4:** The system displays error message.

Table 17: Add Dish Use Case Description

Use Case – UC-18	**Add Dish**	
Related Requirement	Req-6, Req-7, Req-8, Req-15	
Initiating Actor	Restaurant	
Actor's Goal	To add available dishes on the restaurant dashboard	
Participating Actors	Restaurant	
Pre-conditions:	The restaurant should be logged in into the system	
Post conditions	The system displays added dishes	
Basic course of action for the customer	<u>**Actor Action**</u> **Step 1:** Click add dish **Step 3:** Customer enter details	<u>**System Response**</u> **Step 2:** The system requests dish details to add **Step 4:** The system adds dish
Alternative course of action for the customer	**Step 5:** If the dish is not unique	**Step 6:** The system displays error message.

Table 18: Remove Dish Use Case Description

Use Case – UC-19	**Remove dish**	
Related Requirement	Req-9, Req-16	
Initiating Actor	Restaurant	
Actor's Goal	To remove the dish that was added formerly on the system	
Participating Actors	Restaurant	
Pre-conditions:	The dish must be added before	
Post conditions	The dish is no longer be available on the system	
Basic course of action for the customer	**Actor Action** **Step 1:** Click remove dish **Step 3:** Customer enter details	**System Response** **Step 2:** The system requests dish details to be removed **Step 4:** The system removes dish
Alternative course of action for the customer	**Step 5:** If the dish is not there	**Step 6:** The system sends "failed to remove" message

Table 19: Manage Restaurant Use Case Description

Use Case – UC-20	**Manage Restaurant**
Related Requirement	Req-31
Initiating Actor	Admin
Actor's Goal	To do the task of management over the restaurant
Participating Actors	Admin
Pre-conditions:	The Admin must login to the system first
Post conditions	Changes on management related tasks

Basic course of action for the customer	**Actor Action** **Step 1:** The admin selects to sign in to the system **Step 3:** The admin fills up the sign in form **Step 6:** Once the admin logged in to the system The admin possesses the permission to manage the restaurant **Step 7:** When The admin selects the sign-out option	**System Response** **Step 2:** The system displays sign in form to the admin **Step 4:** The system allows the admin to log in to the system **Step 8:** The system forewords the admin to o the sign in page
Alternative course of action for the customer	**Step 6:** The admin can reenter the form	**Step 5:** If the admin fills wrong password and username the system refuses the admin to log in to the system and displays the sign in form again

Table 20: Manage Courier Use Case Description

Use Case Name	"Manage Courier"	
Use Case number	**21**	
Related Requirement	Req-32	
Initiating Actor	Admin	
Actor's Goal	To accept or decline the courier account for registration. To manage fraud and mismanagement. To delete a courier account if a discipline issue found.	
Participating Actors	Admin, courier	
Pre-conditions:	The courier to be managed must be registering or already registered to the system.	
Basic course of action for the Admin	**1.** The Admin clicks on manage courier button **3.** The Admin selects the courier to be managed **5.** The Admin manages the courier. And save the changes **7,** If the courier is applying for registration the admin verifies the given documentation and information. If the	**2.** The system shows the list of already registered couriers and new registration request. **4.** The system shows the details of the courier. **6** The system updates the managed courier account and save the changes

	documentation is not legit the admin will decline the courier **9,** If the documentation is legit the admin will accept the courier	**8,** The system sends verification declination mail to the courier. **10,** The system sends verification accepted mail to the courier.

3.7 Class Diagram

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. [10]

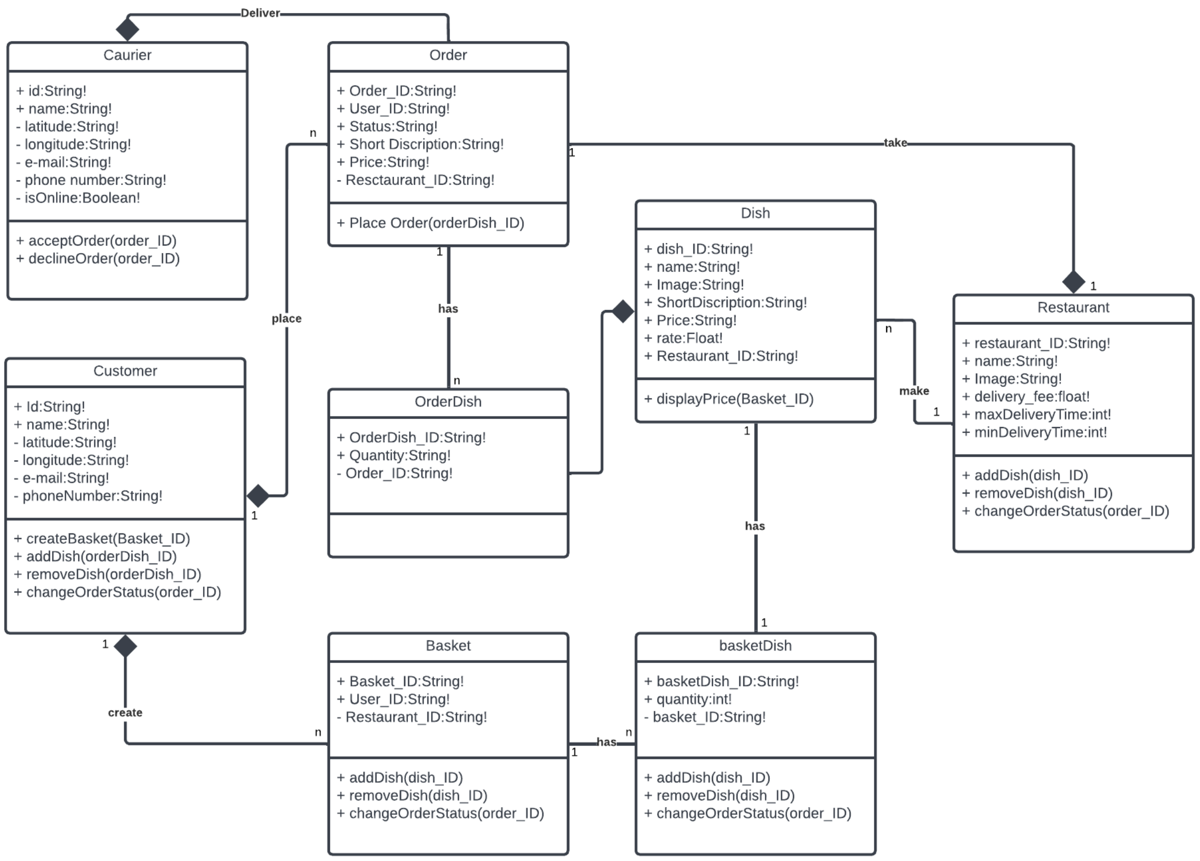


Figure 4 Class diagram of the food delivery system

3.8 Sequence Diagram

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. [11]

In sequence diagram, "Actors are external objects they don't need activation bars". [12]

A. Sequence Diagram for Log in

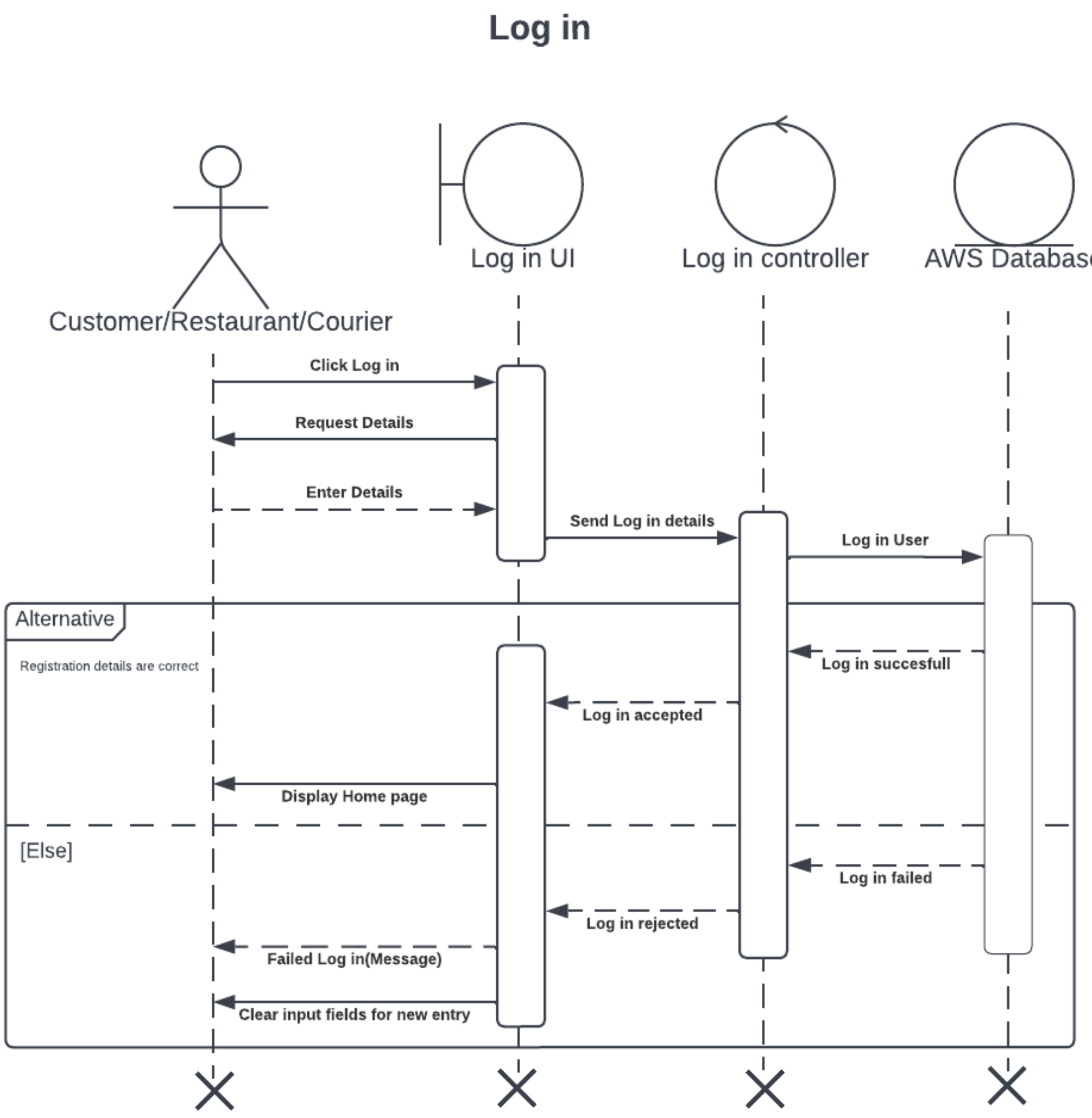


Figure 5: Log in Sequence Diagram

B. Sequence Diagram for Manage Restaurant

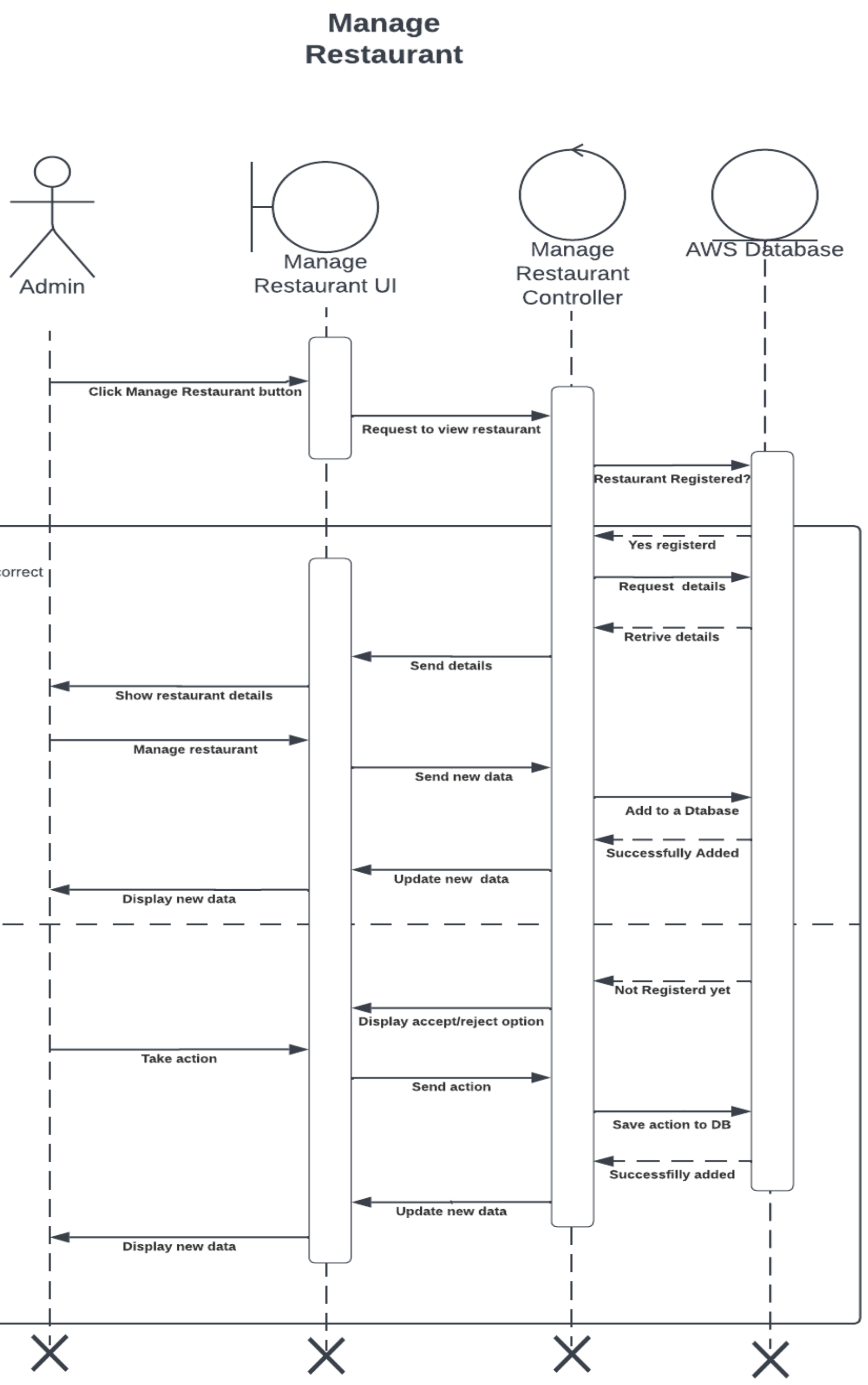


Figure 6: Manage Restaurant Sequence Diagram

C. Sequence Diagram for Manage Basket

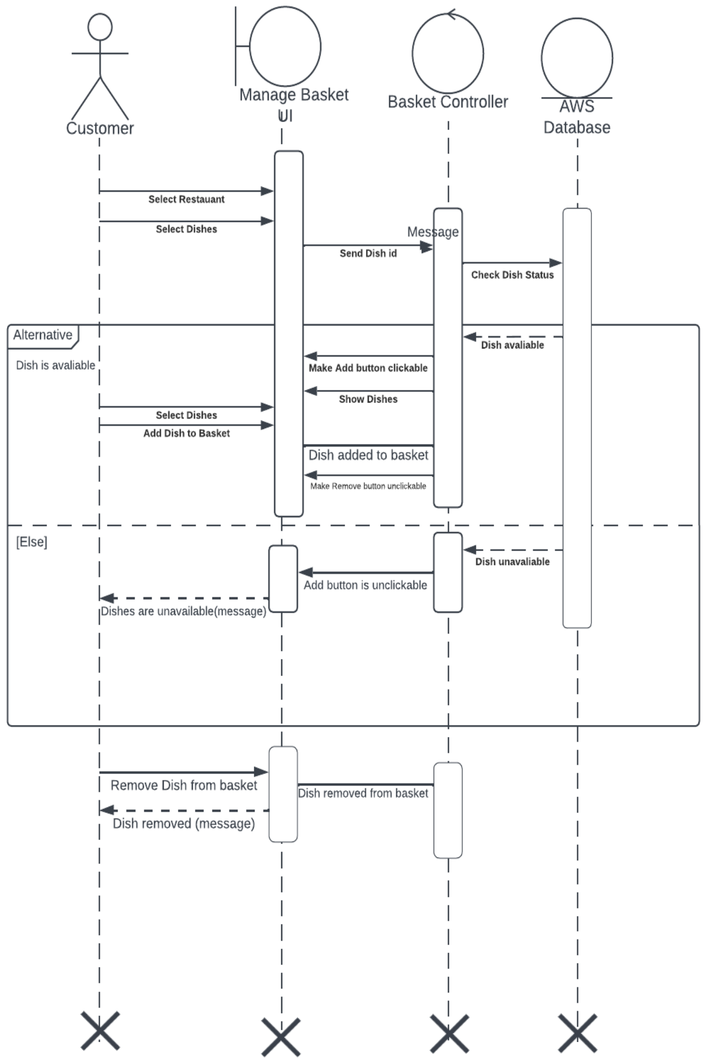


Figure 7: Manage Basket Sequence Diagram

D. Sequence Diagram for Manage Dish

I. Sequence Diagram for Add dish

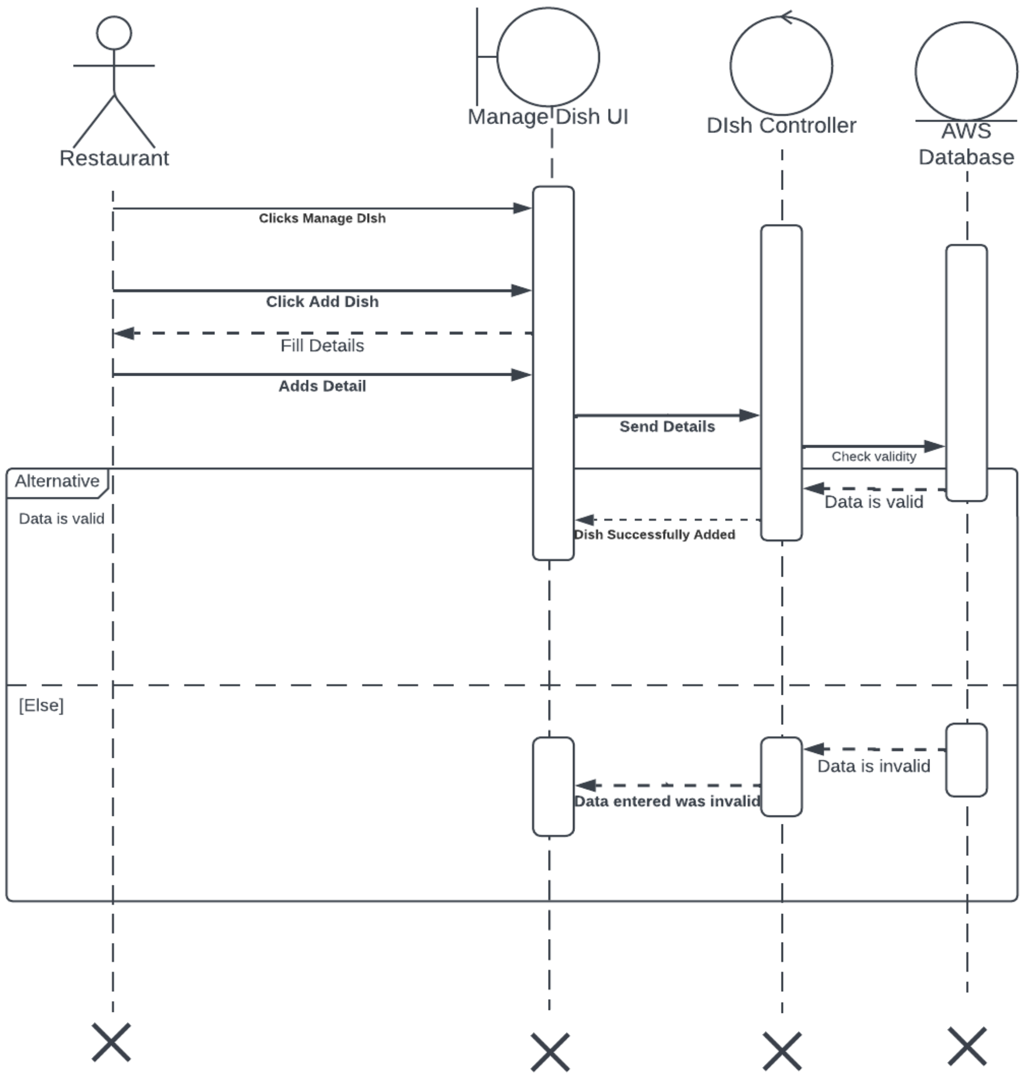


Figure 8: Add Dish Sequence Diagram

II. Sequence Diagram for Remove Dish

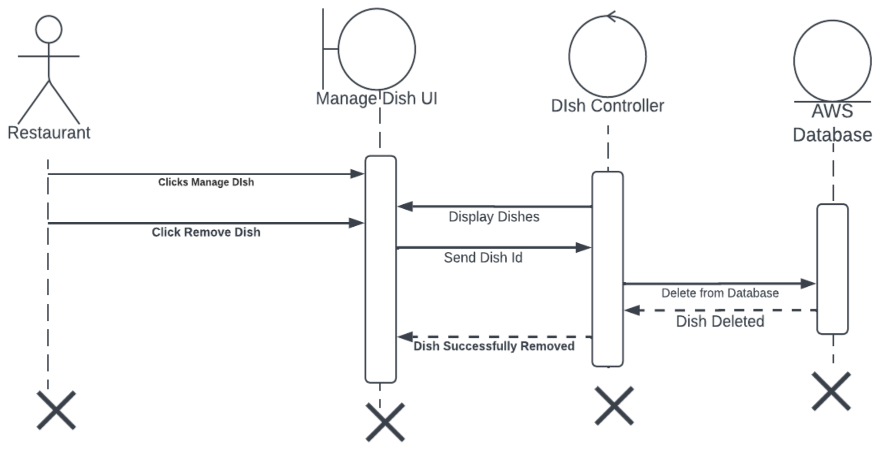


Figure 9: Remove Dish Sequence Diagram

E. Sequence Diagram for Place Order

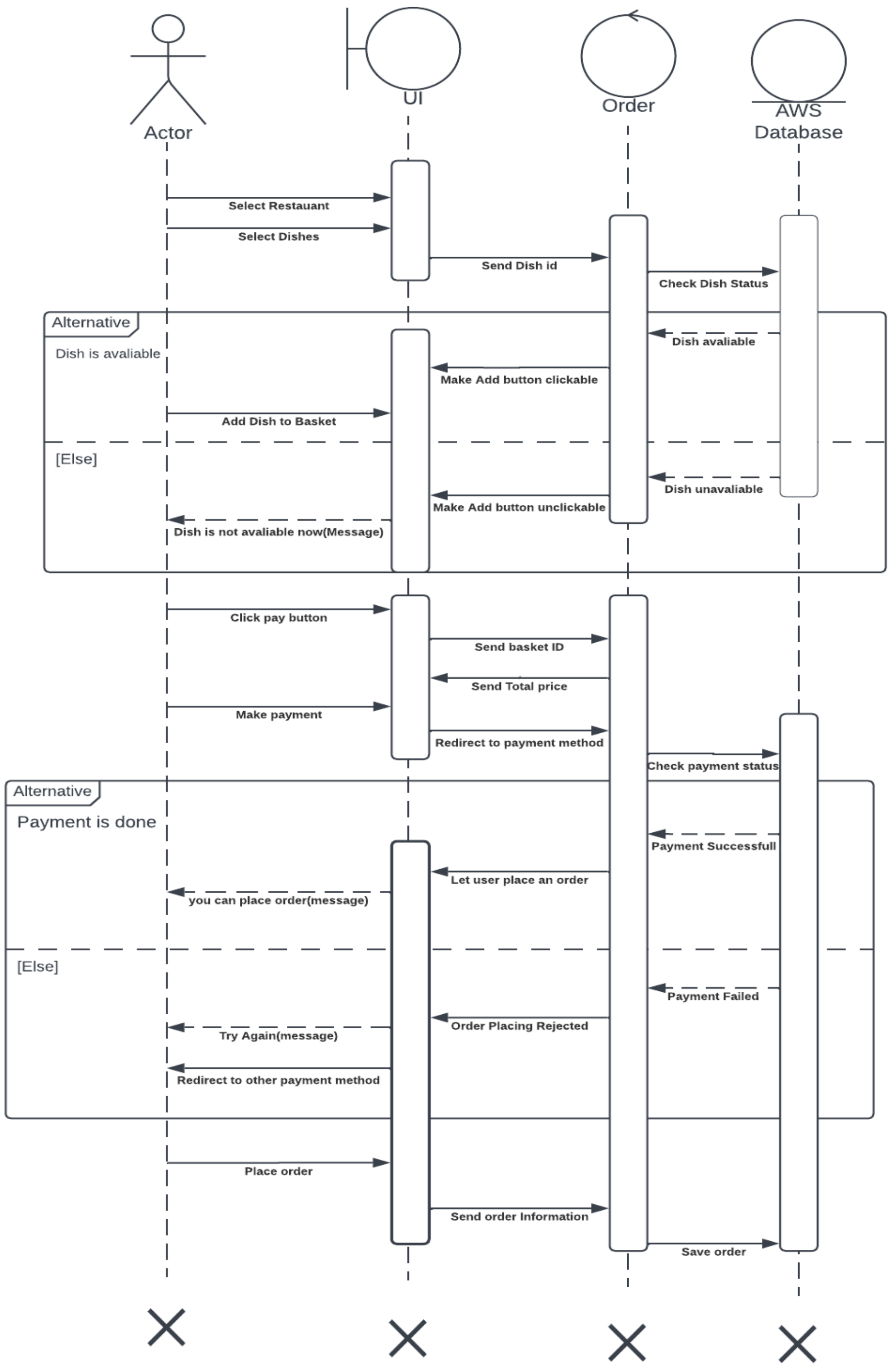


Figure 10: Place Order Sequence Diagram

F. Sequence Diagram for Change Order Status

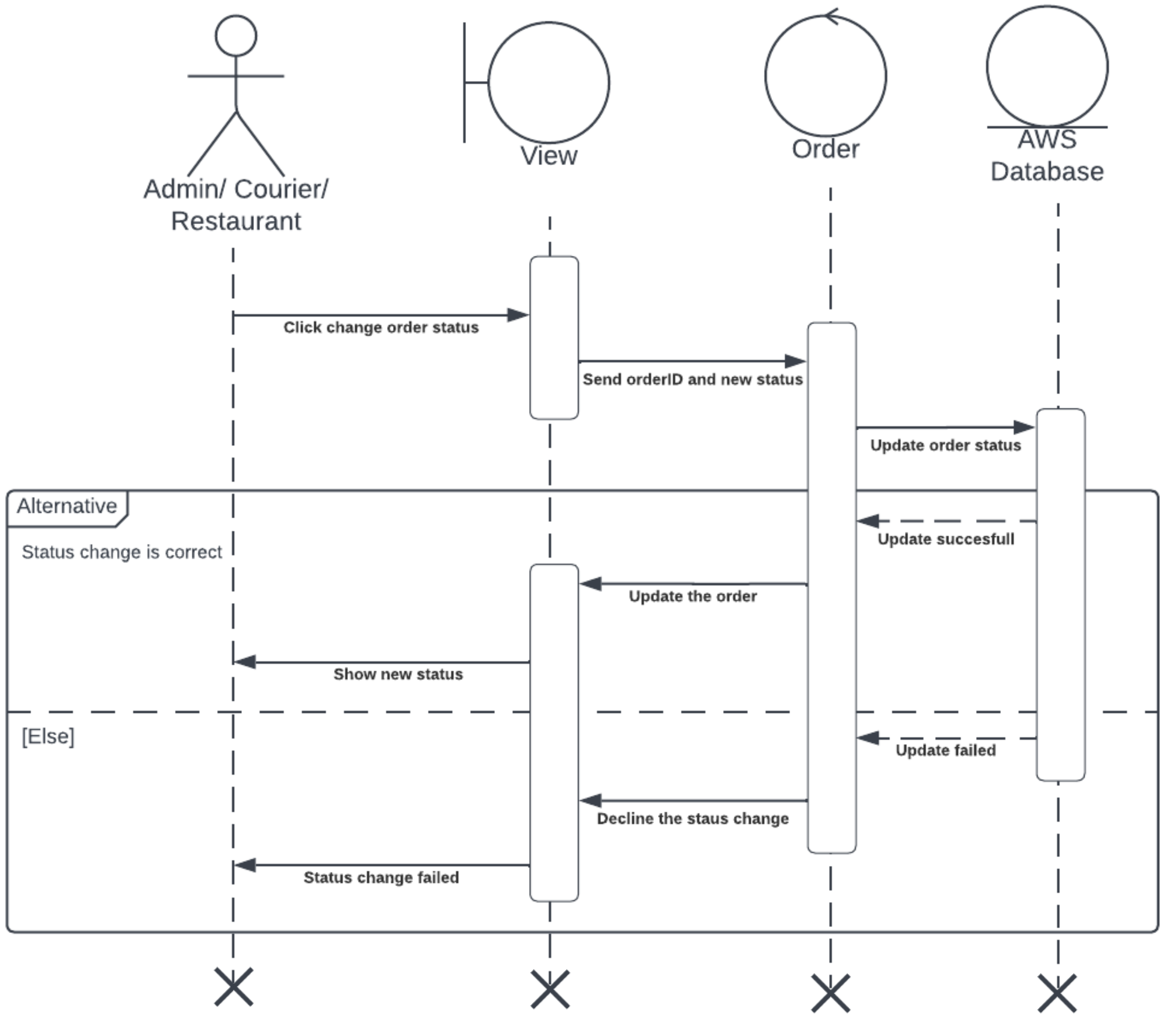


Figure 11 Change Order Status Sequence Diagram

G. Sequence Diagram for Track Order

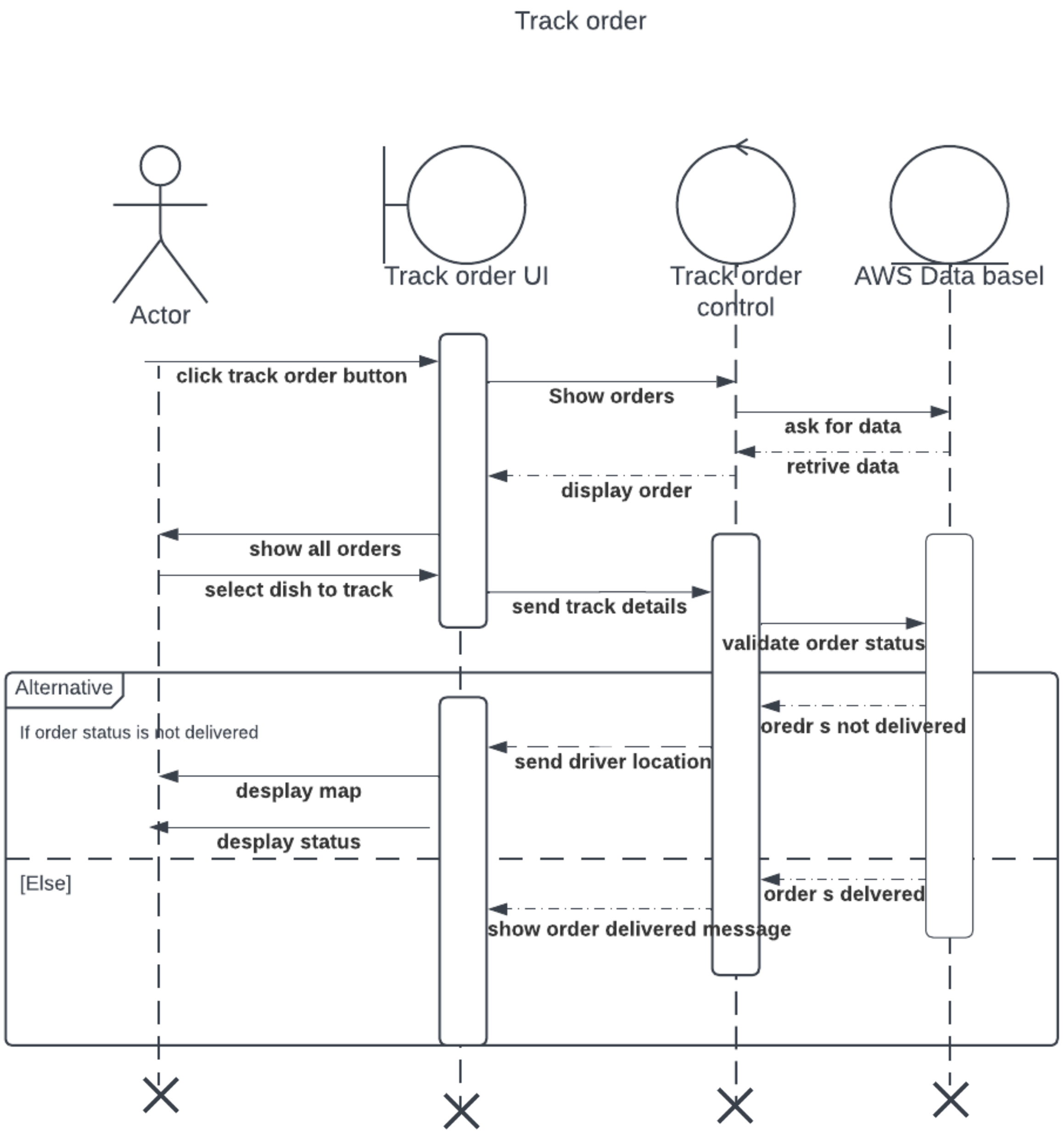


Figure 12: Track order Sequence Diagram

3.9 Activity Diagram

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. [13]

A. Activity Diagram for Register

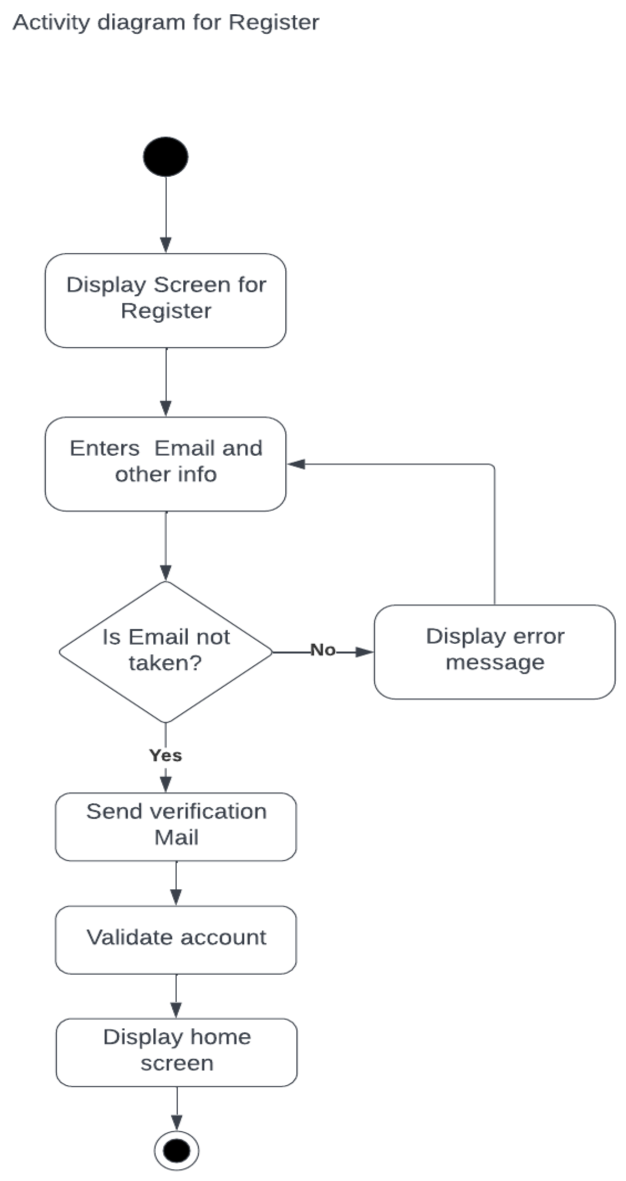


Figure 13: Register Activity Diagram

B. Activity Diagram for Log In

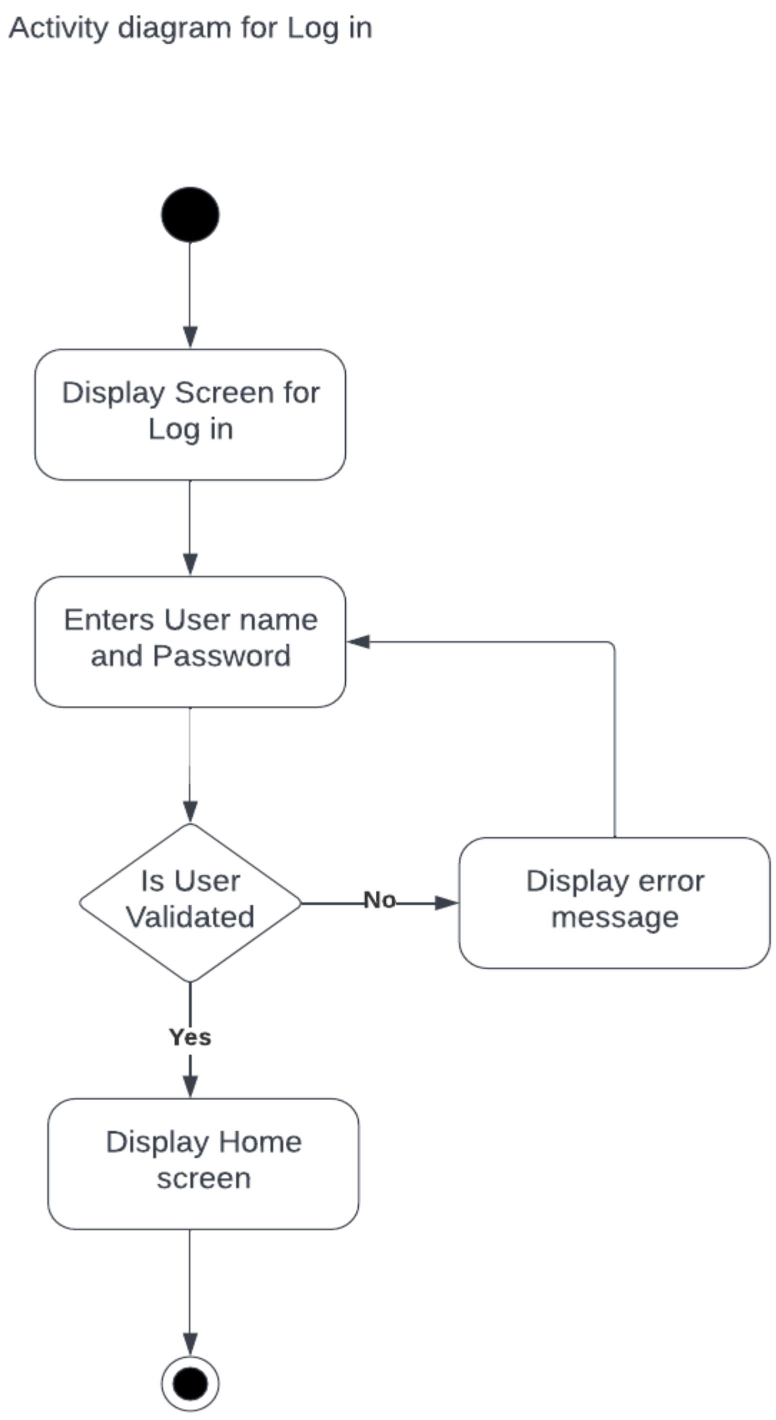

Activity diagram for Log in
Display Screen for Log in
Enters User name and Password
Is User Validated
No
Display error message
Yes
Display Home screen

Figure 14: Log in Activity Diagram

C. Activity Diagram for Manage Restaurant

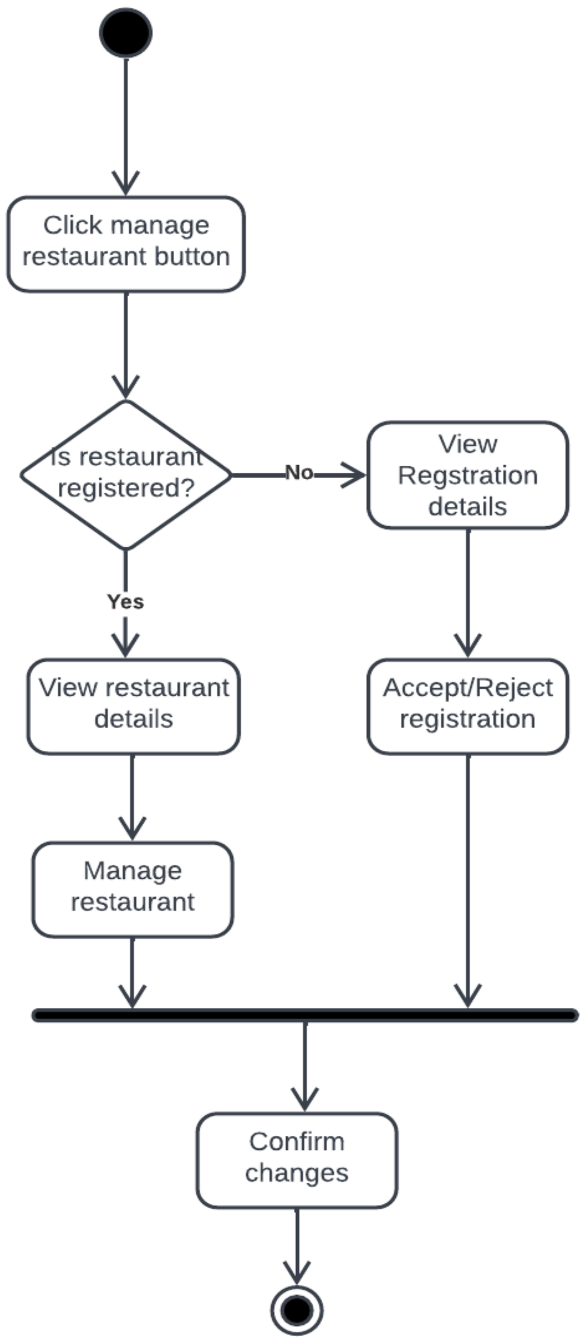


Figure 15: Manage Restaurant Activity Diagram

D. Activity Diagram for Manage courier

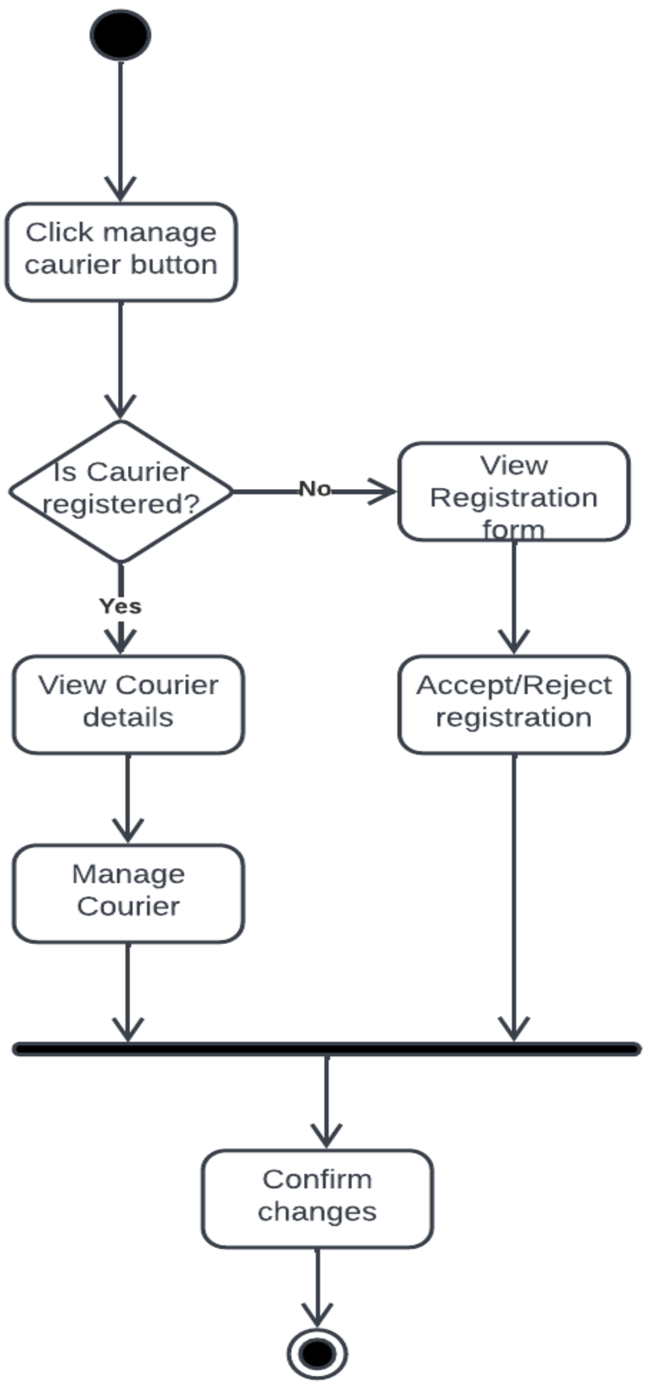


Figure 16: Manage Courier Activity Diagram

E. Activity Diagram for Accept or Decline Order

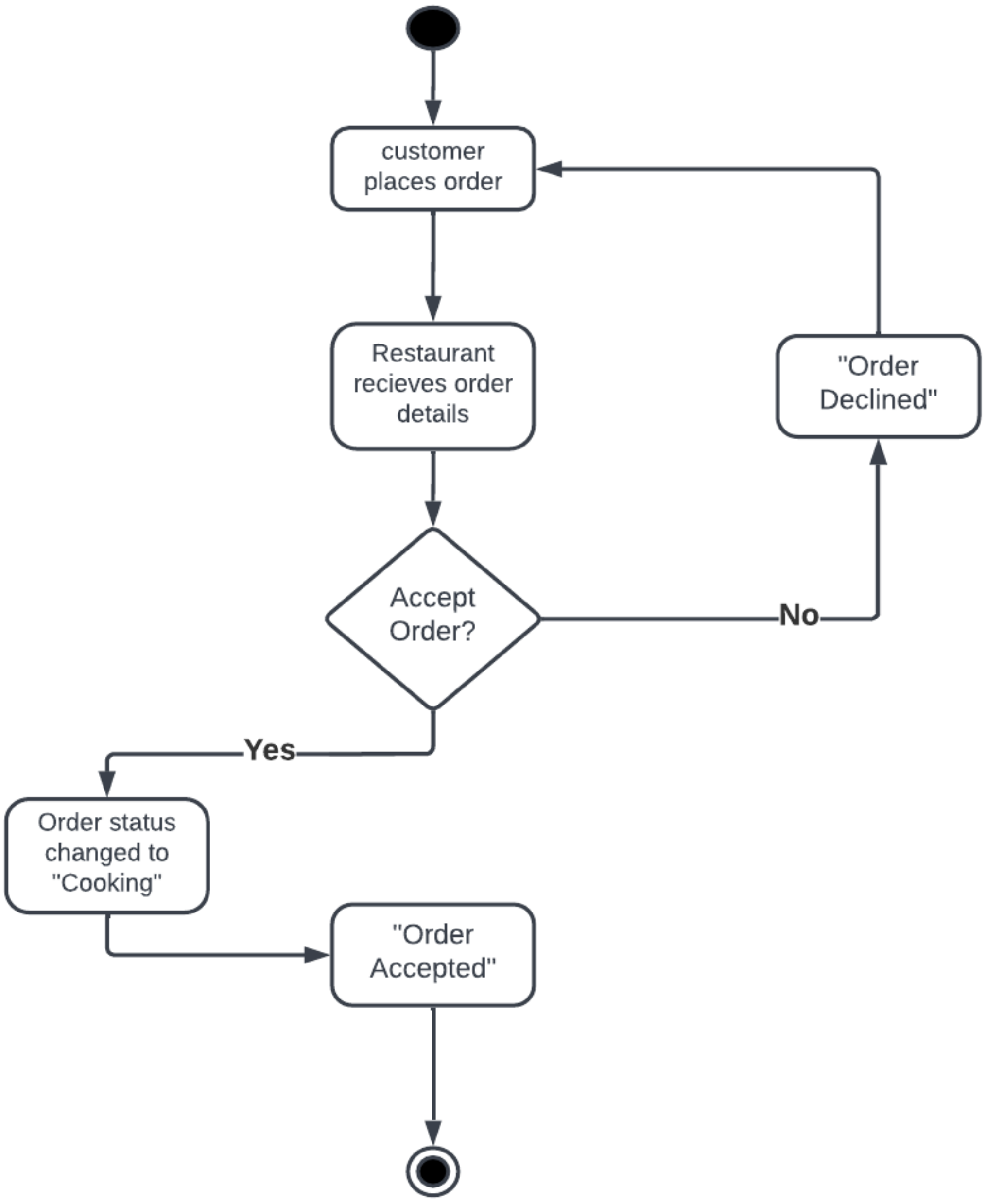


Figure 17: Accept or Decline Order Activity Diagram

F. Activity Diagram for Status Change

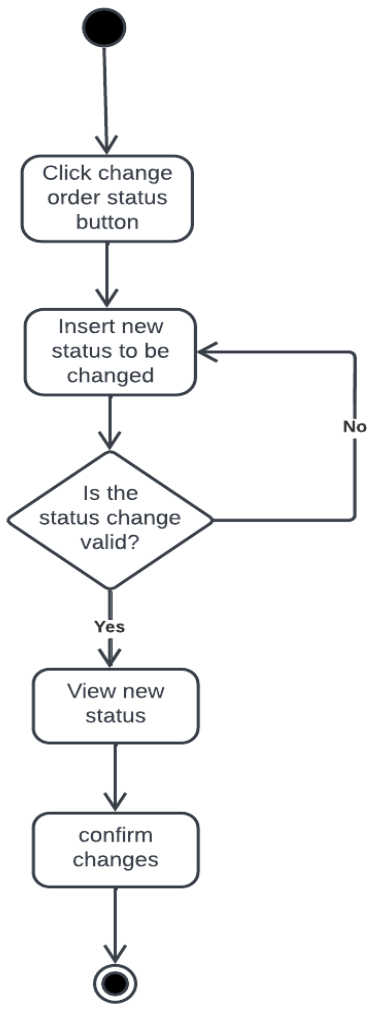

Click change order status button
Insert new status to be changed
Is the status change valid?
No
Yes
View new status
confirm changes

Figure 18: Change Status Activity Diagram

3.10 User Interface Prototype

A prototype is a simulation of the real system. A prototype is essentially a tool for identifying and fine-tuning requirements. User interface prototypes are used to test the usability of the interface. The diagrams listed below are low-fidelity User Interface diagrams.

A. Customer User Interface prototype Diagram

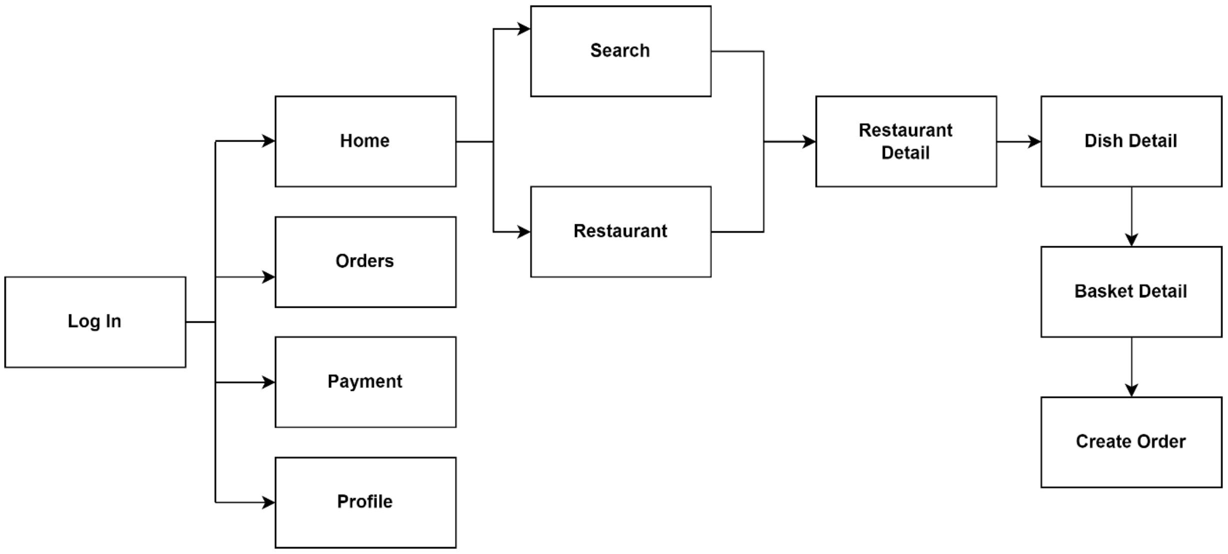


Figure 19 UI prototyping for customer app

B. Restaurant User Interface prototype Diagram

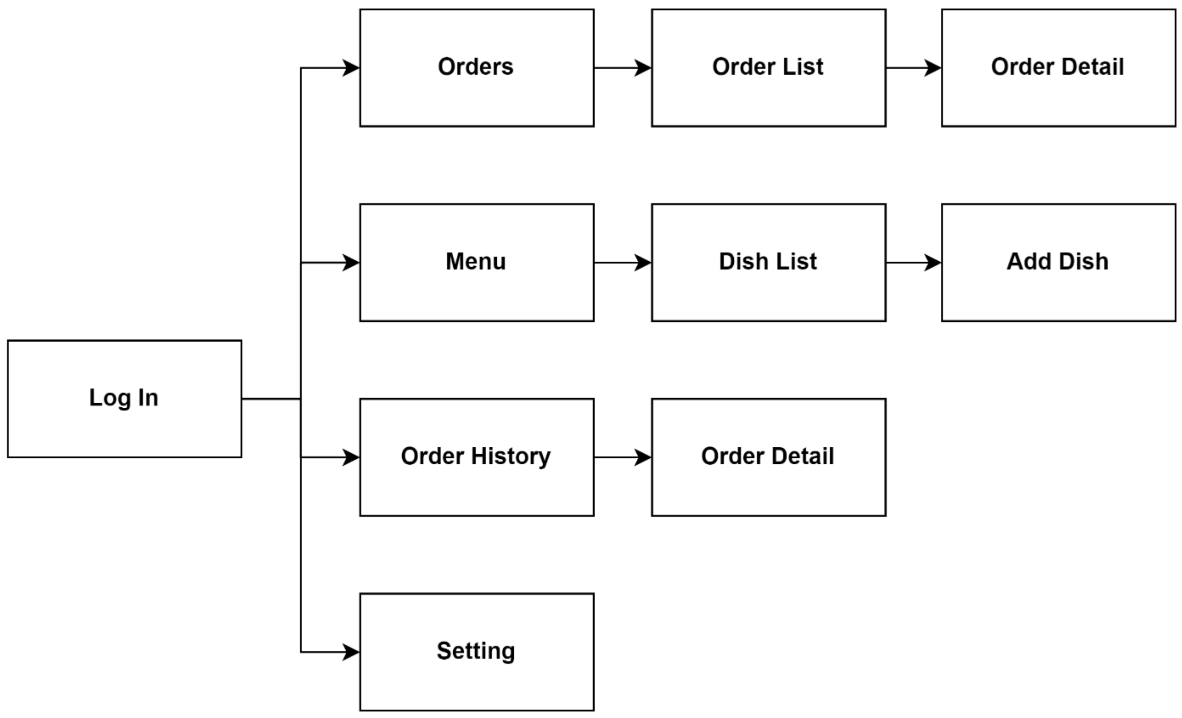


Figure 20: UI prototyping for restaurant

C. Courier User Interface prototype Diagram

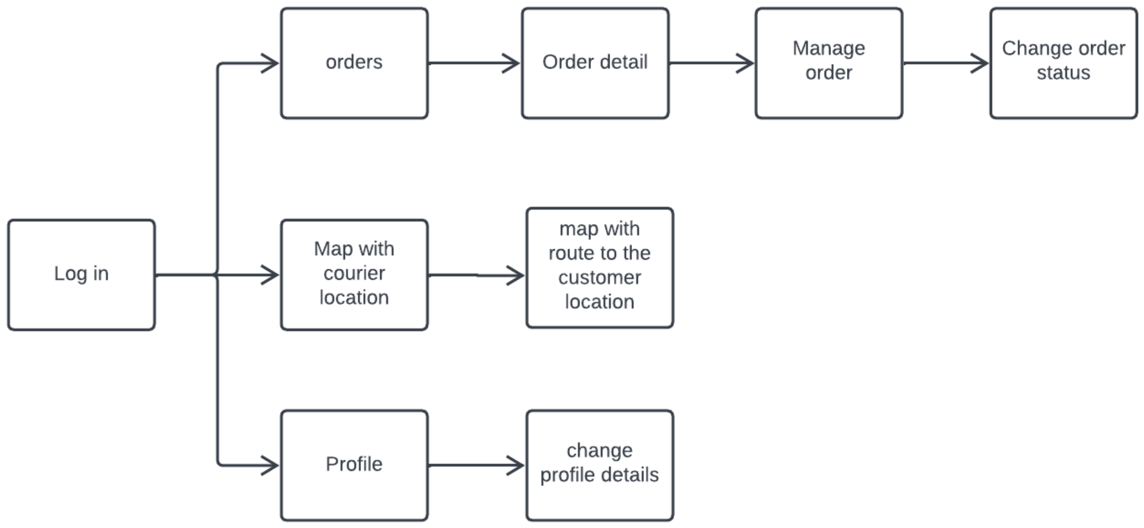


Figure 21 UI Prototyping for Courier

D. Courier User Interface Prototype Diagram

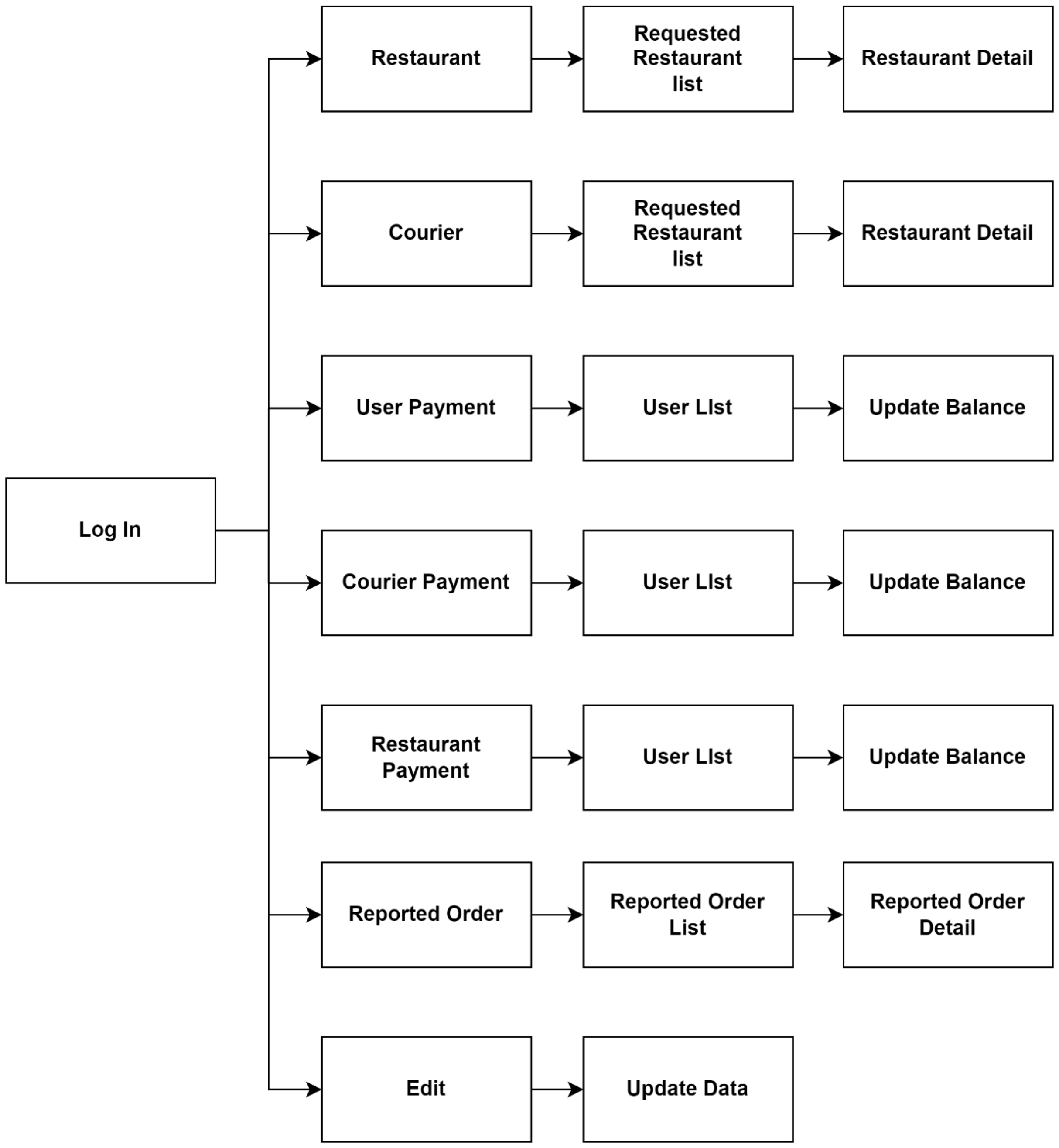


Figure 22 UI Prototyping for Administrator

Chapter 4

4. System Design

4.1 Introduction

The conversion of the analytical model into a system design model is known as systems design. The design aspect of the cloud based food ordering and delivery system is the main topic of this chapter. This chapter's goals are to give a general overview of how to construct the suggested system and to gather the data required to determine how our system will really be implemented. We will examine various system modeling strategies in this part, including class diagrams, collaboration diagrams, state chart diagrams, component diagrams, deployment diagrams, and persistence diagrams.

4.2 Class type architecture

The entire layout and structure of a computer network or system are referred to as the system architecture. There are many different physical devices included, so a system is needed to arrange and link them in a logical way. Furthermore, sophisticated computer software tools are referred to by this name. A generic discipline termed "systems architecture" is used to handle "systems"—objects that now exist or will be created—in a way that encourages analysis of their structural characteristics. The conceptual model that specifies the structure, behavior, and other aspects of a system is known as its architecture.

- **User interface layer**
  UI layer, handles the interactions that users have with the software. It's the most visible layer and defines the application's overall look and presentation to the end-users. This is the tier that's most accessible, which anyone can use from their client device, like a desktop, laptop, mobile phone or tablet [1]

- **Controller/process layer**
  The application layer handles the main programs of the architecture. It includes the code definitions and most basic functions of the developed application. This is the layer that programmers spend most of their time in when working on the software. You can use this layer to implement specific coordination logic that doesn't align exactly with either the presentation or business layer [1]

- **Business/Domain layer**

  The business layer, also called the domain layer, is where the application's business logic operates. Business logic is a collection of rules that tell the system how to run an application, based on the organization's guidelines. This layer essentially determines the behavior of the entire application. After one action finishes, it tells the application what to do next [1]

- **Persistence layer**

  The persistence layer, also called the data access layer, acts as a protective layer. It contains the code that's necessary to access the database layer. This layer also holds the set of codes that allow you to manipulate various aspects of the database, such as connection details and SQL statements [1]

- **Database layer**

  The database layer is where the system stores all the data. It's the lowest tier in the software architecture and houses not only data but indexes and tables as well. Search, insert, update and delete operations occur here frequently. Application data can store in a file server or database server, revealing crucial data but keeping data storage and retrieval procedures hidden [1]

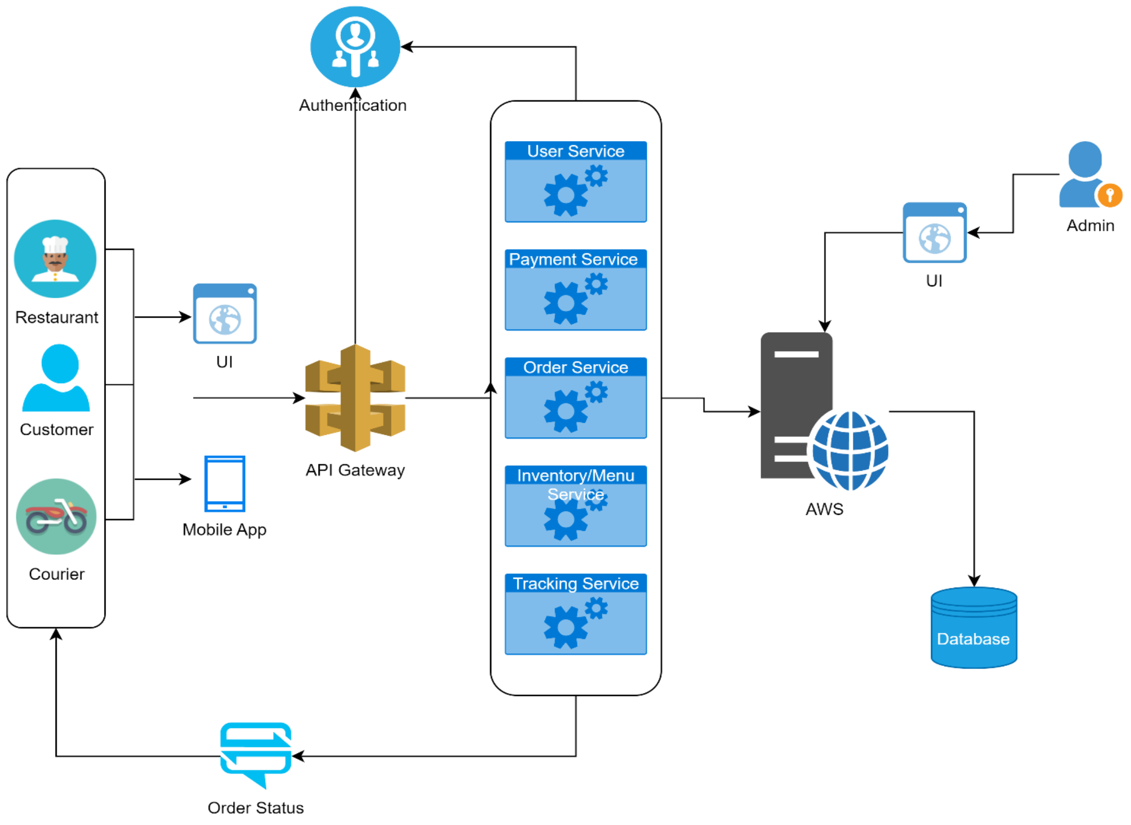


Figure 23: System architecture

4.3. Class modeling

A class diagram is a pictorial representation of the detailed system design. It is made up of a set of classes and interfaces that represent significant business domain entities for the system being modelled, as well as the interactions and connections between those classes and interfaces. Classes are placed in groups according to shared traits in a class diagram. In a class diagram, classes are represented as boxes with three rectangles inside each box, like a flowchart. The class name is in the top rectangle, the class's attributes are in the middle rectangle, and the class's methods, which are also known as operations, are in the bottom rectangle. The boxes are connected by lines, which could include arrows at one or both ends. These lines define the relationships, also called associations, between the classes.

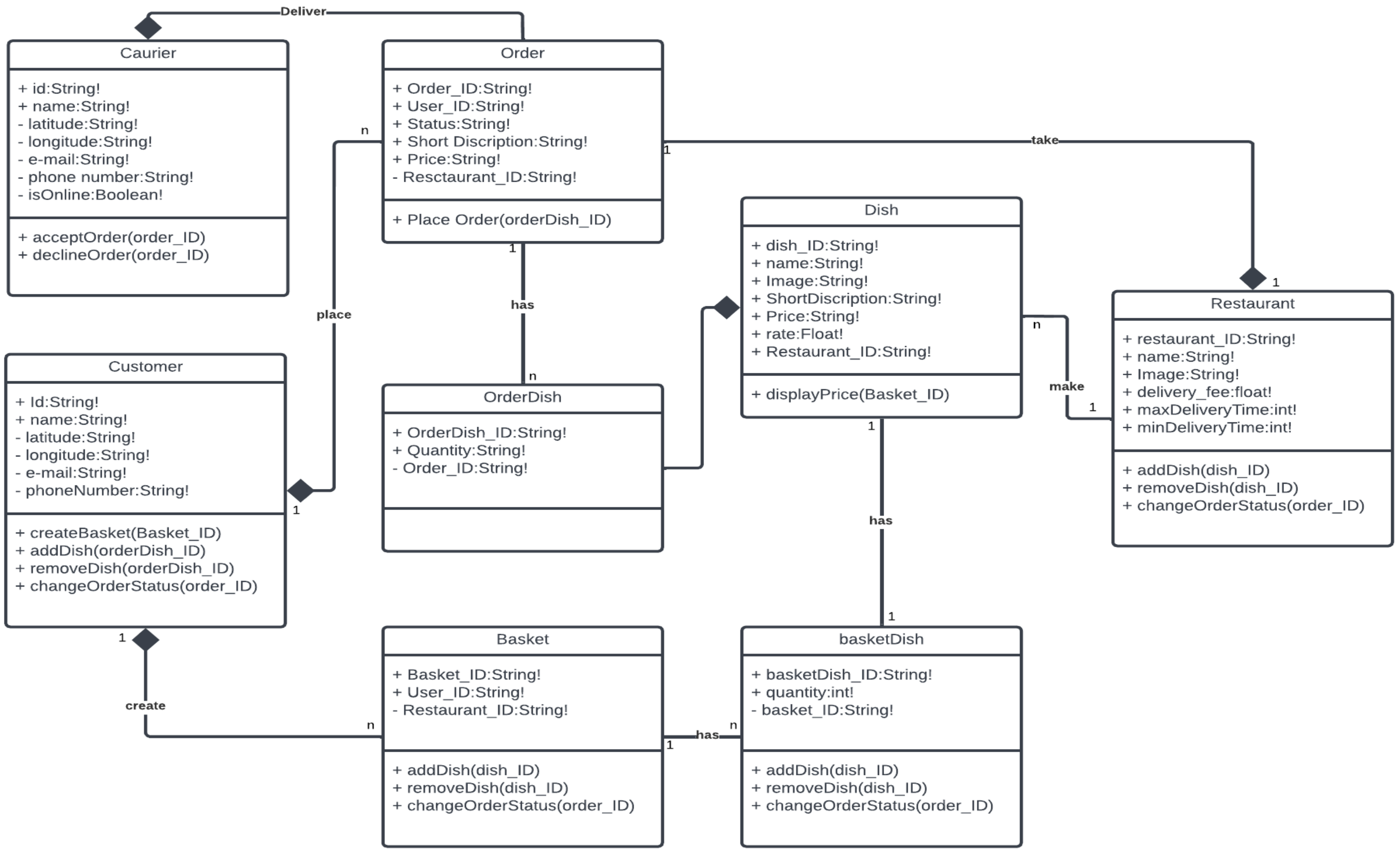


Figure 24: Class diagram

4.4. State chart diagram

A state diagram, sometimes referred to as a state machine diagram or a state chart diagram, is a visual depiction of the states and transitions that an object in the Unified Modeling Language may experience (UML). An object's "state," which in this context refers to a specific entity in a program or the unit of code that represents that thing, specifies a stage in its development or activity. State diagrams are useful for many kinds of object-oriented programming (OOP). Even if the concept predates 10 years, it has improved as OOP modelling paradigms have developed. In a state diagram, the initial stage is denoted by a large black dot, while the next phases are depicted as boxes with rounded corners. One or two horizontal lines may separate the box into stacked sections. In this case, the state's name appears at the top, any appropriate middle part also contains state variables, and the state's activities appear in the lower portion. If there are no horizontal lines across a box, only the state's name is written inside of it. External straight lines with arrows at either end join several pairs of boxes together. These lines explain the changes between the states. The final state is represented as a significant.

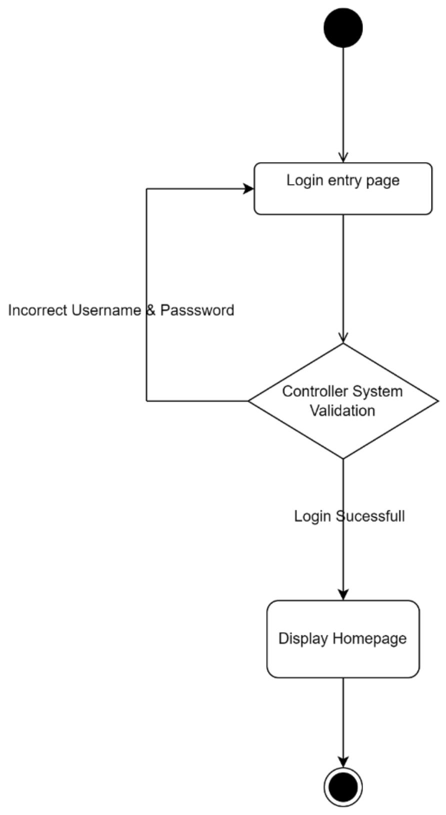


Figure 25: State chart diagram for login

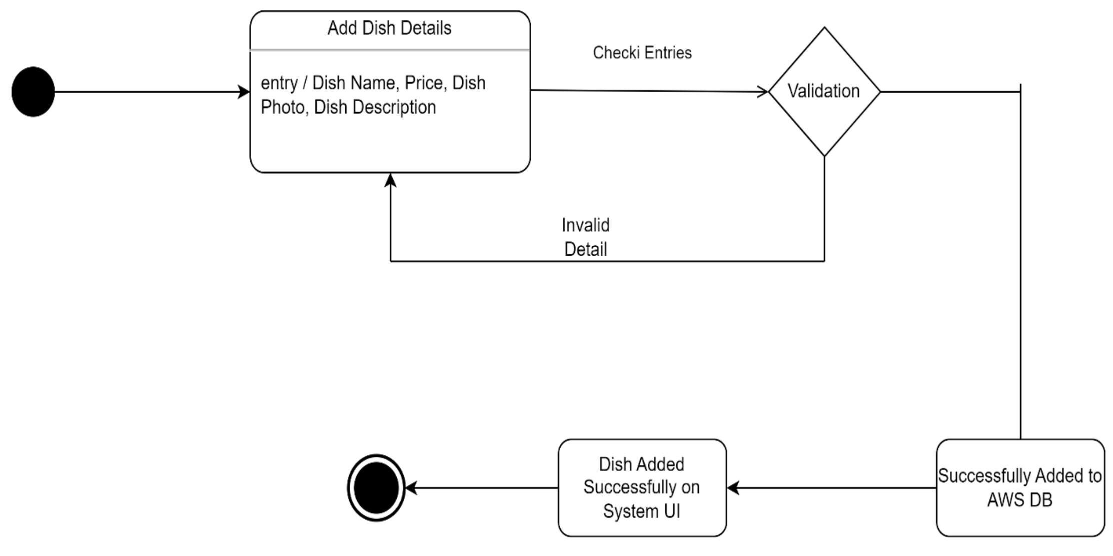

Add Dish Details
entry / Dish Name, Price, Dish Photo, Dish Description
Checki Entries
Validation
Invalid Detail
Successfully Added to AWS DB
Dish Added Successfully on System UI

Figure 26: State chart diagram for add dish

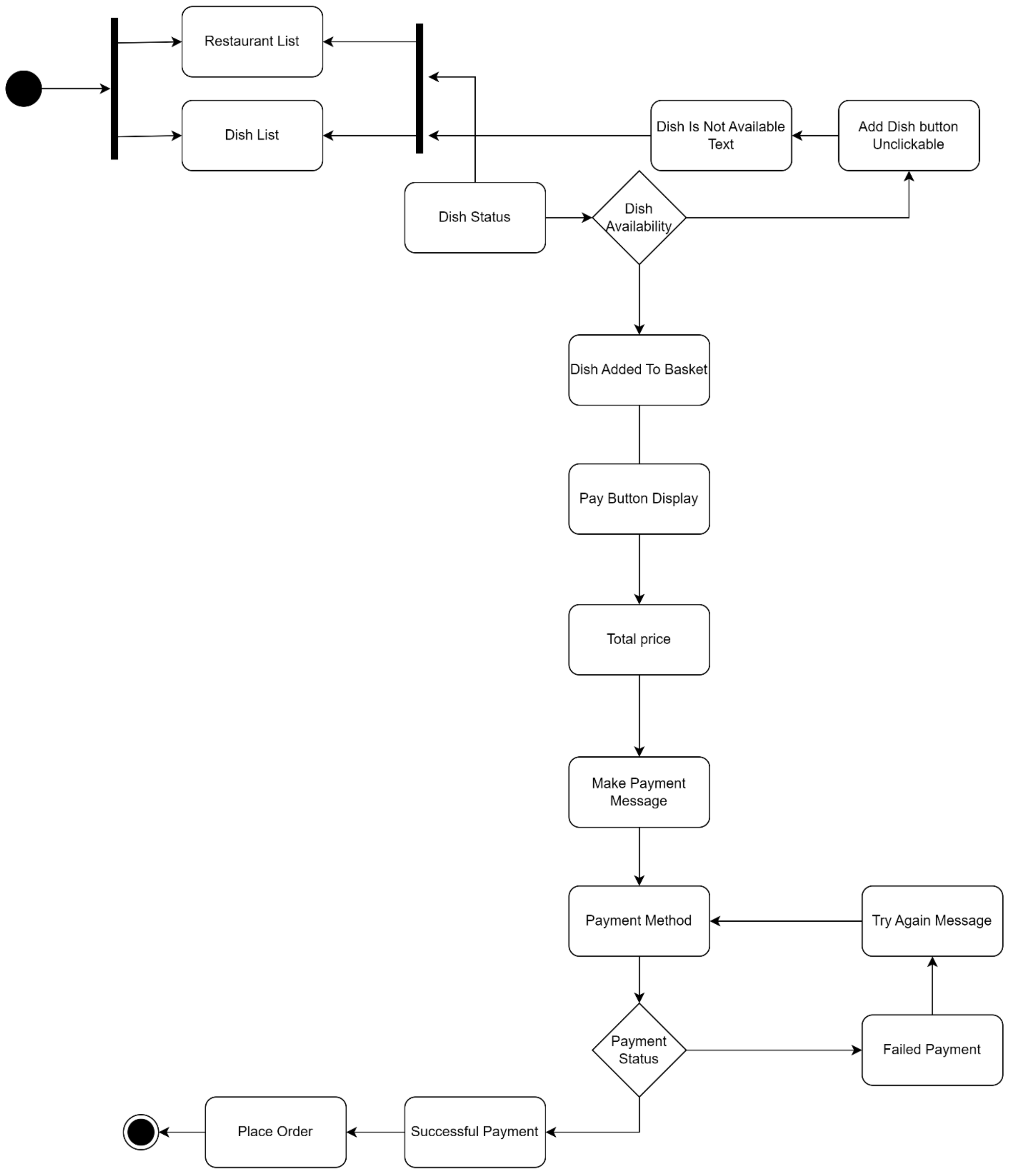


Figure 27: State chart diagram for place order

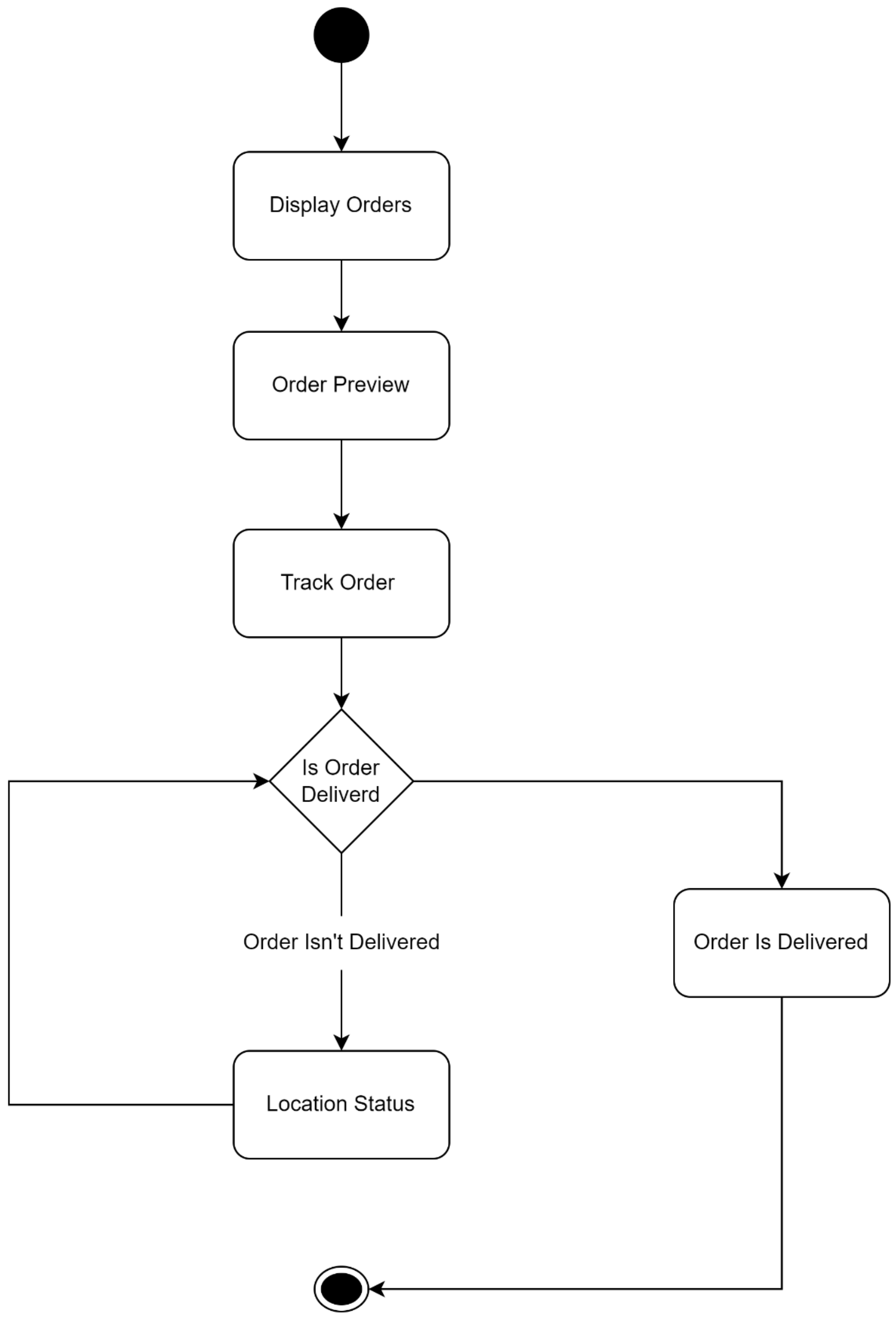


Figure 28: State chart diagram for track order

4.5. Collaboration Diagram

The first stage appears as a large black dot on a state diagram, while the succeeding stages are represented by rounded-cornered boxes. The box may be divided into stacked portions by one or two horizontal lines. The state's name is displayed at the top, state variables are included in any suitable middle sections, and the state's activities are displayed in the lower section. In boxes without horizontal lines, only the name of the state is printed inside. Several pairs of boxes are connected by external straight lines with arrows at either end. These lines describe how the states differ from one another. The ultimate stage is depicted as being important. These labels are followed by colons and may be italicized. The links between the items are shown by lines linking the rectangles. The messages between the objects are represented as arrows connecting the relevant rectangles and labels showing the sequence of the communications between the elements. Collaboration diagrams are effective for showing simple interactions between a few components. Understanding a collaboration diagram may be difficult as the number of objects and messages grows. Numerous vendors offer software for creating and editing collaborative diagrams.

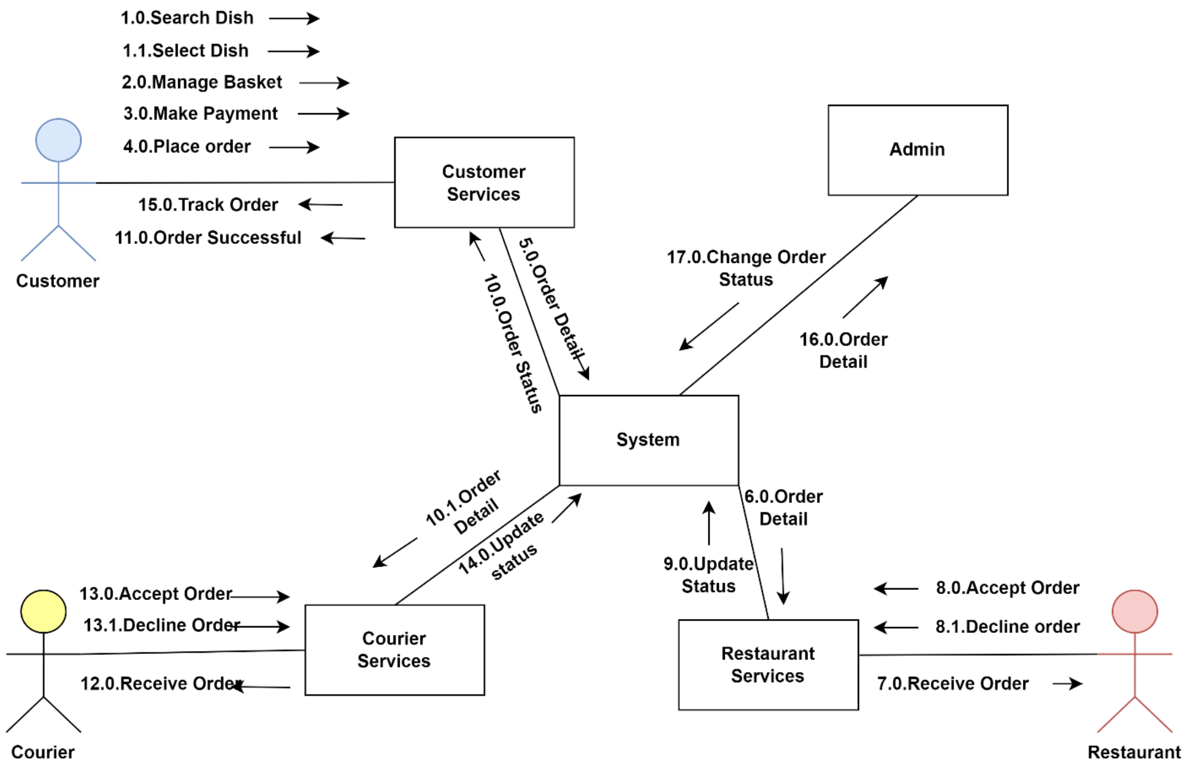


Figure 29: Collaboration Diagram of the System

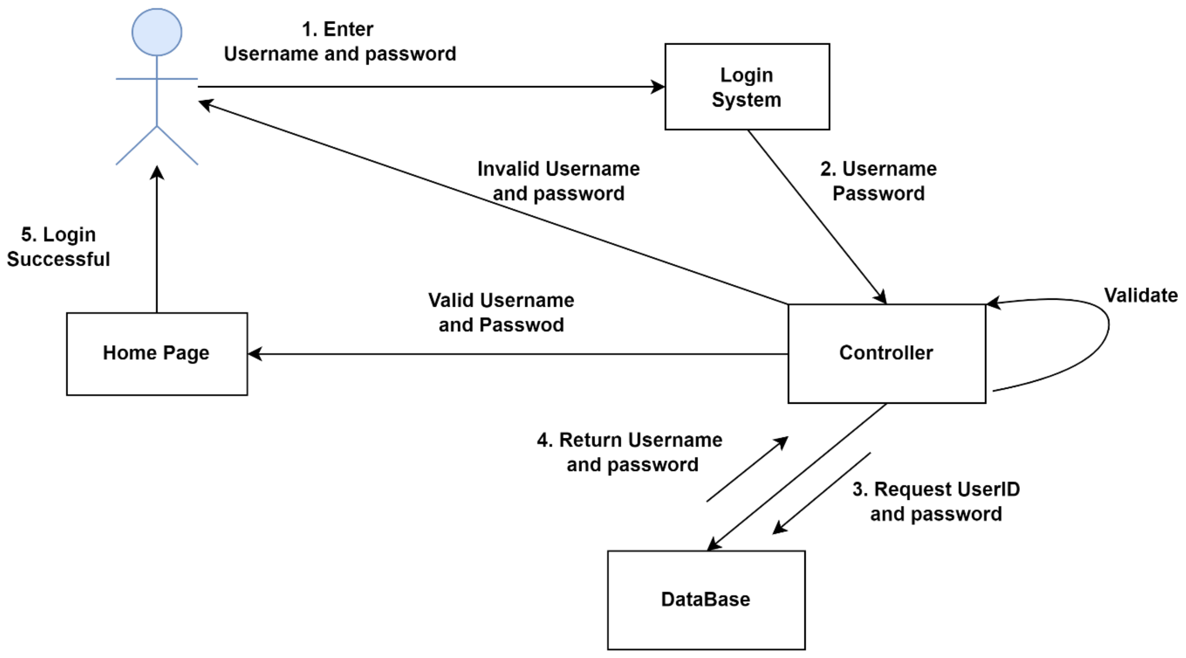

1. Enter
Username and password
Login System
Invalid Username and password
2. Username Password
5. Login Successful
Valid Username and Passwod
Validate
Home Page
Controller
4. Return Username and password
3. Request UserID and password
DataBase

Figure 30: Collaboration Diagram for login

4.6. Component diagram

The way that a system's pieces are physically arranged is shown on component diagrams. It also demonstrates the security infrastructures that are being used and the components or objects that are accessible to whom. The diagram is recreated below.

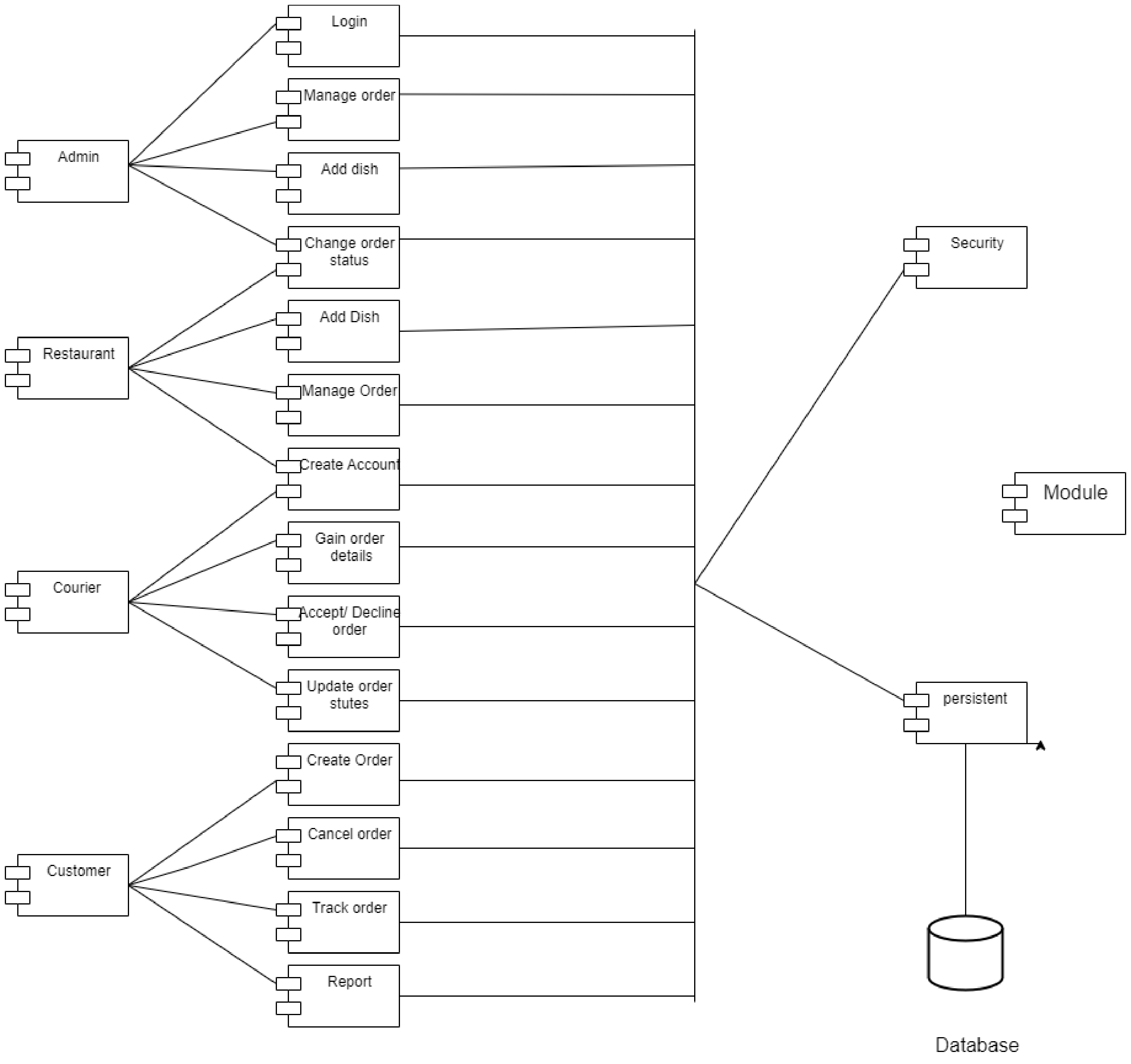


Figure 31: Component diagram

4.7. Deployment diagram

Deployment diagrams show how a system's physical components, where the software components are deployed, are organized. Deployment diagrams are used to depict a system's static deployment viewpoint. The key elements of deployment diagrams are nodes and connections between them.

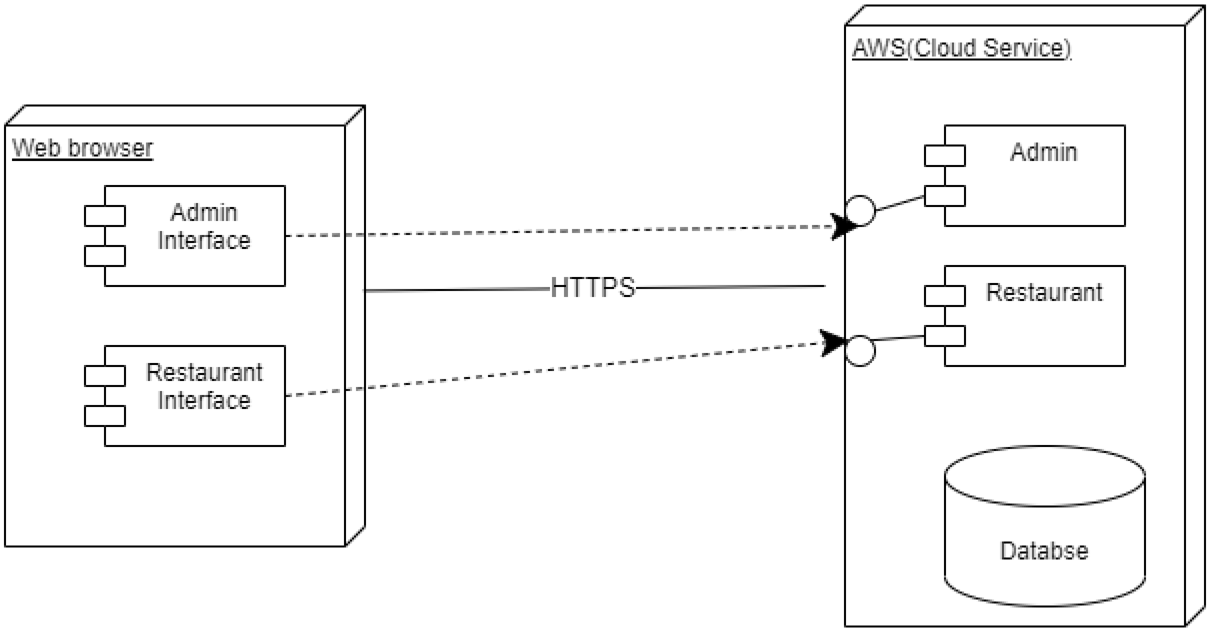


Figure 32:- Deployment diagram for Websites

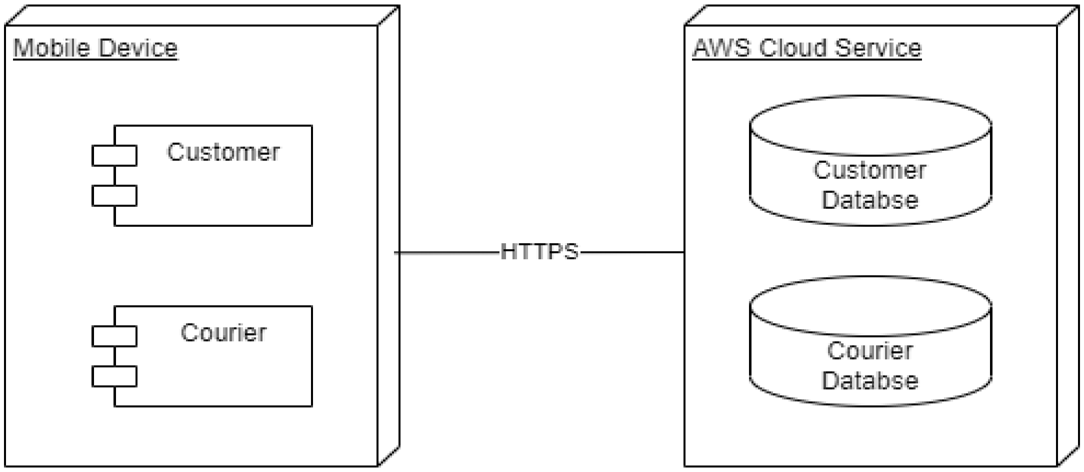


Figure 34: Deployment diagram for Mobile Applications

4.8. Persistence modeling

Persistence modelling is the method used to explain the design of the database, which is frequently the data base, to users and developers. It is also employed to describe the persistence data aspect of the system. The image below displays the persistence diagram for the system.

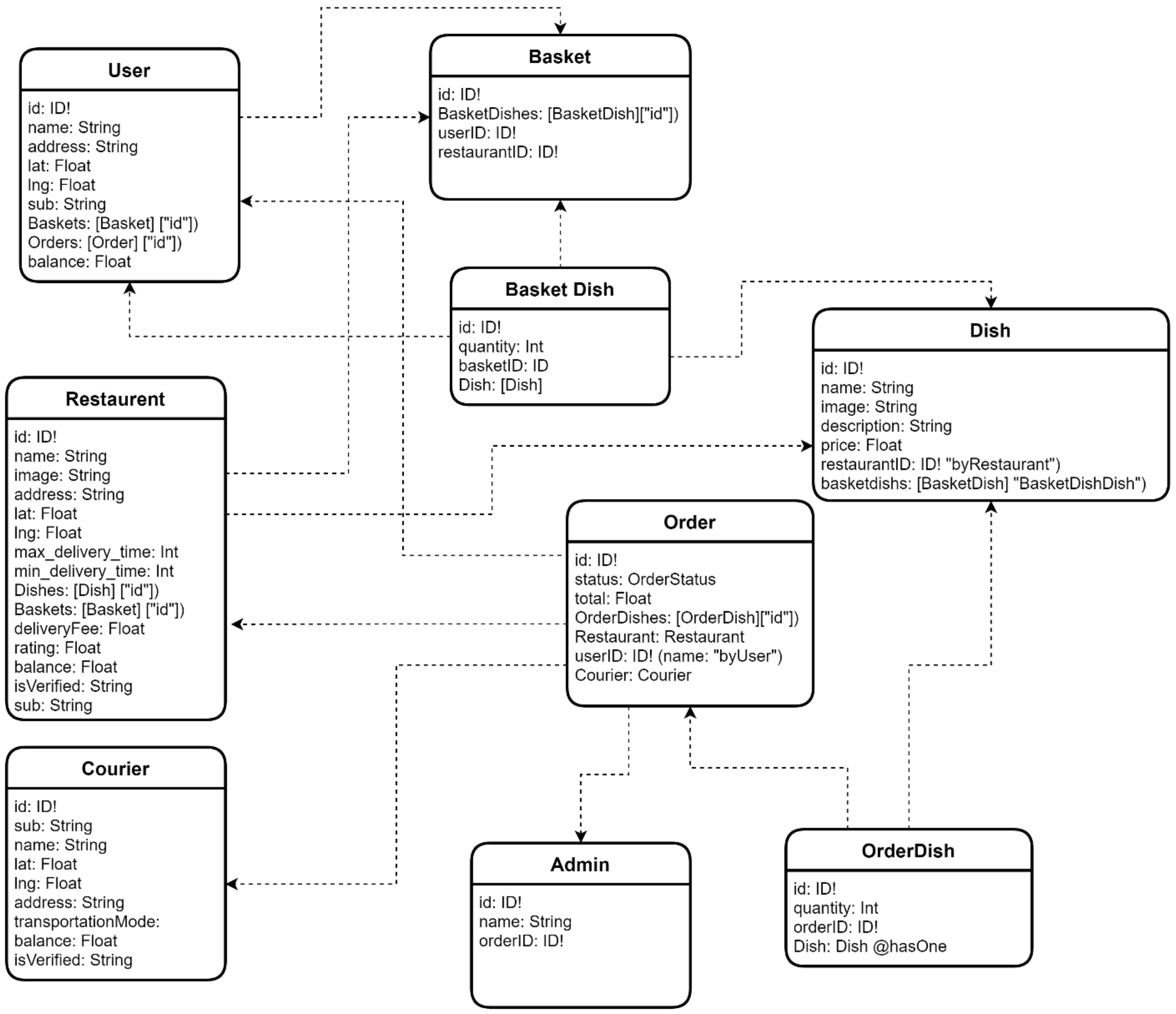


Figure 33 Persistence Diagram

4.9. User Interface design

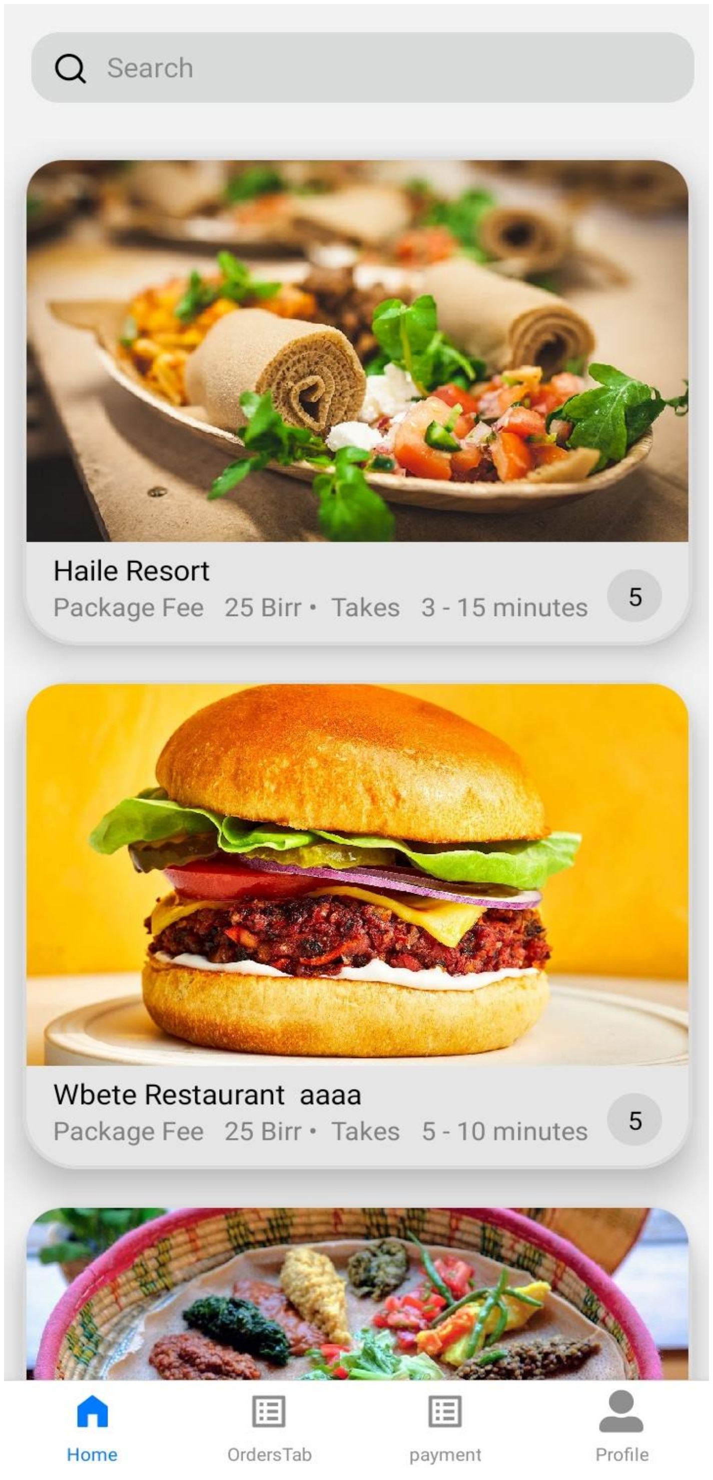
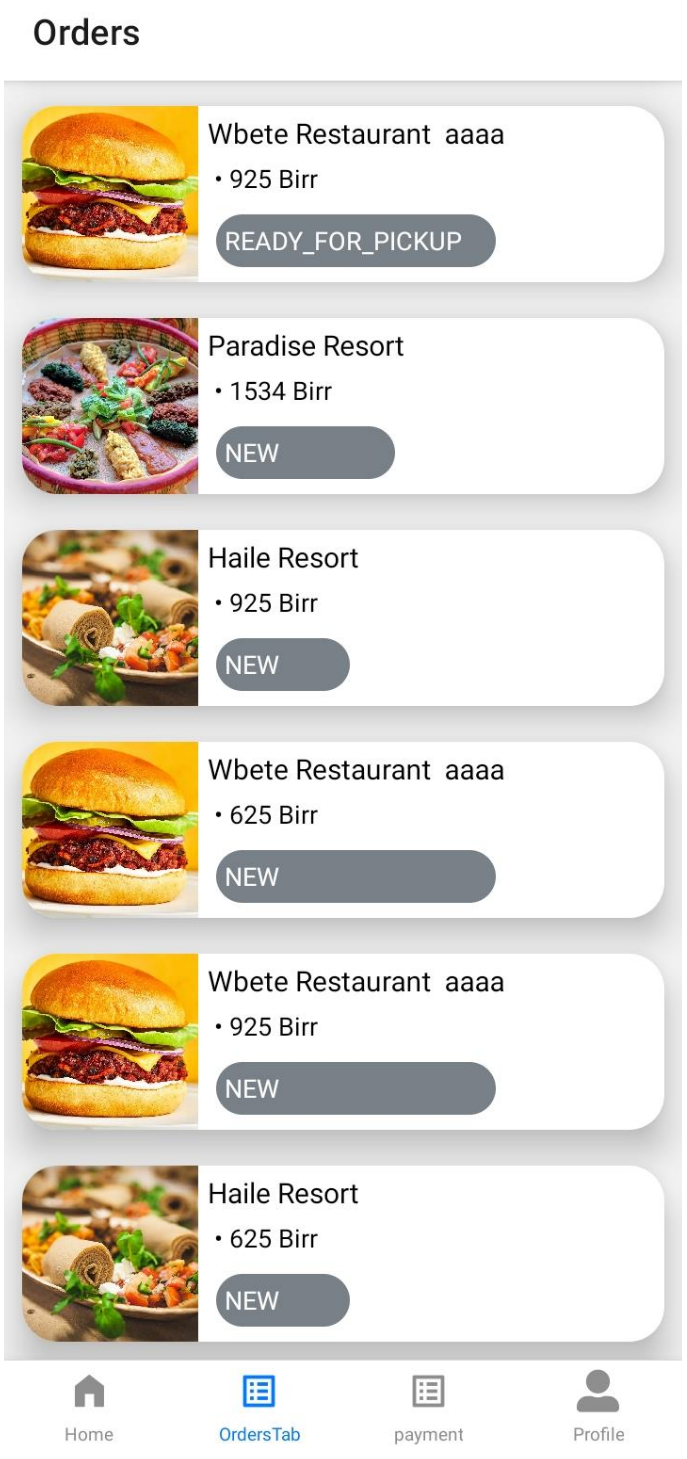

Search
Haile Resort
Package Fee 25 Birr • Takes 3 - 15 minutes
5
Wbete Restaurant aaaa
Package Fee 25 Birr • Takes 5 - 10 minutes
5
Home
OrdersTab
payment
Profile

Figure 34: Home screen for the user app


Orders

Wbete Restaurant aaaa
• 925 Birr
READY_FOR_PICKUP

Paradise Resort
• 1534 Birr
NEW

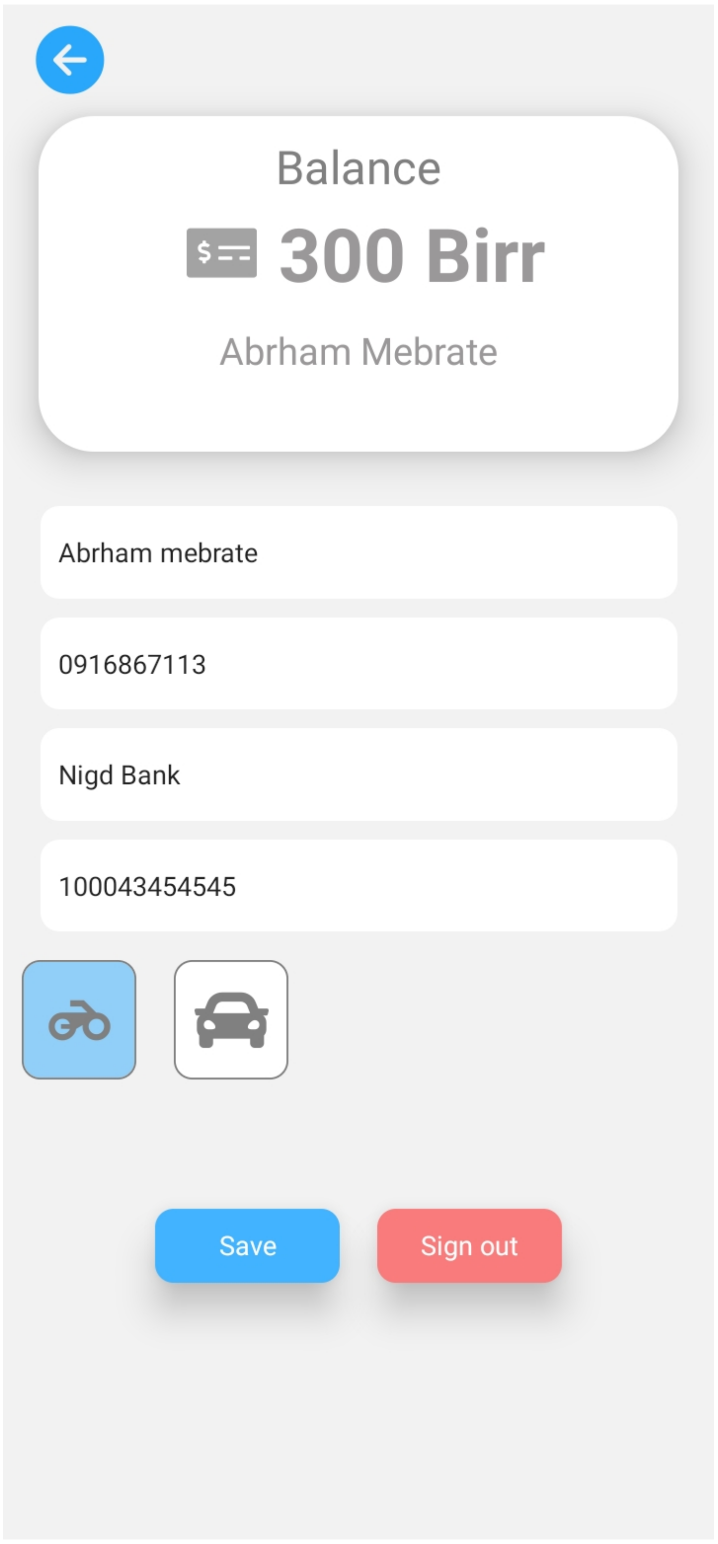
Haile Resort
• 925 Birr
NEW

Wbete Restaurant aaaa
• 625 Birr
NEW

Wbete Restaurant aaaa
• 925 Birr
NEW

Haile Resort
• 625 Birr
NEW

Figure 35 Orders page for the user app


Balance

300 Birr

Abrham Mebrate

Abrham mebrate

0916867113

Nigd Bank

100043454545

Figure 36 Profile screen for the courier app

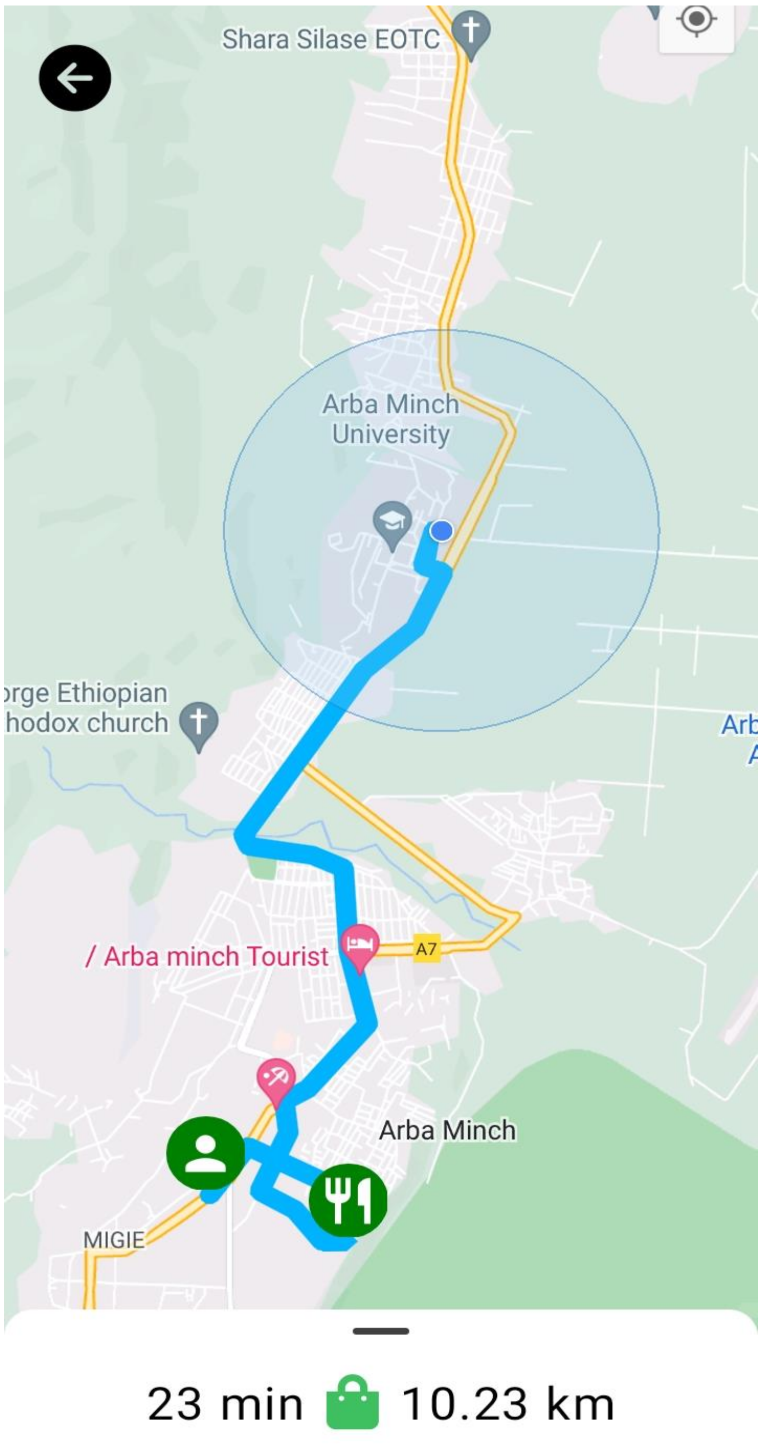
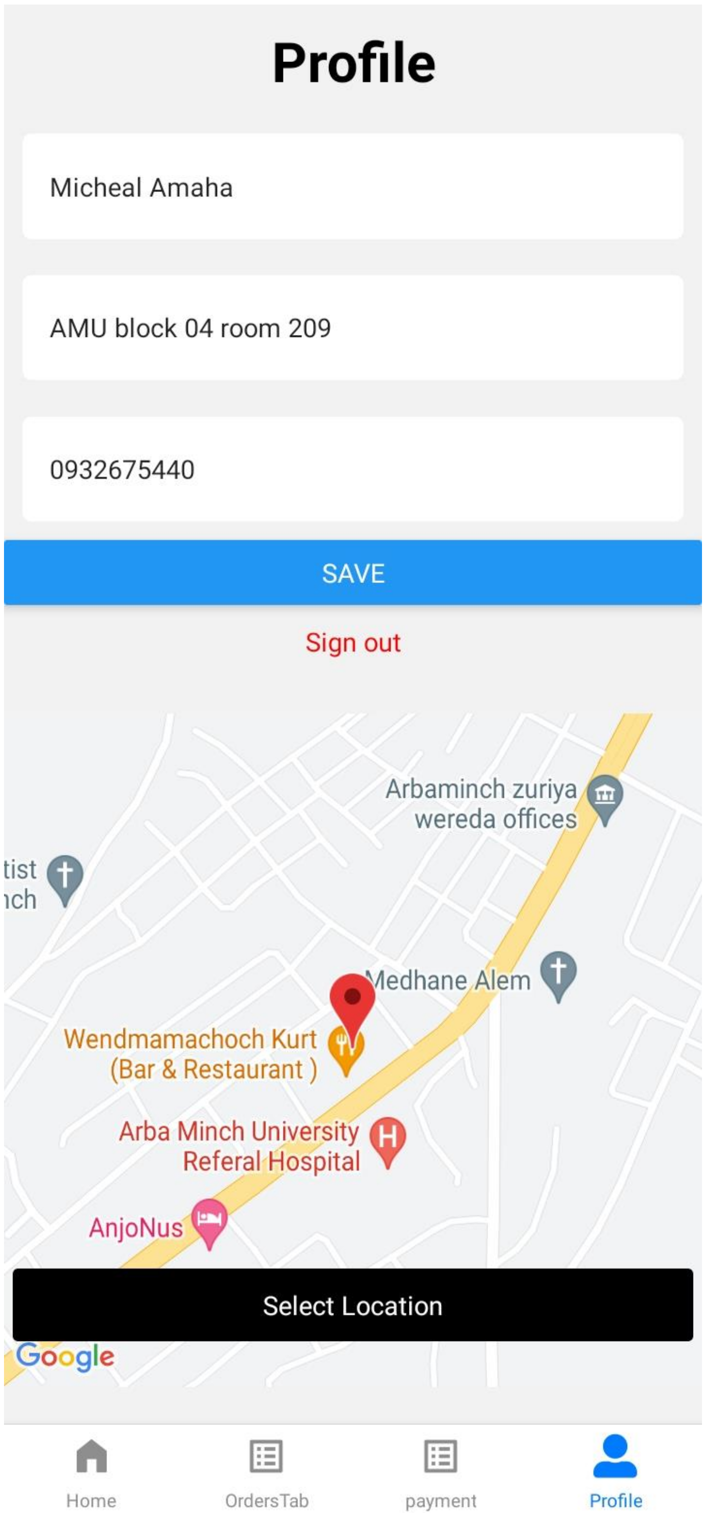


Figure 37 Order Detail


Profile

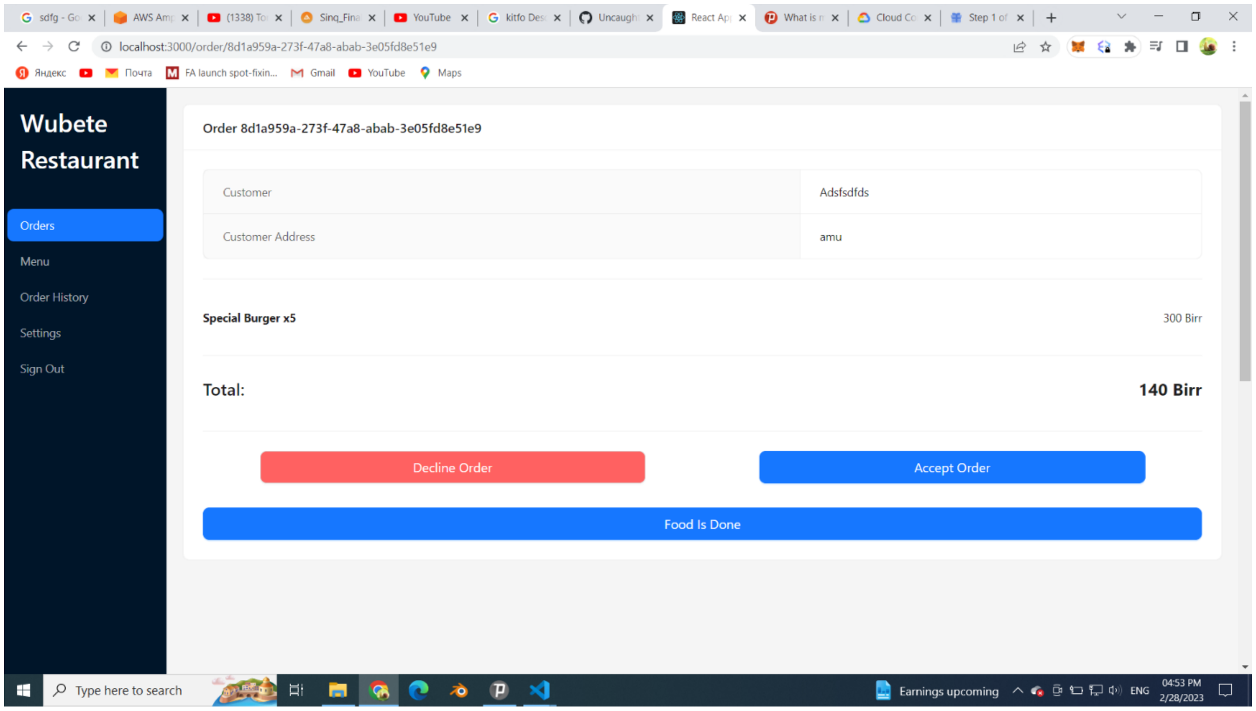
Micheal Amaha

AMU block 04 room 209

0932675440

SAVE
Sign out
Arbaminch zuriya wereda offices
tist
ıch
Medhane Alem
Wendmamachoch Kurt (Bar & Restaurant)
Arba Minch University Referal Hospital
AnjoNus
Select Location
Google
Home
OrdersTab
payment
Profile

Figure 38 Profile screen for user app



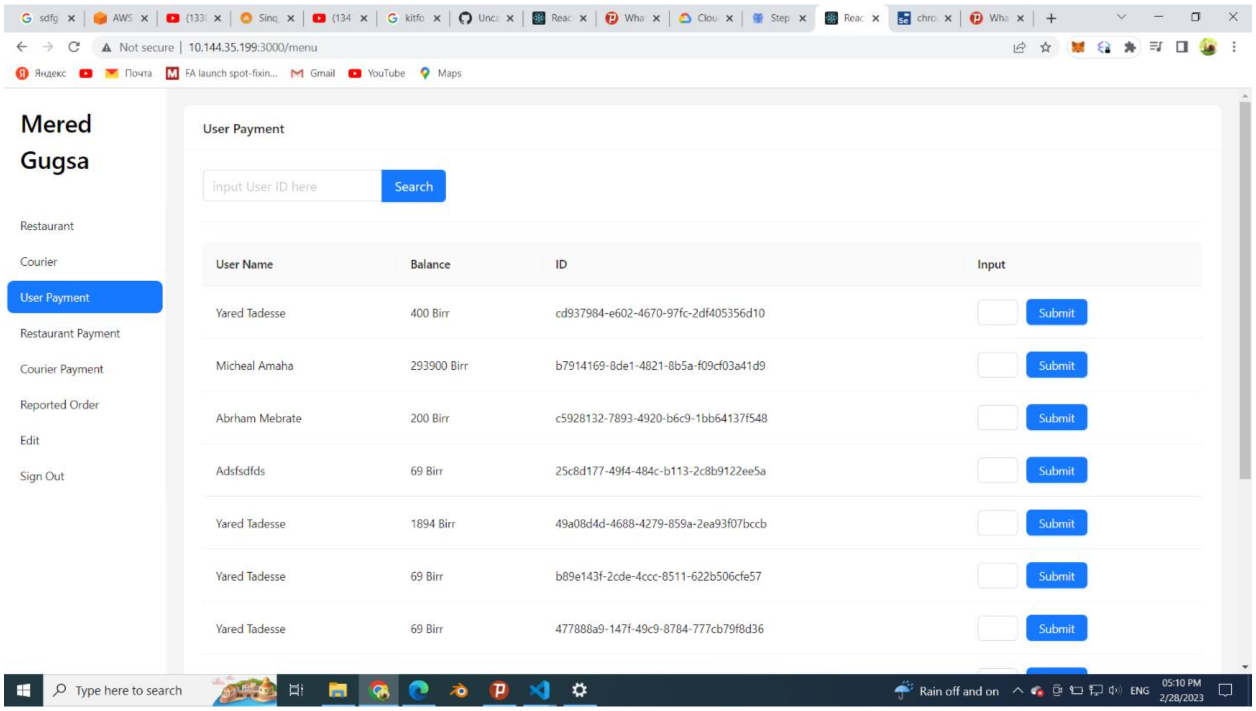
Order 8d1a959a-273f-47a8-abab-3e05fd8e51e9

Customer	Adsfsdfds
Customer Address	amu

Special Burger x5 300 Birr

Total: 140 Birr

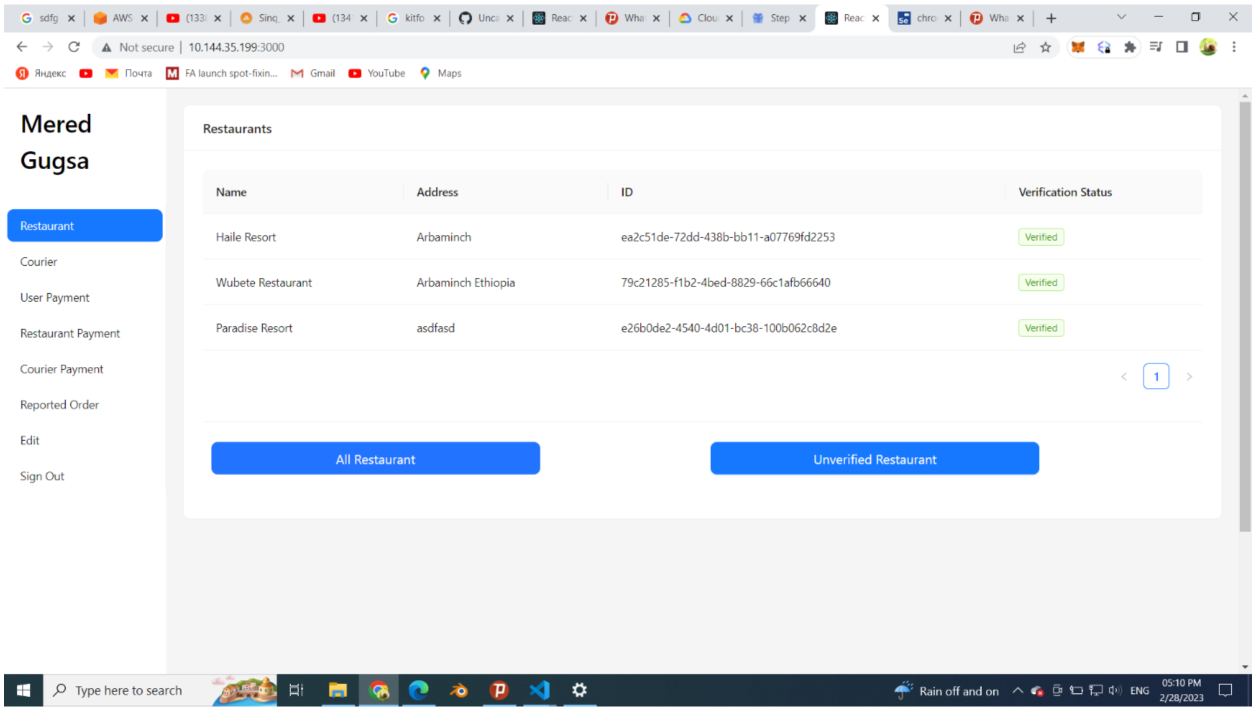
Figure 39 Order Details screen of Restaurant web app



User Payment

User Name	Balance	ID	Input
Yared Tadesse	400 Birr	cd937984-e602-4670-97fc-2df405356d10	Submit
Micheal Amaha	293900 Birr	b7914169-8de1-4821-8b5a-f09cf03a41d9	Submit
Abrham Mebrate	200 Birr	c5928132-7893-4920-b6c9-1bb64137f548	Submit
Adsfsdfds	69 Birr	25c8d177-49f4-484c-b113-2c8b9122ee5a	Submit
Yared Tadesse	1894 Birr	49a08d4d-4688-4279-859a-2ea93f07bccb	Submit
Yared Tadesse	69 Birr	b89e143f-2cde-4ccc-8511-622b506cfe57	Submit
Yared Tadesse	69 Birr	477888a9-147f-49c9-8784-777cb79f8d36	Submit

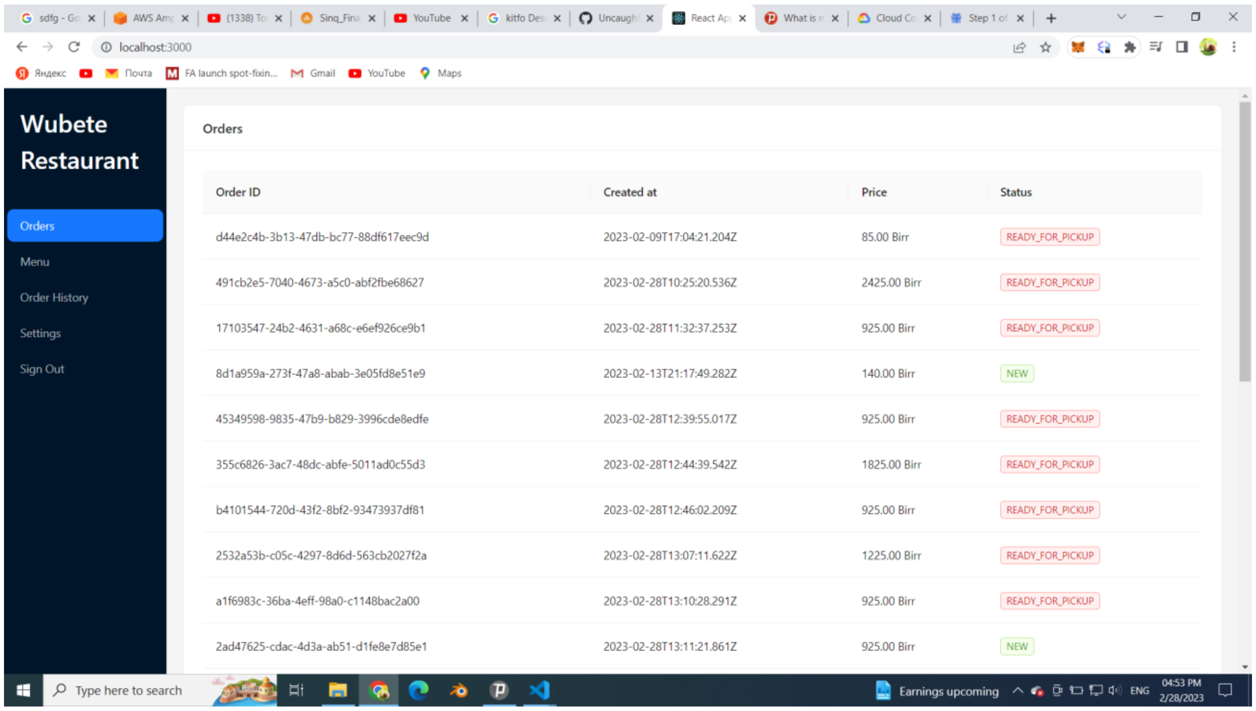
Figure 40 Manage User payment screen of Admin Web app



Restaurants

Name	Address	ID	Verification Status
Haile Resort	Arbaminch	ea2c51de-72dd-438b-bb11-a07769fd2253	Verified
Wubete Restaurant	Arbaminch Ethiopia	79c21285-f1b2-4bed-8829-66c1afb66640	Verified
Paradise Resort	asdfasd	e26b0de2-4540-4d01-bc38-100b062c8d2e	Verified

Figure 41 Manage Restaurant screen of admin web app



Orders

Order ID	Created at	Price	Status
d44e2c4b-3b13-47db-bc77-88df617eec9d	2023-02-09T17:04:21.204Z	85.00 Birr	READY_FOR_PICKUP
491cb2e5-7040-4673-a5c0-abf2fbe68627	2023-02-28T10:25:20.536Z	2425.00 Birr	READY_FOR_PICKUP
17103547-24b2-4631-a68c-e6ef926ce9b1	2023-02-28T11:32:37.253Z	925.00 Birr	READY_FOR_PICKUP
8d1a959a-273f-47a8-abab-3e05fd8e51e9	2023-02-13T21:17:49.282Z	140.00 Birr	NEW
45349598-9835-47b9-b829-3996cde8edfe	2023-02-28T12:39:55.017Z	925.00 Birr	READY_FOR_PICKUP
355c6826-3ac7-48dc-abfe-5011ad0c55d3	2023-02-28T12:44:39.542Z	1825.00 Birr	READY_FOR_PICKUP
b4101544-720d-43f2-8bf2-93473937df81	2023-02-28T12:46:02.209Z	925.00 Birr	READY_FOR_PICKUP
2532a53b-c05c-4297-8d6d-563cb2027f2a	2023-02-28T13:07:11.622Z	1225.00 Birr	READY_FOR_PICKUP
a1f6983c-36ba-4eff-98a0-c1148bac2a00	2023-02-28T13:10:28.291Z	925.00 Birr	READY_FOR_PICKUP
2ad47625-cdac-4d3a-ab51-d1fe8e7d85e1	2023-02-28T13:11:21.861Z	925.00 Birr	NEW

Figure 42 Order list screen of the restaurant web app

Chapter 5

5. Implementation and Testing

5.1 Introduction

Cloud based online food Ordering and delivery application is a digital platform that enables users to order food from their favorite restaurants and have it delivered to their location. The application is designed to provide a convenient and seamless experience for users who are looking to order food online and have it delivered to their doorstep.

Implementing a food delivery app requires careful planning and execution to ensure a smooth user experience and efficient backend processes. Developing a food delivery app involves a wide range of factors such as software and hardware requirements, app design and development, backend development and integration, payment processing, and user experience.

The purpose of the tests included are to ensure that the application meets its functional and non-functional requirements, and that it is ready for evaluation. It outlines the testing approach, test scenarios, and test cases that will be executed to verify that the application functions as intended, is user-friendly, and meets the needs of its users.

The testing effort will be performed by the development team, with the involvement of external testing resources as required. The test plan will cover functional, performance, usability, security, and compatibility testing to ensure that the application is reliable, efficient, and user-friendly.

This test plan is intended to be used as a reference by the development team and testing resources throughout the testing process. It is expected that all test cases will be executed, and any issues that are identified will be logged and tracked until they are resolved.

The successful completion of this testing effort will ensure that the application meets the expectations of its users and is ready for release to the public.

5.2 Implementation

The implementation of a cloud-based food delivery app using React and React Native on Amazon's Cloud Service with Google API integration requires careful planning and execution. React and React Native are powerful and efficient front-end technologies that provide a high-quality user interface and user experience. Amazon's Cloud Service provides a reliable and scalable backend infrastructure to host and manage the app and database in the cloud. Google API integration enables efficient and secure location-based services such as tracking delivery partners and calculating distances between the customer and the restaurant. The development team followed agile methodologies, maintained proper documentation, and ensured timely communication and collaboration to ensure a successful implementation of the food delivery app. By leveraging the latest technology and best practices, a food delivery app can be developed that delivers a seamless experience for customers and efficient operations for restaurant owners and delivery partners.

The previous phases of the system development are completed. In this stage, coding takes next position. Coding includes implementation of user interface, implementation of database and logical implementation. In the following sections implementation is discussed in context of the whole system. It includes hardware and software used for implementation as well as code snippets.

5.2.1 Hardware and Software Acquisitions

For the proper implementation of the system, the following hardware and software are required

Hardware Requirements:

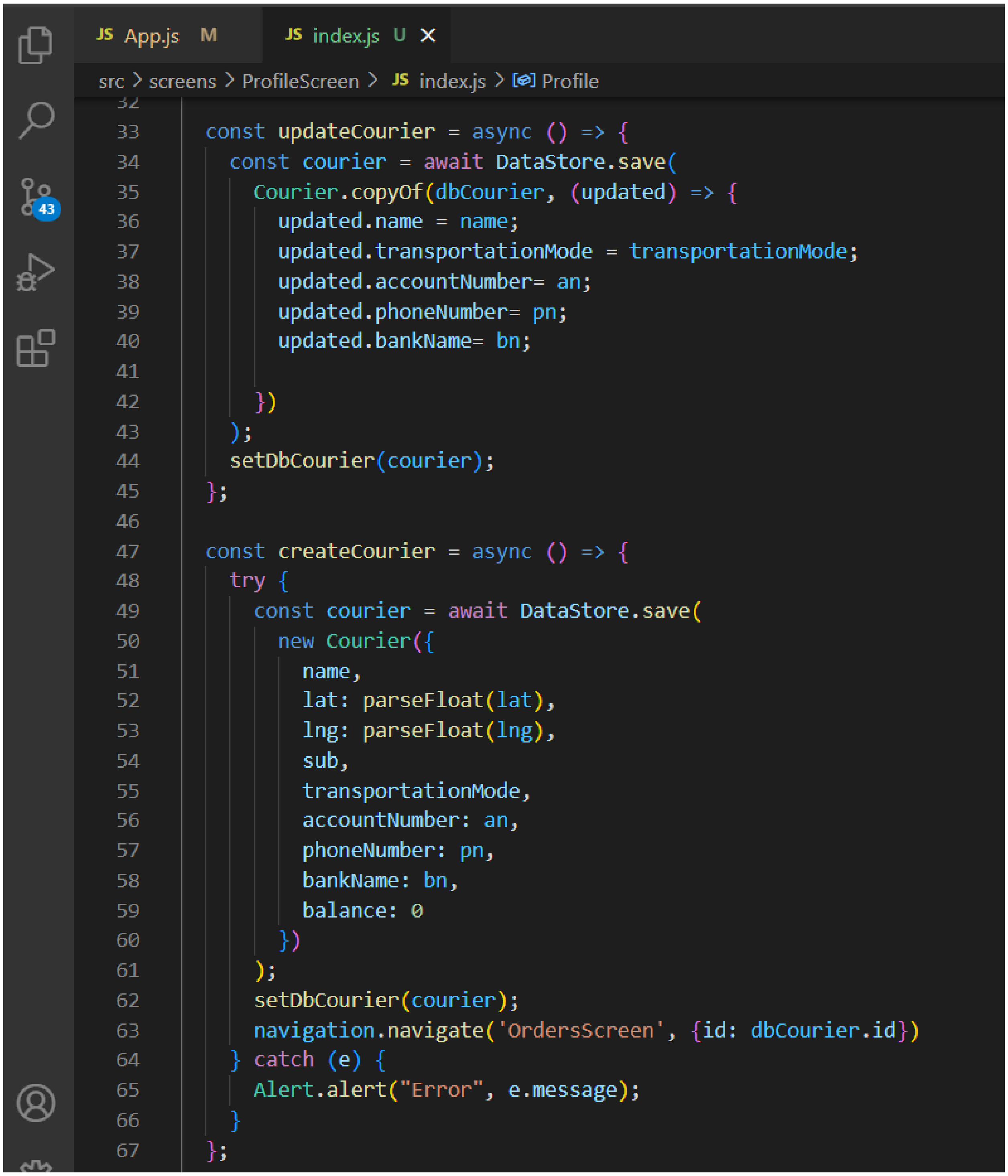
1. Server: A cloud hosting service such as Amazon Web Services is required to store and manage the application's data, including user profiles, order details, and payment information.
2. Storage: Sufficient storage is needed to accommodate the application's database and media files such as food images and videos.
3. Processor: A high-performance processor (minimum Intel CORE-i5) is recommended to ensure that the application runs smoothly without lagging or crashing.
4. Memory: Sufficient RAM (at least 8 GB) is necessary to handle multiple user requests and data processing simultaneously.

Software Requirements:

1. React Native: React Native should be installed and set up for mobile application development.
2. Node.js: Node.js should be installed on the development environment to build and run the application.

3. React Navigation: React Navigation should be integrated for smooth navigation between screens and pages.
4. Amazon Web Services: Amazon Web Services (AWS) should be set up and configured to host the application and database in the cloud.
5. Google API Gateway: Amazon API Gateway should be integrated to facilitate communication between the application and the serverless backend.
6. Amazon DynamoDB: Amazon DynamoDB should be used as the database management system for efficient and scalable data storage.
7. VS Code: VS Code should be used as the code editor for efficient and productive development.

5.2.2 Code Snippets

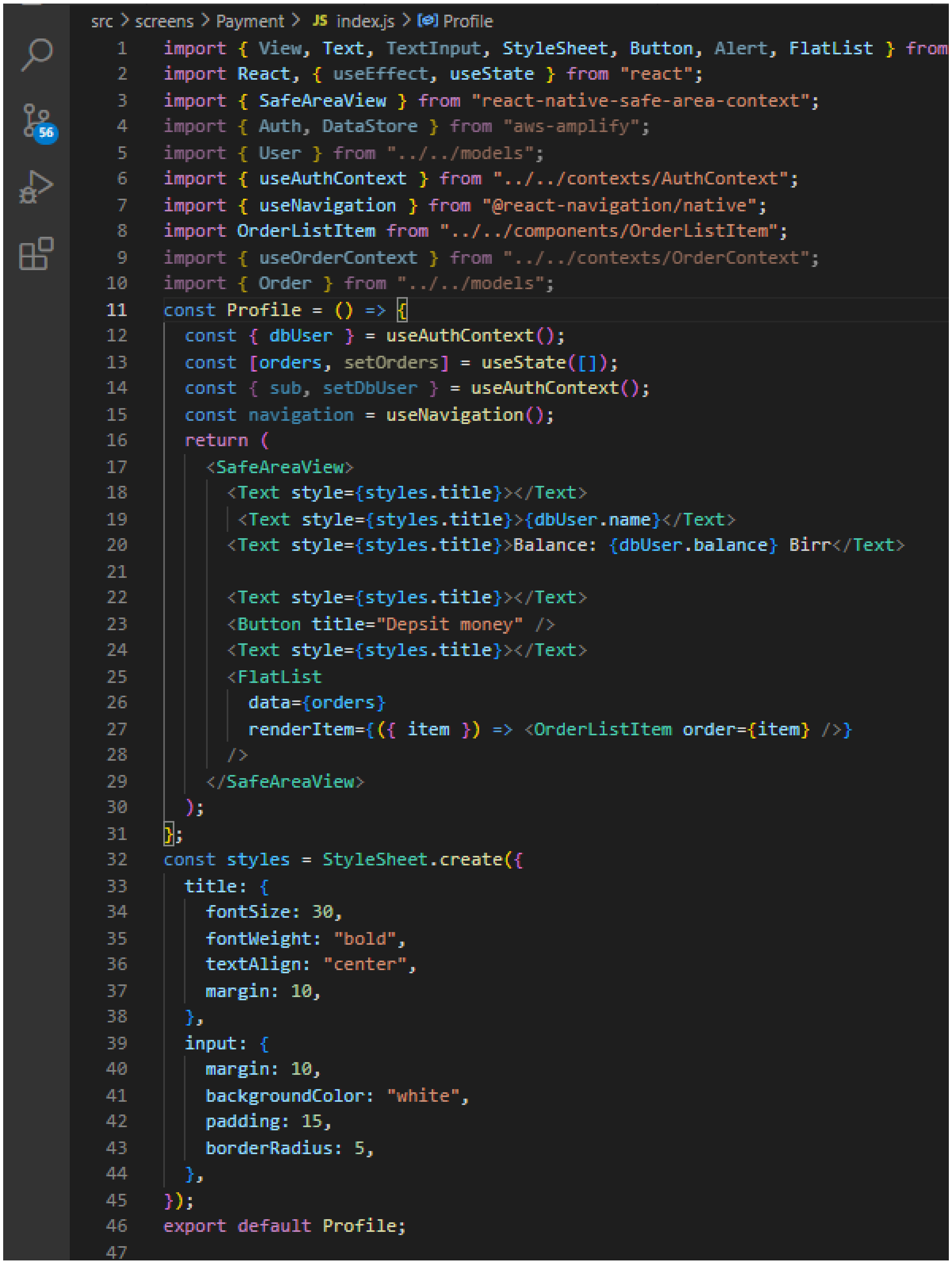


```
const updateCourier = async () => {
  const courier = await DataStore.save(
    Courier.copyOf(dbCourier, (updated) => {
      updated.name = name;
      updated.transportationMode = transportationMode;
      updated.accountNumber= an;
      updated.phoneNumber= pn;
      updated.bankName= bn;

    })
  );
  setDbCourier(courier);
};

const createCourier = async () => {
  try {
    const courier = await DataStore.save(
      new Courier({
        name,
        lat: parseFloat(lat),
        lng: parseFloat(lng),
        sub,
        transportationMode,
        accountNumber: an,
        phoneNumber: pn,
        bankName: bn,
        balance: 0
      })
    );
    setDbCourier(courier);
    navigation.navigate('OrdersScreen', {id: dbCourier.id})
  } catch (e) {
    Alert.alert("Error", e.message);
  }
};
```

Figure 43 Profile Screen



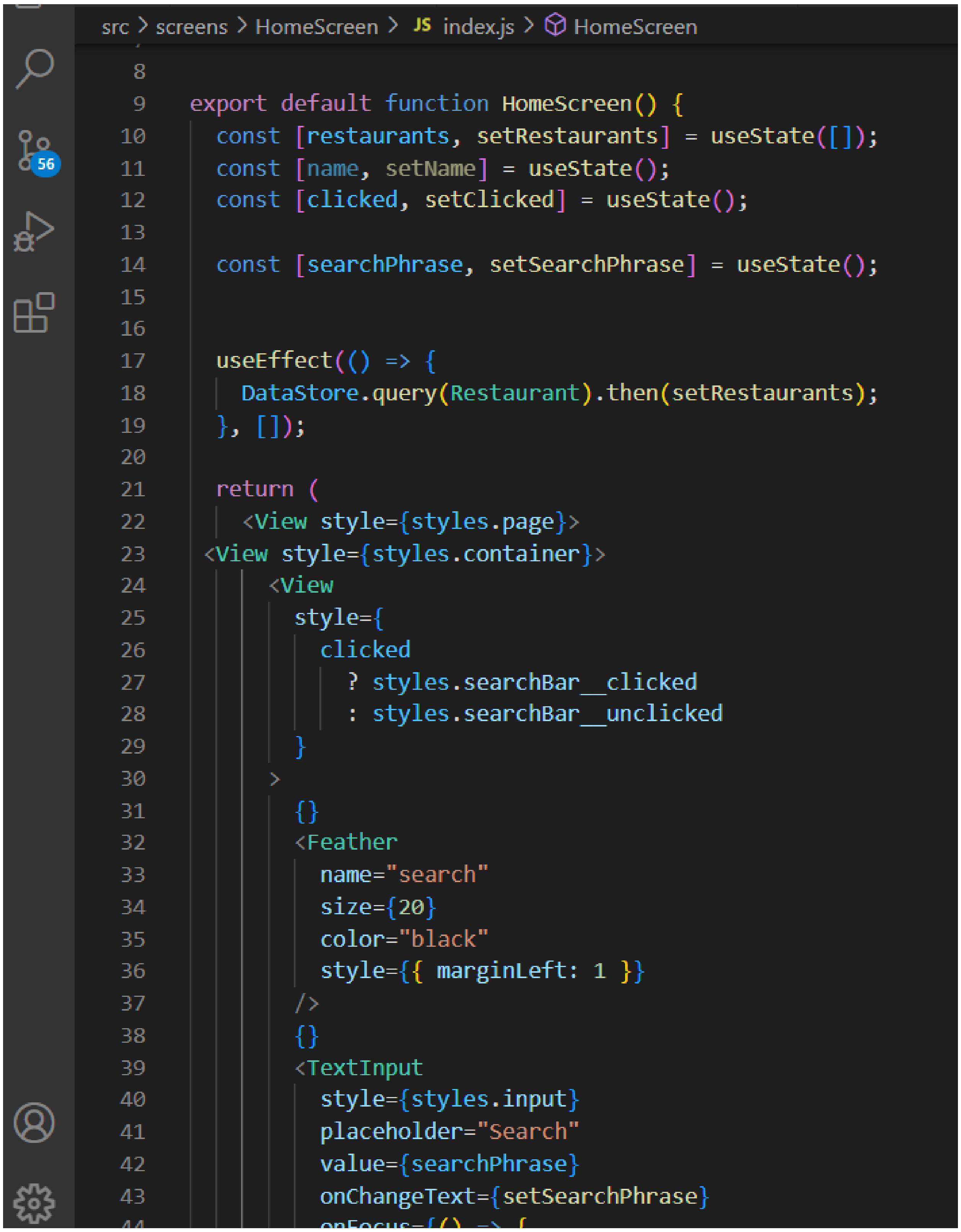
src > screens > Payment > JS index.js > Profile

```
import { View, Text, TextInput, StyleSheet, Button, Alert, FlatList } from
import React, { useEffect, useState } from "react";
import { SafeAreaView } from "react-native-safe-area-context";
import { Auth, DataStore } from "aws-amplify";
import { User } from "../../models";
import { useAuthContext } from "../../contexts/AuthContext";
import { useNavigation } from "@react-navigation/native";
import OrderListItem from "../../components/OrderListItem";
import { useOrderContext } from "../../contexts/OrderContext";
import { Order } from "../../models";
const Profile = () => {
  const { dbUser } = useAuthContext();
  const [orders, setOrders] = useState([]);
  const { sub, setDbUser } = useAuthContext();
  const navigation = useNavigation();
  return (
    <SafeAreaView>
      <Text style={styles.title}></Text>
       <Text style={styles.title}>{dbUser.name}</Text>
      <Text style={styles.title}>Balance: {dbUser.balance} Birr</Text>

      <Text style={styles.title}></Text>
      <Button title="Depsit money" />
      <Text style={styles.title}></Text>
      <FlatList
        data={orders}
        renderItem={({ item }) => <OrderListItem order={item} />}
      />
    </SafeAreaView>
  );
};
const styles = StyleSheet.create({
  title: {
    fontSize: 30,
    fontWeight: "bold",
    textAlign: "center",
    margin: 10,
  },
  input: {
    margin: 10,
    backgroundColor: "white",
    padding: 15,
    borderRadius: 5,
  },
});
export default Profile;

```

Figure 44 Payment Screen



src > screens > HomeScreen > JS index.js > HomeScreen

```

export default function HomeScreen() {
  const [restaurants, setRestaurants] = useState([]);
  const [name, setName] = useState();
  const [clicked, setClicked] = useState();

  const [searchPhrase, setSearchPhrase] = useState();


  useEffect(() => {
    DataStore.query(Restaurant).then(setRestaurants);
  }, []);

  return (
    <View style={styles.page}>
  <View style={styles.container}>
      <View
        style={
          clicked
            ? styles.searchBar__clicked
            : styles.searchBar__unclicked
        }
      >
        {}
        <Feather
          name="search"
          size={20}
          color="black"
          style={{ marginLeft: 1 }}
        />
        {}
        <TextInput
          style={styles.input}
          placeholder="Search"
          value={searchPhrase}
          onChangeText={setSearchPhrase}
```

Figure 45 Home Screen

5.3 Testing

Testing the app is a critical part of the development process, and it involves several different types of testing to ensure that the app meets the necessary quality standards. The testing process typically starts with unit testing, which involves testing each component of the app in isolation to ensure that it is functioning correctly. Integration testing is then performed to test the interaction between different components of the app. Functional testing is performed to ensure that the app is performing its intended functions correctly, such as allowing customers to place orders or delivery drivers to accept orders. User interface testing is performed to ensure that the app's user interface is intuitive and easy to use. Finally, acceptance testing is performed to ensure that the app meets the customer's requirements and is ready for deployment. By following these testing processes, it is ensured that the food delivery app is of high quality, meets the necessary performance standards, and provides a seamless experience to the end-users.

We have decided to manually test the app, and the next step is to create a comprehensive testing plan that outlines the testing scenarios, procedures, and expected results. This plan covers all aspects of the app, including registration, ordering, payment processing, and delivery. The testing plan also includes specific test cases targeted to test certain functionalities. The testing phase continues until all the issues are resolved, and the app meets the necessary quality standards. Finally, we performed user acceptance testing to ensure that the app meets the customer's requirements and is ready for evaluation.

By following a comprehensive testing plan and performing manual testing, the development team can identify and fix any issues in the app before it is launched, ensuring that it provides a seamless and satisfactory experience to the customers.

5.3.1 Features to be tested

For the cloud-based food delivery app, the following items are to be tested in the test plan document:

1. **User registration/authentication**:

User registration and authentication is an essential part of the app, as it ensures that only authorized and legitimate users have access to the app. The registration process should be straightforward, allowing users to quickly sign up with their email address and create a secure password. The authentication process should use multiple layers of security, such as strong encryption algorithms and two-factor authentication, to protect user data and prevent unauthorized access. The app should also be set up to allow users to reset their passwords in the event that they forget them, in order to ensure that accounts are not compromised. In summary, the registration and authentication process should be robust and secure in order to provide the best user experience

2. **Order process**:

The user order process for the app should be quick and easy to use. The app should feature a comprehensive list of food items, along with relevant images and descriptions, to help users quickly and accurately select the items they want to order. Additionally, the app should allow users to select their preferred delivery address, payment method, and any special instructions they may have. The app should also provide options for users to save their favorite orders, so that the process can be streamlined in the future. Once an order is placed, the app should send an email or text message to the user to confirm their order and provide an estimate of when it will be ready. In this way, users can be sure that their orders are handled quickly and accurately.

3. **Delivery tracking**: This includes verifying that customers can accurately track their orders in transit on the app.

4. **Push notifications**: The app should feature notifications to keep users up to date about the status of their orders. The app should provide users with timely notifications when their orders have been received, accepted, and prepared, as well as when the order has been delivered. Additionally, the app should provide users with the option to receive notifications when their orders are nearing delivery or arriving in the restaurant. This will enable users to keep track of their orders and ensure that they are always aware of their orders' status. Furthermore, notifications should also be sent to users in the event that a restaurant is unable to fulfill an order, so that they can make alternate arrangements.

5. **Device compatibility**: This includes verifying that the app is compatible with various devices and operating systems. It should be compatible with a wide range of devices, including both iOS and Android. The app should also be optimized for both mobile and tablet screens, allowing users to access the app on any device.

5.3.2 Item to be tasted

For the app, we have listed out the test items according to the features we have selected to test.

User registration

- Sign-in
- Log in

Order process

- Search dish
- Select dish
- Manage basket
- Place order

Delivery Tracking

- Track order

5.3.3 Approach

The testing approach that was selected was manual testing. This approach involves human testers executing test cases and scenarios, using their domain knowledge and experience to identify issues and defects in the software. The decision to use manual testing was based on factors such as the complexity of the software, the availability of skilled testers, and the need for a flexible and adaptable testing approach. To ensure that the testing effort was comprehensive, the testing team created detailed test cases and scenarios that covered all aspects of the application, including the login process, search functionality, and order placement. The testing team also performed exploratory testing to identify issues that may not have been captured by the test cases. By using manual testing, the testing team was able to ensure that the software met the specified requirements and was of high quality, providing a positive user experience for customers of the app.

5.3.4 Unit Testing

We have developed test cases that test individual components of the system. Test case is a specific test case designed to verify the functionality, performance, or other aspects of an application or system. The test cases include the test case ID, name, description, preconditions, steps to be executed, post-conditions, expected result, Pass and Fail criteria and test environment information. The test case aims to provide clear and detailed instructions to the testers to ensure that the test is executed consistently and accurately. By defining the specific steps and expected results, test case scenarios help to identify defects or errors in the application or system being tested and provide information that can be used to improve the software. A well-designed test case scenario can help ensure that the application or system meets the requirements and user expectations and is of high quality.

Table 21 Login testcase

Test case ID:	TC001
Test case name	Login
Description:	This test case verifies that the login functionality of the food delivery application works as expected
Preconditions:	The application is installed and running on the device The user has a valid account and credentials
Test steps:	1.Launch the application 2.Click on the "Login" button 3.Enter a valid username and password in the login form 4.Click on the "Login" button 5.Verify that the user is successfully logged in and is directed to the home page

Expected result:	The user is successfully logged in and can access the features of the application
Pass criteria:	The user is successfully logged in with valid credentials The application navigates the user to the home page after successful login
Fail criteria:	The user is not able to log in with valid credentials The application does not navigate the user to the home page after successful login
Test data:	Valid username: testuser1 Valid password: testpass1
Post-conditions:	The user is logged in and can access the features of the application
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 22 Sign-up testcase

Test case ID:	TC002
Test case name	Sign up
Description:	This test case verifies that the sign-up functionality of the application works as expected.
Preconditions:	The application is installed and running on the device

Test steps:	1. Launch the application 2. Click on the "Sign Up" button 3. Enter a valid email address, full name, phone number, and password in the sign-up form 4. Click on the "Sign Up" button 5. Verify that the user is successfully registered and is directed to the home page
Expected result:	The user is successfully registered and can access the features of the application
Pass criteria:	The user is successfully registered with valid credentials The application navigates the user to the home page after successful registration
Fail criteria:	The user is not able to register with valid credentials The application does not navigate the user to the home page after successful registration
Test data:	Valid email address: testuser1@example.com Valid full name: Test User Valid phone number: +251 912345678 Valid password: testpass1
Post-conditions:	The user is registered and can access the features of the application
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 23 Search dish testcase

Test case ID:	TC003
Test case name	Search dish
Description:	This test case verifies that the search dish functionality of the application works as expected.
Preconditions:	The application is installed and running on the device The user is logged in to the application
Test steps:	1. Launch the application 2. Enter the name of the dish or the cuisine in the search bar 3. Click on the search icon or press the "Enter" key 4. Verify that the search results page displays the dishes or restaurants that match the search query 5. Click on a dish or restaurant in the search results page 6. Verify that the details of the dish or restaurant are displayed on the screen
Expected result:	The search results page displays the dishes or restaurants that match the search query The details of the selected dish or restaurant are displayed on the screen
Pass criteria:	The search functionality returns results for valid search queries The search results page displays relevant dishes or restaurants that match the search query The user can click on a dish or restaurant in the search results page to access its details

Fail criteria:	The search functionality does not return any results for valid search queries The search results page displays irrelevant or incorrect dishes or restaurants that do not match the search query The user is unable to access the details of the selected dish or restaurant
Test data:	Search query: "Pizza" Restaurant name: "Haile Resort"
Post-conditions:	The user can access the details of the selected dish or restaurant
Test environment:	application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 24 Seach Dish Testcase

Test case ID:	TC004
Test case name	Select dish
Description:	This test case verifies that the user can select a dish and add it to the cart in the application.
Preconditions:	The application is installed and running on the device The user is logged in to the application
Test steps:	1. Launch the application 2. Browse or search for a dish that the user wants to order 3. Click on the dish to view its details

	4. Verify that the dish details page displays the name, description, price, and other relevant details of the dish 5. Select the quantity of the dish that the user wants to order 6. Click on the "Add to basket" button 7. Verify that the dish is added to the cart and the cart total is updated 8. Repeat steps 2-7 for additional dishes if required 9. Click on the cart icon to view the order summary 10. Verify that the order summary page displays the dishes, quantity, and total price of the order
Expected result:	The user can select a dish and add it to the cart The cart total is updated to reflect the price of the selected dish The user can view the order summary and its details
Pass criteria:	The user can view the details of the selected dish The user can select the desired quantity of the dish The dish is added to the cart and the cart total is updated accordingly The user can view the order summary and its details The application does not crash or freeze during the process
Fail criteria:	The user is unable to view the details of the selected dish The user is unable to select the desired quantity of the dish The dish is not added to the cart or the cart total is not updated The user is unable to view the order summary or its details

	The application crashes or freezes during the process
Test data:	Dish name: "Burger" Quantity: 2
Post-conditions:	The user can view the order summary and proceed to checkout
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 25 Manage Basket Testcase

Test case ID:	TC005
Test case name	Manage basket
Description:	This test case verifies that the user can manage the items in the basket (i.e., add, remove, update quantities, and clear) in the application.
Preconditions:	The application is installed and running on the device The user is logged in to the application The user has at least one dish added to the basket
Test steps:	1. Launch the application 2. Click on the cart icon to view the order summary 3. Verify that the order summary page displays the dishes, quantity, and total price of the order 4. Click on the "Edit basket" button to manage the basket items

	5. To add a new item, browse or search for a dish, select the quantity, and click on the "Add to basket" button 6. To remove an item, click on the "Remove" button next to the dish 7. To update the quantity of an item, change the quantity and click on the "Update" button 8. To clear the basket, click on the "Clear basket" button 9. Verify that the basket is updated and the cart total reflects the changes
Expected result:	The user can add a new item to the basket The user can remove an existing item from the basket The user can update the quantity of an existing item in the basket The user can clear the basket The basket is updated and the cart total reflects the changes
Pass criteria:	The user can add new items to the basket The user can remove items from the basket The user can update the quantity of items in the basket The user can clear the basket The basket and cart total are updated as per the changes made by the user The application does not crash or freeze during the process
Fail criteria:	The user is unable to add new items to the basket The user is unable to remove items from the basket

	The user is unable to update the quantity of items in the basket The user is unable to clear the basket The basket or cart total is not updated as per the changes made by the user The application crashes or freezes during the process
Test data:	Dish name: "Burger" Quantity: 1
Post-conditions:	The user can proceed to checkout with the updated basket items
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 26 Place Order Testcase

Test case ID:	TC006
Test case name	Place order
Description:	This test case verifies that the user can place an order in the application.
Preconditions:	The application is installed and running on the device The user is logged in to the application The user has added at least one dish to the basket The user has entered the delivery address
Test steps:	1. Launch the application 2. Click on the cart icon to view the order summary

	3. Verify that the order summary page displays the dishes, quantity, and total price of the order 4. Click on the "Proceed to checkout" button 5. Verify that the checkout page displays the delivery address, payment information, and the order summary 6. Click on the "Place order" button to complete the transaction 7. Verify that the order confirmation page is displayed with the order details (i.e., order ID, delivery address, delivery time, and order summary)
Expected result:	The user can proceed to checkout with the items in the basket The user can view the order summary before placing the order The user can provide the delivery address and payment information The user can place the order by clicking on the "Place order" button The application displays the order confirmation page with the order details
Pass criteria:	The user can proceed to checkout with the items in the basket The user can view the order summary before placing the order The user can provide the delivery address and payment information The user can place the order by clicking on the "Place order" button The application displays the order confirmation page with the order details

	The order confirmation page contains the correct order details (i.e., order ID, delivery address, delivery time, and order summary) The order is processed successfully and the user receives the confirmation email with the order details The application does not crash or freeze during the process
Fail criteria:	The user is unable to proceed to checkout with the items in the basket The user is unable to view the order summary before placing the order The user is unable to provide the delivery address and payment information The user is unable to place the order by clicking on the "Place order" button The application does not display the order confirmation page with the order details The order confirmation page contains incorrect or missing order details The order is not processed successfully and the user does not receive the confirmation email with the order details The application crashes or freezes during the process
Test data:	Dish name: "Burger" Quantity: 1 Delivery address: automatically from google map
Post-conditions:	The order is processed successfully

	The user receives the confirmation email with the order details
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

Table 27 Track Order Testcase

Test case ID:	TC007
Test case name	Track order
Description:	This test case verifies that the user can track the status of their order in the application.
Preconditions:	The application is installed and running on the device The user is logged in to the application The user has placed at least one order
Test steps:	1. Launch the application 2. Click on the "Orders" tab in the bottom navigation bar 3. Verify that the "Orders" page displays a list of the user's past orders 4. Click on the order that the user wants to track 5. Verify that the "Order details" page is displayed with the status of the order (i.e., pending, preparing, on the way, delivered) 6. Click on the "Track order" button to view the live location of the delivery driver 7. Verify that the "Track order" page displays the real-time location of the delivery driver on a map

	8. Click on the "Cancel order" button to cancel the order (optional) 9. Verify that the "Cancel order" confirmation dialog is displayed 10. Click on the "Yes" button to confirm the cancellation (optional)
Expected result:	The user can view the list of past orders on the "Orders" page The user can view the order details, including the current status of the order, on the "Order details" page The user can track the live location of the delivery driver on the "Track order" page The application displays the real-time location of the delivery driver on a map The user can cancel the order by clicking on the "Cancel order" button (optional) The application displays the "Cancel order" confirmation dialog (optional) The user can confirm the cancellation by clicking on the "Yes" button (optional)
Pass criteria:	The user can view the list of past orders on the "Orders" page The user can view the order details, including the current status of the order, on the "Order details" page The user can track the live location of the delivery driver on the "Track order" page The application displays the real-time location of the delivery driver on a map

	The user can cancel the order by clicking on the "Cancel order" button (optional) The application displays the "Cancel order" confirmation dialog (optional) The user can confirm the cancellation by clicking on the "Yes" button (optional)
Fail criteria:	The user is not able to view the list of past orders on the "Orders" page The user is not able to view the order details, including the current status of the order, on the "Order details" page The user is not able to track the live location of the delivery driver on the "Track order" page The application does not display the real-time location of the delivery driver on a map The user is not able to cancel the order by clicking on the "Cancel order" button (optional) The application does not display the "Cancel order" confirmation dialog (optional) The user is not able to confirm the cancellation by clicking on the "Yes" button (optional) The application crashes or freezes during the process
Test data:	Order ID: 1234567890 Delivery address: From Google Maps Status: on the way

	Delivery driver's current location: latitude 37.7749, longitude -122.4194
Post-conditions:	The user is able to view the order details and track the status of the order If the user cancels the order, the application updates the order status to "cancelled" and refunds the payment The application does not crash or freeze during the process
Test environment:	Application version: 1.0 Device: Android or iOS smartphone Operating system: Android 8.0 or above, iOS 11 or above

5.3.5 Integration Testing

Integration testing is a type of software testing that aims to verify the interactions between different components or modules of a software system. The test cases provided below outline some of the key scenarios that are tested for the app. These scenarios cover the user's journey from registration to order tracking and delivery, and include important aspects such as menu browsing and ordering, payment and checkout. By thoroughly testing these scenarios, we can ensure that the app functions as intended, meets user expectations. However, it is important to note that additional testing may be necessary to cover all possible scenarios and edge cases.

User registration:

- Verify that a new user can successfully register for an account using their email address
- Verify that the user is prompted to provide all necessary information during registration, such as name, address, phone number, etc.
- Verify that the user receives a confirmation email after registration.

Menu browsing and ordering:

- Verify that the user can browse the menu of available food items.
- Verify that the user can add items to their cart and adjust quantities as needed.

- Verify that the user can customize their order, such as choosing toppings or selecting options for each item.
- Verify that the user can place their order and receive a confirmation message.

Payment and checkout:

- Verify that the user can pay.
- Verify that the user can enter their payment information and that it is stored securely.
- Verify that the user can review their order and total cost before submitting payment.
- Verify that the user receives a confirmation message after successful payment.

Order tracking and delivery:

- Verify that the user can track the status of their order, such as when it is being prepared, out for delivery, or delivered.
- Verify that the user can see an estimated delivery time and track the driver's location in real-time.
- Verify that the user receives a notification when their order has been delivered.

5.3.6 Acceptance Testing

The acceptance criteria are the specific standards and requirements that must be met for the cloud-based food delivery app to be considered complete and ready for delivery. The following are the acceptance criteria for the project

1. **Functional Requirements:**
    - User Registration and Login: Users should be able to register and log in to the app using their email or mobile number.
    - Food Menu Display: The app should display a list of available food items from different restaurants.
    - Order Placement: Users should be able to place an order for food items from their selected restaurant.
    - Order Tracking: Users should be able to track their order status and receive updates on delivery time.
    - Delivery Management: The app should have a delivery management system for managing deliveries and tracking delivery agents.

2. **Non-Functional Requirements:**
    - Performance: The app should load quickly and respond to user requests in a timely manner.
    - User Experience: The app should have an intuitive and user-friendly interface.
    - Security: The app should have proper security measures in place to protect user data and payments.
    - Scalability: The app should be scalable and able to handle a large number of users and orders.
    - Compatibility: The app should be compatible with both iOS and Android platforms.

3. **Project Management Requirements:**
    - Timeline: The project should be completed within the agreed-upon timeline.
    - Quality: The app should meet the specified quality standards and requirements.

Chapter 6

6. Conclusions and Recommendation

6.1. Conclusion

Throughout the duration of this project, we have acquired an extensive range of competencies related to website development through the use of various development tools and environments. These experiences have required us to undertake comprehensive research, contributing to our well-rounded and diverse understanding of the field. Furthermore, we have honed our collaborative teamwork skills, ensuring efficient project management.

The primary goal of our system is to provide an effective solution to the complexities surrounding food ordering and delivery by leveraging the benefits of a cloud-based platform. Our system incorporates React to develop its websites and React Native to develop its mobile application, enabling seamless access to our services across a variety of devices. Our database will be hosted on Amazon Web Services, ensuring a secure, reliable and scalable platform for our business.

Our main aim of the project in solving this problem by:

- Order management: A cloud-based system can streamline the process of managing orders, including order placement, payment processing, and delivery tracking.
- Efficiency: By automating many of the processes involved in food ordering and delivery, a cloud-based system can help businesses reduce the time and resources needed to manage their operations.
- Cost savings: By eliminating the need for businesses to invest in their own IT infrastructure, a cloud-based system can provide significant cost savings over traditional on premise solutions.
- Cross-platform compatibility: React and React Native enable businesses to build applications that can run on both web and mobile platforms, providing a seamless user experience across devices.
- Flexibility: React and React Native enable businesses to build modular and reusable components, making it easier to develop and maintain complex applications.
- Security: AWS offers multiple layers of security to protect data and applications, including access controls, firewalls, and encryption, ensuring that sensitive information is kept secure.

- Cost-effectiveness: AWS offers pay-as-you-go pricing, meaning businesses only pay for the resources they use, without having to invest in expensive hardware or infrastructure.
- Reliability: AWS provides a highly available and fault-tolerant infrastructure, minimizing the risk of downtime or service disruptions.

6.2 .Recommendation

- We recommend Android version 5.0 and above and iOS version 10 and above for an optimal running environment for the apps
- We recommend devices with 2GB and above to run the apps
- We recommend a stable internet connection for the systems to integrate seamlessly.
- Multi-language support: We recommend future developers add support for multiple languages to make the platform accessible to a wider audience. English—the sole language the system use— is not a native language in Ethiopia and supporting multiple native languages will make the system more accessible to users.
- Loyalty programs: We recommend future developers add a loyalty program feature that rewards customers for repeat orders and incentivizes them to continue ordering from the platform.
- Social media integration: We recommend future developers add features that allows customers to share their orders and experiences on social media platforms, helping to promote the platform and build brand awareness.
- Analytics and reporting: We recommend future developers to incorporate analytics and reporting functionality can provide businesses with valuable insights into customer behavior and preferences, enabling them to make informed decisions about their products and services.
- Integration with third-party services: We recommend future developers integrate the system with third-party services such as payment gateways, delivery services, and inventory management systems can help businesses streamline their operations and improve efficiency.
- None Visual Interface: We recommend future developers add non-visual interfaces that help the visually impaired users navigate through the system to order food.
- In-app chat: We recommend future developers add an in-app chatting mechanism that enables customers to chat and recommend dishes to each other.

References

[1] Grubtech, "How did food delivery start?," [Online]. Available: https://www.grubtech.com/blog/history-of-food-delivery#:~:text=The%20first%20recorded%20food%20delivery,famous%20pizza%20maker%20Rafaele%20Esposito..

[2] emerald, "The Rise of online food delivery during the covid19 pandemic," [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/EJMBE-04-2021-0128/full/html.

[3] Addis Fortune, "Delivery Services Fill Food Industry Gap In Addis," 24 June 2022. [Online]. Available: https://addisfortune.news/delivery-services-fill-food-industry-gap-in-addis/.

[4] I. Ghosh, "These will be the most important cities by 2035," World Economic Forum, 21 October 2019. [Online]. Available: https://www.weforum.org/agenda/2019/10/cities-in-2035/. [Accessed 06 June 2022].

[5] React Native, "React Native. Learn once write anywhere," [Online]. Available: https://reactnative.dev/.

[6] Amzon, "Amazon," [Online]. Available: https://aws.amazon.com/.

[7] P. Gilbert, "Only 20% of Ethiopians are online," Connecting Africa, 15 3 2021. [Online]. Available: https://www.connectingafrica.com/author.asp?section_id=761&doc_id=768066.

[8] V. DUONG, "08 Best Software Development Methodologies," Savvycom, 16 6 2022. [Online]. Available: https://savvycomsoftware.com/blog/best-software-development-methodologies/.

[9] Tech Target, "What is a use case diagram?," [Online]. Available: https://www.techtarget.com/whatis/definition/use-case-diagram.

[10] T. contributer, "Techtarget," may 2019. [Online].

[11] Lucid Chart, "UML Sequence Diagram Tutorial," [Online]. Available: https://www.lucidchart.com/pages/uml-sequence-diagram#:~:text=A%20sequence%20diagram%20is%20a,to%20document%20an%20existing%20process..

[12] Lucid Software, "How to Make a UML Sequence Diagram," YouTube, 27 August 2018. [Online]. Available: https://youtu.be/pCK6prSq8aw. [Accessed 24 June 2022].

[13] smartdraw, "Tips for Effective UML Diagrams," [Online]. Available: https://www.smartdraw.com/uml-diagram/uml-diagram-tips.htm.