Kingdom of Saudi Arabia
Ministry of Education
King Abdulaziz University
Faculty of Computing and Information Technology
Department of Information Systems

المملكة العربية السعودية
وزارة التعليم
جامعة الملك عبد العزيز
كلية الحاسبات وتقنية المعلومات
قسم نظم المعلومات

Table Time

Turki Abdu Hmq ID :1845928
Salman Said Alshehri ID :1937400

Supervisor

Dr. Ayman Yafoz

Abstract

The primary purpose of this project is to solve problems with restaurant reservation application. We created a survey on restaurant reservation applications with 211 participants, and they expressed a strong interest in the restaurant reservation applications. We tried some restaurant reservation applications and discovered that the majority of them have issues such as the need to have tables only for reservation purposes in order for the reservation application to work without the staff manually assigning customers to their tables; these tables would remain unused until a customer requested a reservation, reducing the number of customers in the restaurant. To address this issue, we utilized NFC technology to reduce the amount of input required by staff to update the status of tables in the database by simply placing their phones near the NFC tag, triggering the application to update the status of these tables in the database. When a customer arrives without a reservation and wishes to sit there, they simply update the table state to occupied, and when they depart, they update the table status to available for reservation. This will address one of the most prevalent issues with restaurant reservation application.

Contents

List of Figures

List of Tables

Chapter 1

Project Outlines

1.1 Introduction

One of the biggest causes of customer dissatisfaction is the waiting time for food preparation, and that's why restaurants have a reservation system. The reservation system does solve the waiting time problem, but it is challenging to organize reservations in a traditional way, even reservation applications have issues such as not having a feature to allow the customer to choose a table he desires which leaves the customer dissatisfied and makes the restaurant lose customers. Our goal is to develop a reservation application that will solve the issues of other reservation applications with the use of Internet of Things technology. Near Field Communication (NFC) tags are devices that will make reservations more automated and easier to use and will be helpful with making a feature which is the ability to choose any table the customers want in the restaurant for dining. Our goal is to make reservations easier for both the customers and the restaurants while improving the reservation experience, which will lead to customer satisfaction.

1.2 Problem Statement

Restaurants reservation services requires lots if information input which takes so much time from the restaurant's employees and makes reservations time consuming. Many restaurant reservation applications do not offer a service to prepare the meal before customer's arrival which causes long wait time for the meals to be prepared. Most if not all reservation applications do not give customers the ability to choose booking for specific table.

1.3 Clear Statement of Objectives

- Develop an application for restaurant reservation management.
- Enhance the customer experience by providing detailed restaurant information.
- Enable customers to choose their preferred seating location.
- Facilitate communication between customers and the restaurant.
- Collect customer feedback to improve restaurant performance.

1.4 Methodology

Our decision was to use the waterfall model for our project because it allows us to thoroughly define and plan each stage of development before moving on to the next, which ensures that we have a clear understanding of the project's requirements and design before implementation begins.

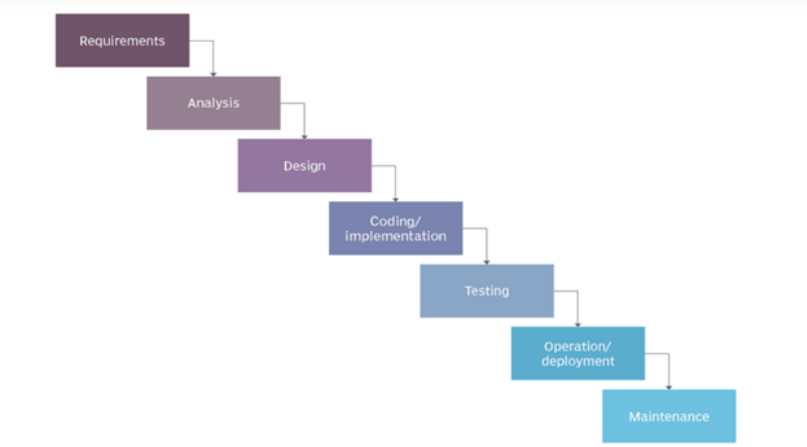


Figure 1.1: Waterfall Model Methodology[1].

1. **requirements:**

   In this initial phase, we performed a survey of potential customers, which was published via social media apps such as WhatsApp, Telegram, and Twitter. To gather feedback from restaurant owners on our project, we performed another survey with them during meetings. The survey findings helped us assess the features, design, and goals of Table Time application.

2. **Analysis:**

   We created extensive diagrams, such as use case diagrams, class diagrams, sequence diagrams, activity diagrams, entity relationship diagrams, and relational schema diagrams, that demonstrate the app's structure based on the requirements acquired during the "Requirements Definition" phase.

3. **Design:**

   After analyzing requirements, you will apply Shneiderman's 8 Golden Rules to build the user interface, application architecture, and database structures.

4. **Deployment:**

   We developed the application using Flutter and added PHP for website support. In addition, we used XAMPP to construct a local database and connect it to both the program and the website.

5. **Test:**

   Following implementation, we tested the application using both black-box and white-box approaches, including black-box acceptance testing and white-box unit testing.

6. **Deployment:**

   After successful quality testing, the system will be deployed and activated for actual use.

7. **Maintenance:**

   After the system is deployed, it is necessary to maintain, develop, and address any issues that arise during actual usage.

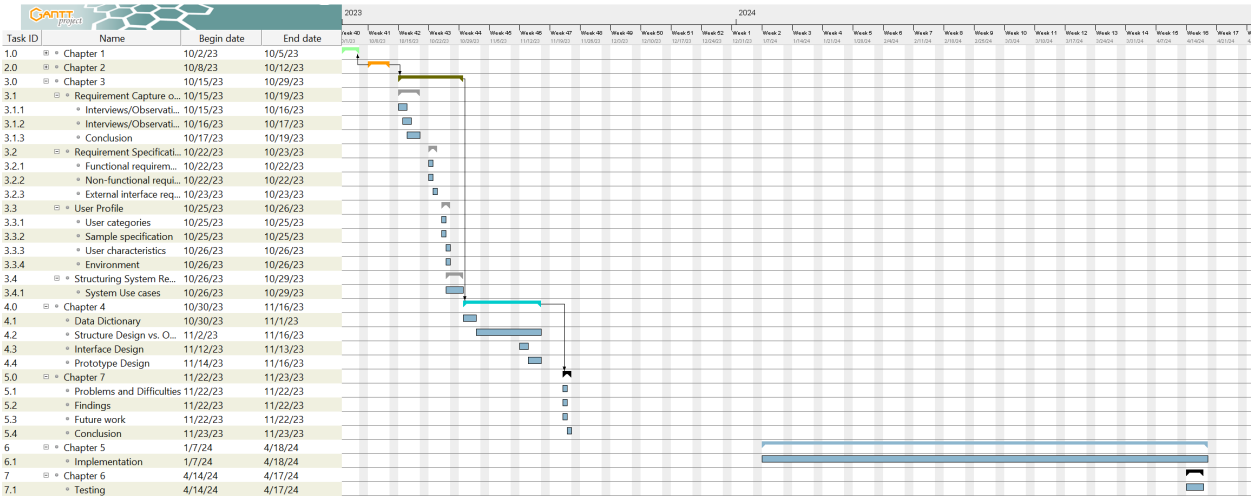
1.5 Project Plan

1.5.1 Clear Task Specification

Task ID	Task Name	Assigned To	Duration	Start Date	End Date	Dependency	status
0	Chapter 1 " **Project Outlines**"		4 Days	10/2/2023	8/5/2023	-	close
1.1	Introduction	Salman/Turki	2 Days	10/2/2023	10/3/2023	0	close
1.2	Problem Statement	Turki	2 Days	10/2/2023	10/3/2023	0	close
1.3	Clear Statement of Objectives	Salman/Turki	1 Day	10/4/2023	10/4/2023	0	close
1.4	Methodology	Salman/Turki	1 Day	10/4/2023	10/4/2023	0	close
1.5	Project Plan	Salman	1 Day	10/5/2023	10/5/2023	0	close
2	Chapter 2 "**Literature Review**"		5 days	10/8/2023	10/12/2023		close
2.1	Background and Overview Of Related Work	Salman/Turki	1 Day	10/8/2023	10/8/2023	-	close
2.2	Analysis of Related Work	Salman/Turki	2 Days	10/8/2023	10/10/2023	2.1	close
2.3	Overview of implementation Tools	Turki	2 Days	10/11/2023	10/12/2023	1.3 , 1.4	close
3	Chapter 3 "**Analysis**"		11 Days	10/15/2023	10/29/2023	-	close
3.1	Requirement Capture	Salman/Turki	5 Days	10/15/2023	10/19/2023		close
3.1.1	Interview/Observation description	Salman/Turki	2 Days	10/15/2023	10/16/2023	3.1	close
3.1.2	Interviews/Observation analysis	Salman/Turki	2 Days	10/16/2023	10/17/2023	3.1	close
3.1.3	Conclusion	Salman/Turki	3 Days	10/17/2023	10/19/2023		close
3.2	Requirement Specification	Salman/Turki	2 Days	10/22/2023	10/23/2023		close
3.2.1	Functional requirement	Salman/Turki	1 Day	10/22/2023	10/22/2023		close
3.2.2	Non-functional requirements	Salman/Turki	1 Day	10/22/2023	10/22/2023		close
3.2.3	External interface requirements	Salman/Turki	1 Day	10/23/2023	10/23/2023		close
3.3	User Profile	Salman/Turki	2 Days	10/24/2023	10/26/2023		close
3.3.1	User categories	Salman/Turki	1 Day	10/25/2023	10/25/2023		close
3.3.2	Sample specification	Salman/Turki	1 Day	10/25/2023	10/25/2023	3.2.3	close
3.3.3	User characteristics	Salman/Turki	1 Day	10/26/2023	10/26/2023	3.2.3	close
3.3.4	Environment	Salman/Turki	1 Day	10/26/2023	10/26/2023		close
3.4	Structuring System Requirements	Salman/Turki	4 Days	10/26/2023	10/29/2023		close
3.4.1	System Use cases	Salman/Turki	4 Days	10/26/2023	10/29/2023	3.2.1	close
4	Chapter 4 "**System Design**"		14 Days	10/30/2023	11/16/2023	-	close
4.1	Data Dictionary	Salman/Turki	3 Days	10/30/2023	11/1/2023		close
4.2	Structure Design vs. Object Oriented design	Salman/Turki	11 Days	11/2/2023	11/16/2023	4.1	close
4.3	Interface Design	Salman/Turki	1 Day	11/12/2023	11/13/2023		close
4.4	Prototype Design	Salman/Turki	1 Day	11/14/2023	11/16/2023	4.3	close
5	Chapter 5 "**Conclusion**"		2 Days	11/22/2023	11/23/2023	-	close
5.1	Problems and Difficulties	Salman/Turki	3 Days	11/22/2023	11/22/2023		close
5.2	Findings	Salman/Turki	1 Day	11/22/2023	11/22/2023		close
5.3	Future work	Salman/Turki	1 Day	11/22/2023	11/22/2023		close
5.4	Conclusion	Salman/Turki	1 Day	11/23/2023	11/23/2023		close

Figure 1.2: Project Plan

1.5.2 Task Duration - (Gantt Chart)



Task ID	Name	Begin date	End date
1.0	Chapter 1	10/2/23	10/5/23
2.0	Chapter 2	10/8/23	10/12/23
3.0	Chapter 3	10/15/23	10/29/23
3.1	Requirement Capture o...	10/15/23	10/19/23
3.1.1	Interviews/Observati...	10/15/23	10/16/23
3.1.2	Interviews/Observati...	10/16/23	10/17/23
3.1.3	Conclusion	10/17/23	10/19/23
3.2	Requirement Specificati...	10/22/23	10/23/23
3.2.1	Functional requirem...	10/22/23	10/22/23
3.2.2	Non-functional requi...	10/22/23	10/22/23
3.2.3	External interface req...	10/23/23	10/23/23
3.3	User Profile	10/25/23	10/26/23
3.3.1	User categories	10/25/23	10/25/23
3.3.2	Sample specification	10/25/23	10/25/23
3.3.3	User characteristics	10/26/23	10/26/23
3.3.4	Environment	10/26/23	10/26/23
3.4	Structuring System Re...	10/26/23	10/29/23
3.4.1	System Use cases	10/26/23	10/29/23
4.0	Chapter 4	10/30/23	11/16/23
4.1	Data Dictionary	10/30/23	11/1/23
4.2	Structure Design vs. O...	11/2/23	11/16/23
4.3	Interface Design	11/12/23	11/13/23
4.4	Prototype Design	11/14/23	11/16/23
5.0	Chapter 7	11/22/23	11/23/23
5.1	Problems and Difficulties	11/22/23	11/22/23
5.2	Findings	11/22/23	11/22/23
5.3	Future work	11/22/23	11/22/23
5.4	Conclusion	11/23/23	11/23/23
6	Chapter 5	1/7/24	4/18/24
6.1	Implementation	1/7/24	4/18/24
7	Chapter 6	4/14/24	4/17/24
7.1	Testing	4/14/24	4/17/24

Figure 1.3: Gantt Chart

Chapter 2

Literature Review

In Chapter 2, we will talk about related work and compare it to our potential project. We will also discuss the tools that we may use to construct our project.

2.1 Background and Overview of Related Work

We conducted a comprehensive review of projects with similarities to our proposed project. During our evaluation of these existing projects, it became evident that they lacked many key features compared to our potential project. One of the initial steps in achieving success is gaining a deep understanding of the project's environment, learning from it, analyzing the competition, and evaluating their work. To compare our potential project with other existing applications, we adopted a method that involved downloading these apps and conducting practical tests, including reservation processes, understanding terms and conditions, and carrying out live user experiences.

2.1.1 Skipro

2.1.1.1 Introduction

Skipro is a restaurant reservation application that lets you make appointments on waiting lists at various restaurants mentioned on the application. Customers can choose the best location and time for them using Skipro, avoiding the need to wait for minutes or even hours. The software provides an easy experience, allowing users to enter restaurants without having to wait[2].

2.1.1.2 Advantages of Skipro

- Skipro features an easy-to-use interface that is well-organized, making it user-friendly and accessible.
- The app provides suggestions for nearby places based on the user's location, helping them discover suitable options.
- Users can easily share the location of the chosen restaurant with others, enhancing social sharing and coordination.
- After selecting a restaurant, the app allows users to access information about its various branches, which is a helpful feature for those who prefer specific locations.
- Offering information about business hours helps users plan their visits more effectively.
- The ability to add favorite restaurants to a list for quick access enhances the user experience.

2.1.1.3 Disadvantages of Skipro

- The application has no restaurant menu pages, which negatively impacts the customer experience.
- If the restaurant is full, customers are unable to make reservations.
- Customers cannot choose restaurants based on ratings because the review and comment section is disabled.

2.1.1.4 Application Interfaces Skipro

Figure 2.1: Home Page

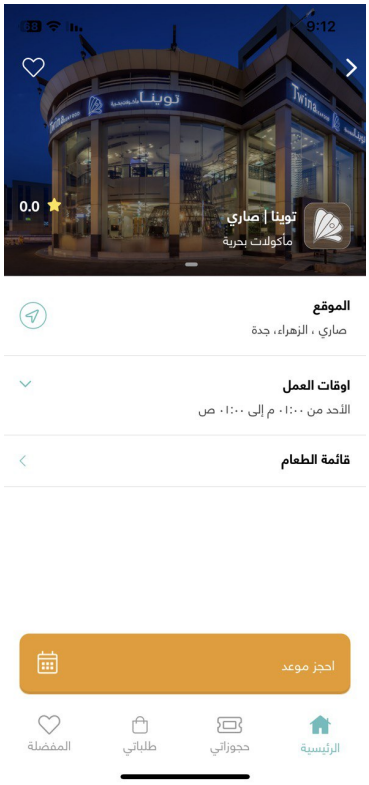


Figure 2.2: Select restaurant

Figure 2.3: Make reservation

2.1.2 ReQueue

2.1.2.1 Introduction

ReQueue is a Kuwaiti firm that provides electronic solutions and has developed an application to organize congested restaurants and cafes to improve the customer experience. By allowing the customer to enter the application and then for waiting lists at the selected restaurant, and by specifying the number of people and providing a temporary time to wait with the availability of contact information and geographical location for the restaurant[3].

2.1.2.2 Advantages of ReQueue

- The application has a simple layout that shows available restaurants, making it easier for users to choose.
- The presence of subsidiary boxes for each restaurant and the capacity for user ratings allow customers to offer personal comments and appraisals of their experiences.
- Customers benefit from the "ReQueueTickets" service, which allows them to change the number of seats or ensure a place in the queue even when the restaurant is filled. This enhances the flexibility of the customer experience.
- The flexibility to customize searches based on chosen cities, ratings, or congestion levels assists consumers in quickly identifying places that match their tastes.
- The availability of data revealing the waitlist number and the time remaining in the queue enables customers to estimate their waiting times more effectively.

2.1.2.3 Disadvantages of ReQueue

- The presence of a call icon without an accompanying phone number can detract from the user's experience. It is imperative to provide customers with a means of contacting the restaurant effortlessly.
- The absence of menus on the select restaurant's page can pose a challenge for customers in discerning the available food offerings.
- The issue of delayed updates to queue statuses can lead to inaccuracies in customers' waiting time estimates, thus necessitating more precise and timely updates.

2.1.2.4 Application Interfaces ReQueue

Figure 2.4: Home Page

Figure 2.5: Select restaurant

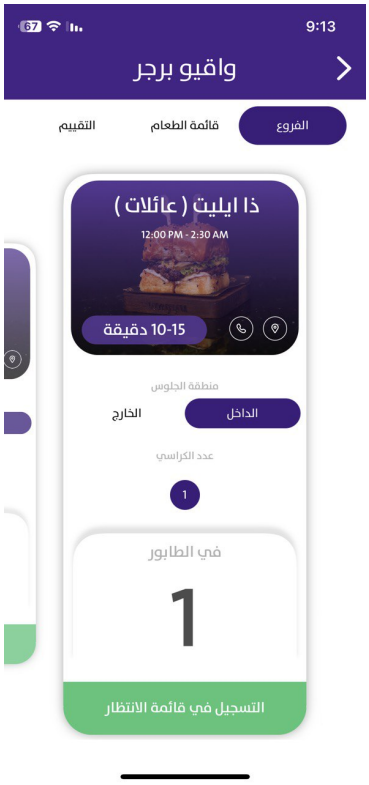


Figure 2.6: Make reservation

2.1.3 Wddk

2.1.3.1 Introduction

Wddk is a restaurant and coffee shop reservation application designed to secure tables in busy eateries. Wddk is an app that makes bookings for you before you get to the restaurant, ensuring that you receive a table when you want it. It offers a variety of reservation features such as booking a table, revealing the location of the restaurant, sending SMS notifications, and reviewing the reservation order[4].

2.1.3.2 Advantages of Wddk

- Viewing the restaurant's meal menu.
- SMS message for reservation confirmation when the customer is due to arrive at the restaurant.
- The customer is not required to sign in because he can access the site as a guest.
- Reservation reminder SMS notification.

2.1.3.3 Disadvantages of Wddk

- Poor navigation for restaurant information and menus because the information is ordered vertically and you have to scroll a lot to see the available drinks.
- There is a 60-minute time limit for sitting at the table, which is insufficient when you have to wait for the restaurant to cook the food.
- When using a guest account to book a table, you must sign in to cancel the reservation, which negates the purpose of utilizing a guest account.
- There is no possibility to select or request that the restaurant prepare the dish that the consumer desires.
- There is no option for the customer to select a specific table.

2.1.3.4 Application Interfaces Wddk

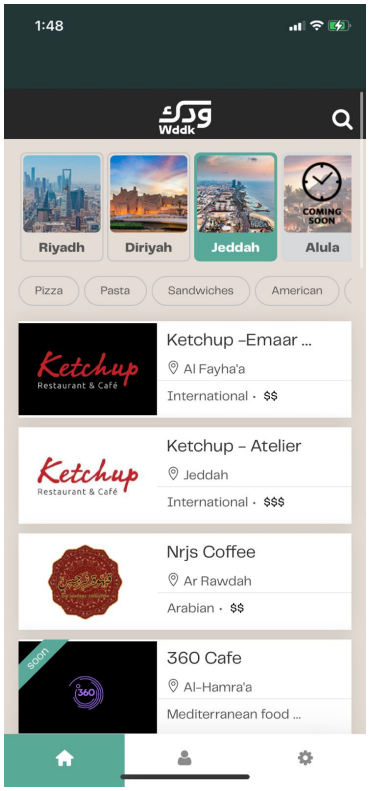


Figure 2.7: Home Page

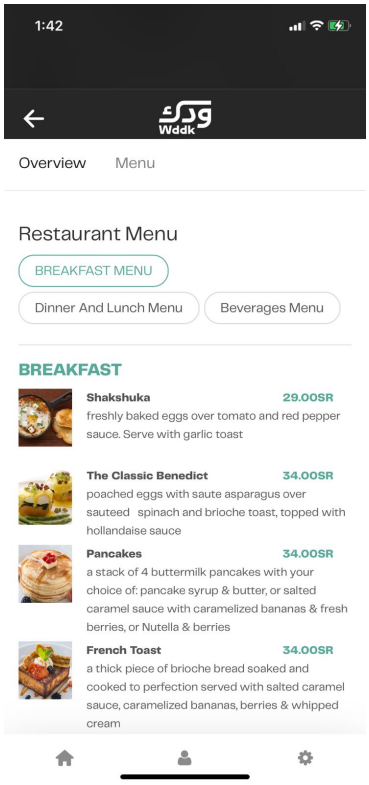


Figure 2.8: Select restaurant

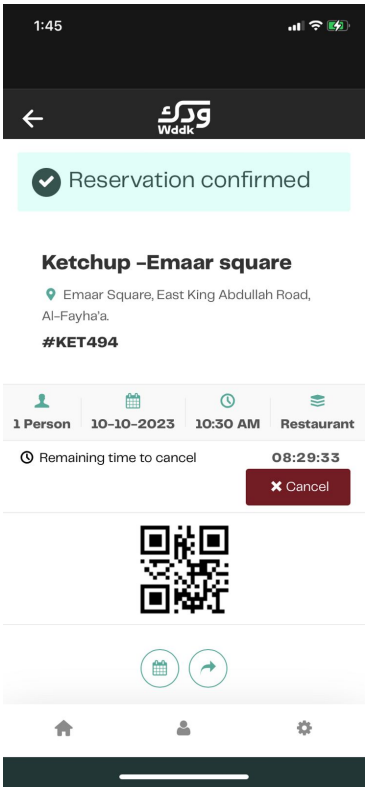


Figure 2.9: Make reservation

2.1.4 MyTable

2.1.4.1 Introduction

MyTable is an application that provides a variety of services. It serves as a hub for restaurants and customers, allowing restaurants to post pictures to advertise their businesses while customers interact with the posts by liking and commenting. Restaurants can also provide services such as reservations and delivery[5].

2.1.4.2 Advantages of MyTable

- Having a hub for restaurants to post their food and so advertise their restaurants.
- Navigating different services is simple and straightforward.
- Every restaurant displays its menu.
- The customer can filter the restaurant list based on the type of service he requires.
- In the application, customers can remark on restaurant accounts and review their services.
- Customers can read a variety of information about the restaurants, including the number of bookings made today, working hours, the minimum fee for reservations, the business's website, and the hours when children are permitted inside the restaurant.
- MyTable features a loyalty system in which customers earn points for making reservations or ordering delivery.

2.1.4.3 Disadvantages of MyTable

- The restaurant does not have the option of preparing meals for reservations before the customer arrives.
- There is no option for the customer to select a specific table.

2.1.4.4 Application Interfaces MyTable

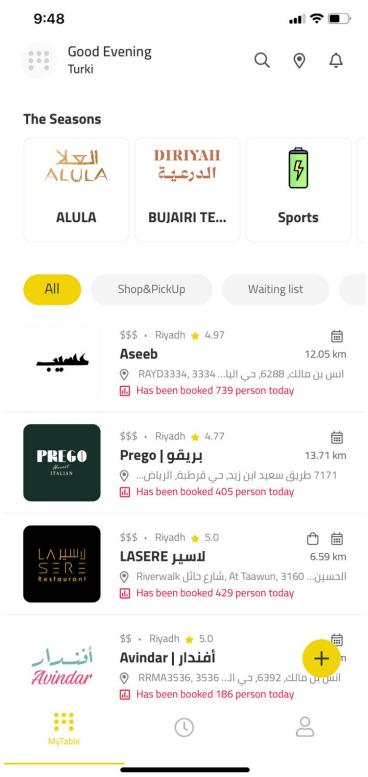


Figure 2.10: Home Page

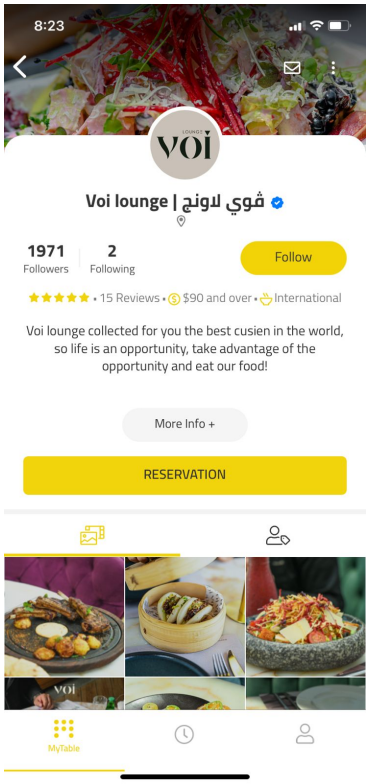


Figure 2.11: Select restaurant

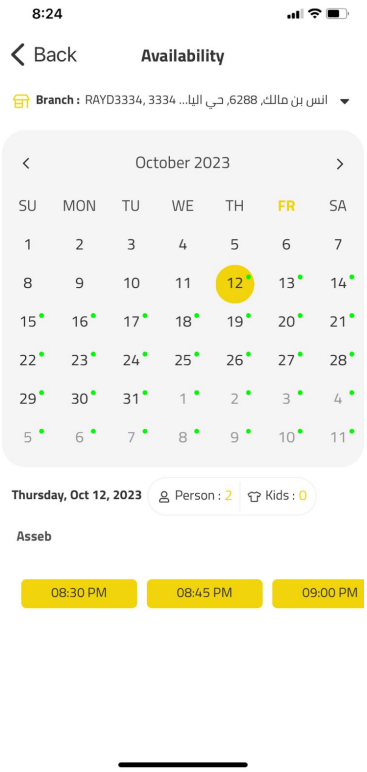


Figure 2.12: Make reservation

2.2 Analysis of Related Work

Comparison Table Between Competing Apps and Our App:

Table 2.1: Feature Comparison

Features/Points					
	Skipro	ReQueue	Wddk	MyTable	Our App
User-Friendly Interface	✓	✓	×	✓	✓
Providing Restaurant Menus	×	✓	✓	✓	✓
Reservation When Restaurant Is Full	✓	✓	✓	✓	✓
Providing Information About Restaurant	✓	×	✓	✓	✓
User Comments and Ratings	×	✓	✓	✓	✓
Information Organization by Categories	×	✓	✓	✓	✓
Providing Working Hours Information	✓	✓	✓	✓	✓
Feature to choose a specific table	×	×	×	×	✓
Sending Customer Notifications and Reminders	×	✓	✓	✓	✓
Feature to choose meals and order them	×	×	×	×	✓
Show the user which table is available	×	×	×	×	✓

2.3 Added Value for Time Table

Following an analysis of related work, Table Time introduces further features that enhance customer and restaurant owner experiences, these added values are:

1. No restaurants in Saudi Arabia uses NFC technology to the best of our knowledge.
2. All tables are accessible to both reservation holders and walk-in clients.
3. Customers can select their table place in the restaurant.
4. Dashboard website for restaurant owners to manage their restaurants.

2.4 Overview of Implementation Tools

Development tools:

	Flutter is an open-source framework by Google, it is well known framework for developing several platforms including smart phones. We choose Flutter because it is a popular framework and there are many tutorials on using it.
	Dart is a programming language which is used for developing multiple platforms such as iOS, Android, websites and desktop. We chose Dart programming language because it is easy to use, there are lots of tutorials on Dart and lastly it can run on both iOS and Android.
	XAMPP is an open-source software package that provides a local web server environment for testing and development, utilizing MySQL to provide a local database.
	MySQL Database is a client/server system. We chose MySQL because we have previous experience using it and that experience would make it easy to use.
	PHP is an open-source server-side scripting language used to create a variety of software applications, including interactive and static websites. We utilized it to build a dashboard website for our application.

Designing and prototyping tools:

	Figma is the leading collaborative design tool, it is used for collaborative designing which can be used for designing User Interface. We choose Figma because it has good features which we need such as real time collaboration, ease of file sharing, lastly the designs can be translated to codes and there are many more features.

Additional used technology:

	NFC stands for near-field communication; it is a type of Internet of Things technology that allows for short-range data transmission. We chose to combine this technology with NFC tags since it is an economical way of achieving project objectives.

Chapter 3

Analysis

3.1 Requirement Capture

3.1.1 Interview/Observation Description for Customers

We created a survey that was successfully completed and posted on WhatsApp and Snapchat for the purpose of collecting data and participant viewpoints that helped us evaluate consumers' experiences with restaurant reservation applications.
The survey included an introduction to the concept, an application that aims to simplify the process of booking tables by utilizing near-field communication technology (NFC).

The survey consists of 8 primary questions, and the answers are categorized by question type.

1. Have you ever used applications for booking tables at restaurants?
2. Do you support the idea of creating a dedicated application for table reservations at restaurants?
3. Do you support the idea of using application services without the need to log in?
4. How do you feel about the service of selecting a table position in restaurants?
5. Do you support providing a menu with information such as (price and calories)?
6. Do you support the idea of having a food preparation service before arriving at the restaurant?
7. Do ratings and reviews affect your choice of restaurants?
8. What features do you consider the most important when using a restaurant reservation application in your opinion?

3.1.2 Interview/Observation Analysis for Customers

The survey was completed by using Google Forms, enabling us to collect and evaluate responses easily. This survey has 211 participants who were sharing their opinions and feedback on our project and its importance to them. This deep connection displays a full understanding of and interest in the survey's topic matter. We assessed the results and displayed response ratios in a methodical manner. This will enable us to better identify issues and challenges that may develop while booking tables, enabling us to provide a better customer experience. We developed our applications based on participant feedback, ensuring that their needs and expectations were met, resulting in a more user-friendly and unique experience that will improve the user experience and speed up the table booking process.

These are the results of our survey:

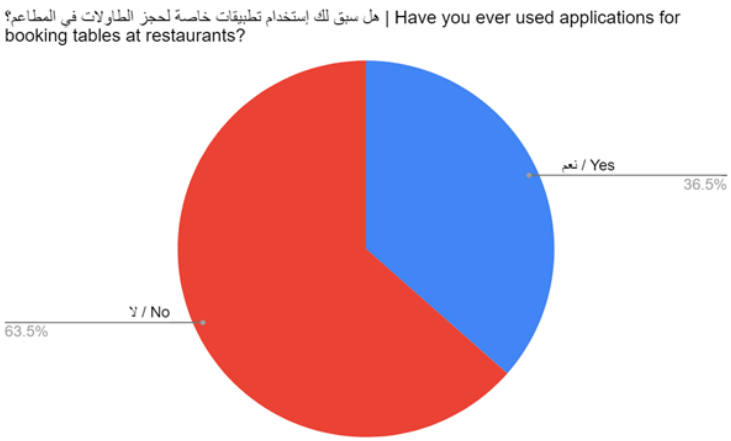


Figure 3.1: Question 1 in the survey

Figure 3.2: The question complements the first question in this survey

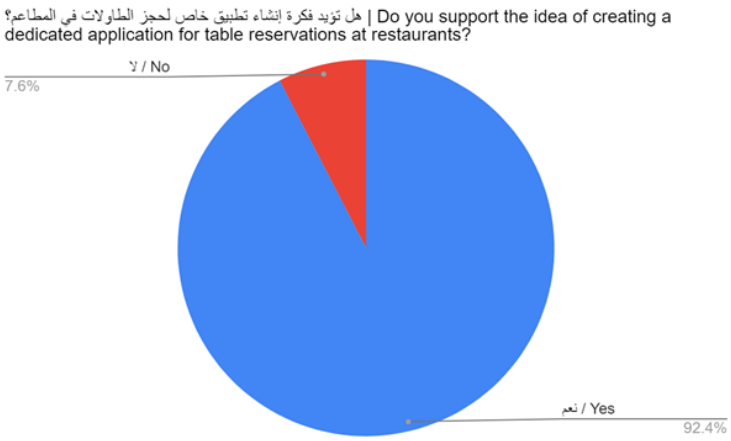


Figure 3.3: Question 2 in the survey

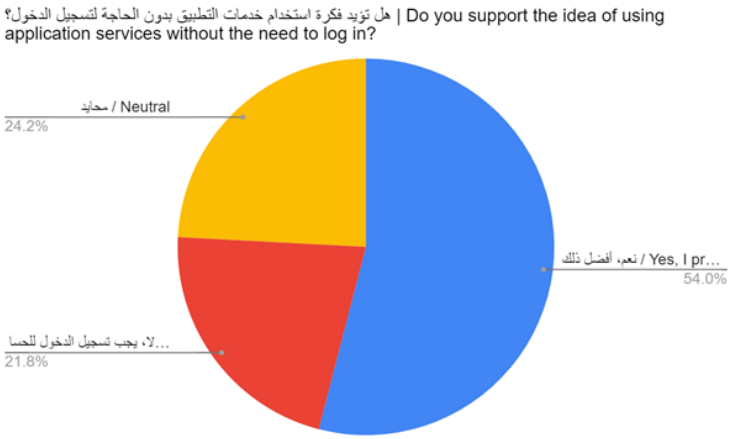


Figure 3.4: Question 3 in the survey

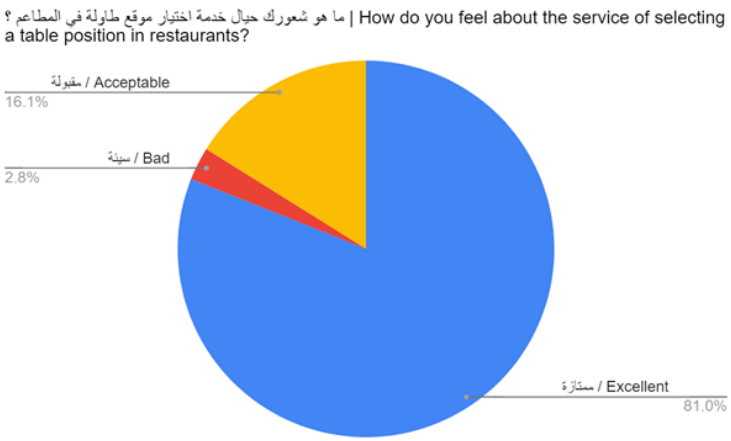


Figure 3.5: Question 4 in the survey

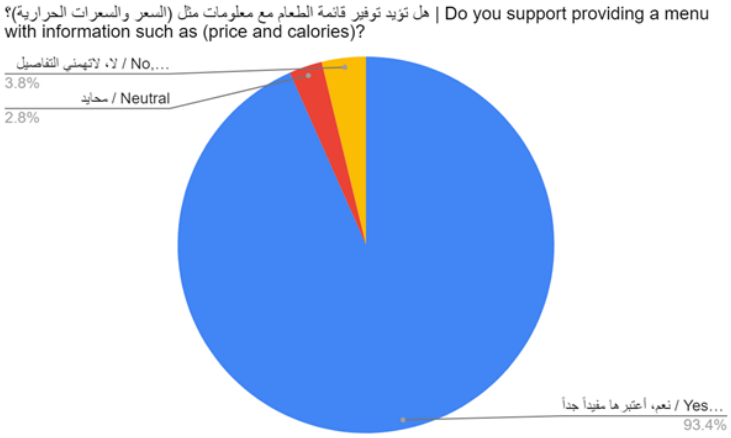


Figure 3.6: Question 5 in the survey

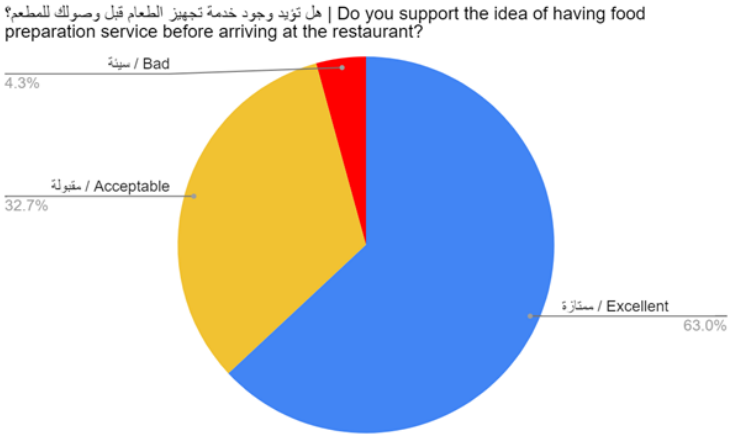


Figure 3.7: Question 6 in the survey

Figure 3.8: Question 7 in the survey

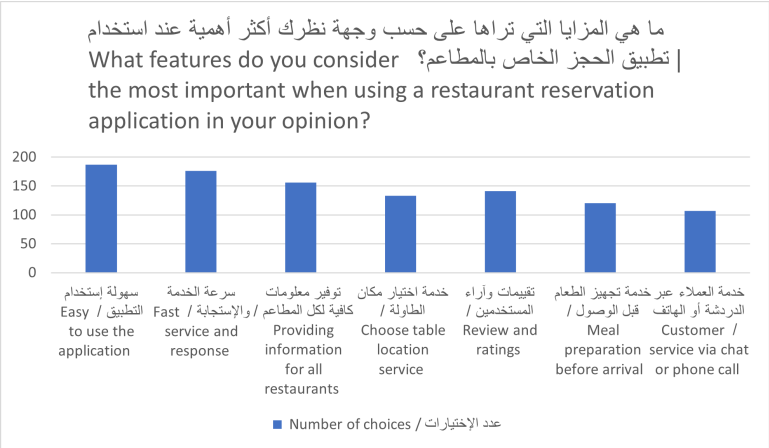


Figure 3.9: Question 8 in the survey

3.1.3 Interview/Observation Description for Owners

We planned and successfully completed a survey, in addition to our physical presence, to explain the benefits to the participants. The following questions are intended to elicit feedback from restaurant owners and managers on their experience with restaurant reservation applications and the potential use of Near Field Communication (NFC) technology to facilitate table reservation processes in their restaurants, as well as to assess the application's design and their interest in implementing it in their restaurants.

3.1.4 Interview/Observation Analysis for Owners

1. **Do you have prior experience using restaurant reservation applications?**
   One restaurant owner has past experience utilizing restaurant reservation applications, whereas the other does not.

2. **What challenges have you encountered throughout the table reservation process at your restaurant?**
   The results are:

   - Managing orders at busy times.
   - Providing enough tables to meet requests during peak hours.

- Customers not arriving at their table within their reservation time.

3. **Do you think NFC technology can improve your restaurant's table reservation process?**
   A restaurant owner believes the application has the potential to improve the restaurant reservation process, while another says it will not.

4. **Do you have concerns about adopting NFC technology for reservations?**
   One restaurant owner has no concerns, whilst the other owner is somewhat concerned.

5. **What are your concerns about adopting NFC technology for reservations?**
   The results are:

   - The customer's lack of concern for their appointment and its impact on other customers.
   - The integration between the booking application and the cash register program.

6. **Do you like our application's design?**
   One restaurant owner liked the application's design, while the other had mixed opinions.

7. **Are you interested in adopting our application in your restaurant?**
   One restaurant owner is interested in using our application in his restaurant, but the other is not.

3.1.5 Conclusion

According to the survey results, potential customers indicate that there is a great demand for reservation applications. The survey indicates that potential customers are interested in a reservation application that allows them to select their table position.

Meanwhile, restaurant owners are open to the concept of incorporating Table Time into their reservation system, as long as their concerns are addressed.

Our objective is to improve both the customer and restaurant owner experience by creating a user-friendly application that meets their expectations.

3.2 Requirement Specification

"A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development"[6]. The users will have different interfaces with different features, we will list all the functional and non-functional requirements.

3.2.1 Functional Requirements

"Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks"[7]. We have three different actors in our project: restaurant owners, restaurant employees, and customers.

- **Register**: Required to gain access to certain services (Customers, Owners).
- **Login**: To use additional services, the user must first log in (Customers, Employees, Owners).
- **Search**: The user can use the search function to find a specific restaurant (Customers).
- **Select Restaurant**: The user will select a restaurant of his choice (Customers).
- **Booking table**: make a reservation and select a specific table (Customers).
- **Review**: Restaurants can be rated and commented on as well (Customers).
- **Check restaurant's information**: A feature that allows you to check all of the restaurant's information (Customers).
- **Approve reservation**: After placing the booking order, the user must approve the reservation (Customers).
- **Order meal preparation before arrival**: A service that allows customers to order meal preparation before arrival (Customers).

- **Restaurant location**: A function that displays the restaurant's location on Google Maps (Customers).
- **Payment system**: A service to pay for meal orders (Customers).
- **Add tables in the restaurant**: Add additional tables in the restaurant (Employees).
- **Assign NFC tags to their tables**: Assign NFC tags to each table available in the restaurant (Employees).
- **Managing reservation orders**: Remove customers from reservation waiting list (Employees).
- **Update table state**: Using NFC technology, change the status of tables from free to occupied to free again (Employees).
- **Check and interact with reviews**: To provide a better experience for the consumer, the employee can read and respond to reviews (Employees, Owners).
- **Menu management**: Feature for adding or editing menu items (meals and beverages) with photographs and further information (Owners).
- **Add restaurant information**: Include information about the restaurant, such as contact information (Owners).
- **Add restaurant's location**: Restaurant owners can add their locations to the application (Owners).

3.2.2 Non-Functional Requirements

"Non-functional requirements These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards"[8].

- **Reliability**: A database's performance should be consistent without any problems.

- **Accuracy**: Displaying the exact location of the restaurant branch on a map.
- **Response Time**: Reservation requests should be processed in less than 10 seconds.
- **Security**: Encrypting the application to prevent exploitation protects the application and its users from potential risks.
- **Usability**: To guarantee that our application is simple to use for the majority of users, we will followed Shneiderman's 8 Golden Rules.

3.2.3 External Interface Requirement

"External interfaces are the connections and interactions between a software system and external entities. These entities can be other software systems, APIs, web services, hardware devices, databases, or even human users"[9]. Here are the key external interface requirements categorized by type:

- **User Interface (UI) Requirements**:
  - The application should have a user-friendly interface accessible on smartphones.
  - The user interface should allow customers to view available tables, make reservations, and manage their bookings.
  - It should provide real-time availability of tables and allow customers to select their preferred date and time for reservations.
  - The user interface should provide confirmation of reservations, including details such as reservation number, date, time, and number of guests.
- **Hardware Interface Requirements**:
  - The application should be compatible with a variety of hardware devices, including touchscreens, keyboards, and printers.
  - Ensure compatibility of the application or device with NFC technology to enable interaction and communication via NFC.

- **Software Interface Requirements**:
  - The application is developed using Flutter, a framework that uses the Dart programming language.
  - The application is integrated into a cloud database, specifically MySQL.
  - Integration with a secure payment gateway is needed to facilitate online payments for reservations.
  - The application should have an interface for managing reservation orders.
- **Communication Interface Requirements**:
  - The application should have the capability to send notifications and reminders to customers via Firebase Cloud Messaging regarding their upcoming reservations.
  - It should integrate with external APIs, such as google map APIs, to provide directions to the restaurant or review platforms to display customer ratings and reviews.

3.3 User Profile

This section will describe the different user categories and characteristics additionally how they interact with our application.

3.3.1 User Categories

After analyzing our project, we have identified three types of users:

- **The customers**: Customers are the key focus of our application, and they will use our services such as table bookings and food ordering.
- **The restaurant employees**: Restaurant personnel will give services to customers with the assistance of our application.

- **The restaurant owners**: The owners will set up their restaurant on our application and provide their employees with access to the restaurant's account.

3.3.2 Sample Specification

The target audience for our application includes customers who seek convenient reservation services, restaurant owners looking for more efficient reservation solutions, and restaurant employees dedicated to ensuring customer satisfaction. These users will rely on our reservation application to streamline the reservation management process and deliver superior service to enhance the dining experience.

3.3.3 User Characteristics

- **Customers**: Individuals or families seeking dining experiences utilize our app to reserve tables for meals, whether it's for dine-in or takeout services.
- **Restaurant employees**: The receptionist efficiently checks and confirms new customer bookings through the application.
- **Restaurant owner**: Chain restaurant owners utilize the app to efficiently register and manage all branches of their restaurants.

3.3.4 Environment

Our reservation application is compatible with both Android and iOS operating systems, enabling customers to make reservations conveniently from their homes or any location. Our user-friendly application provides instant access to our restaurant's available services and reservations.

3.4 Structuring System Requirements

3.4.1 System Use cases

The use cases below show the various processes that different users could perform in our application.

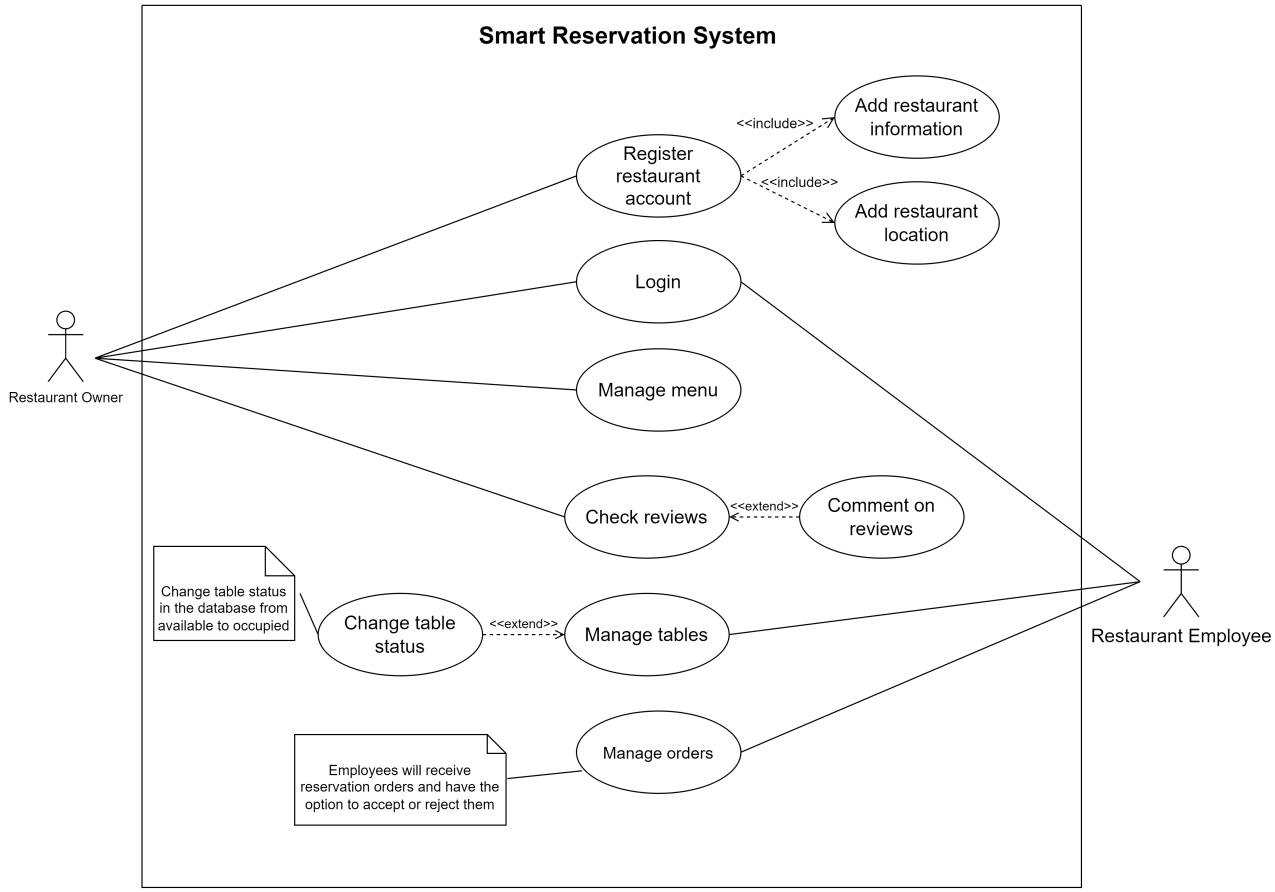


Figure 3.10: The use case diagram for the restaurant

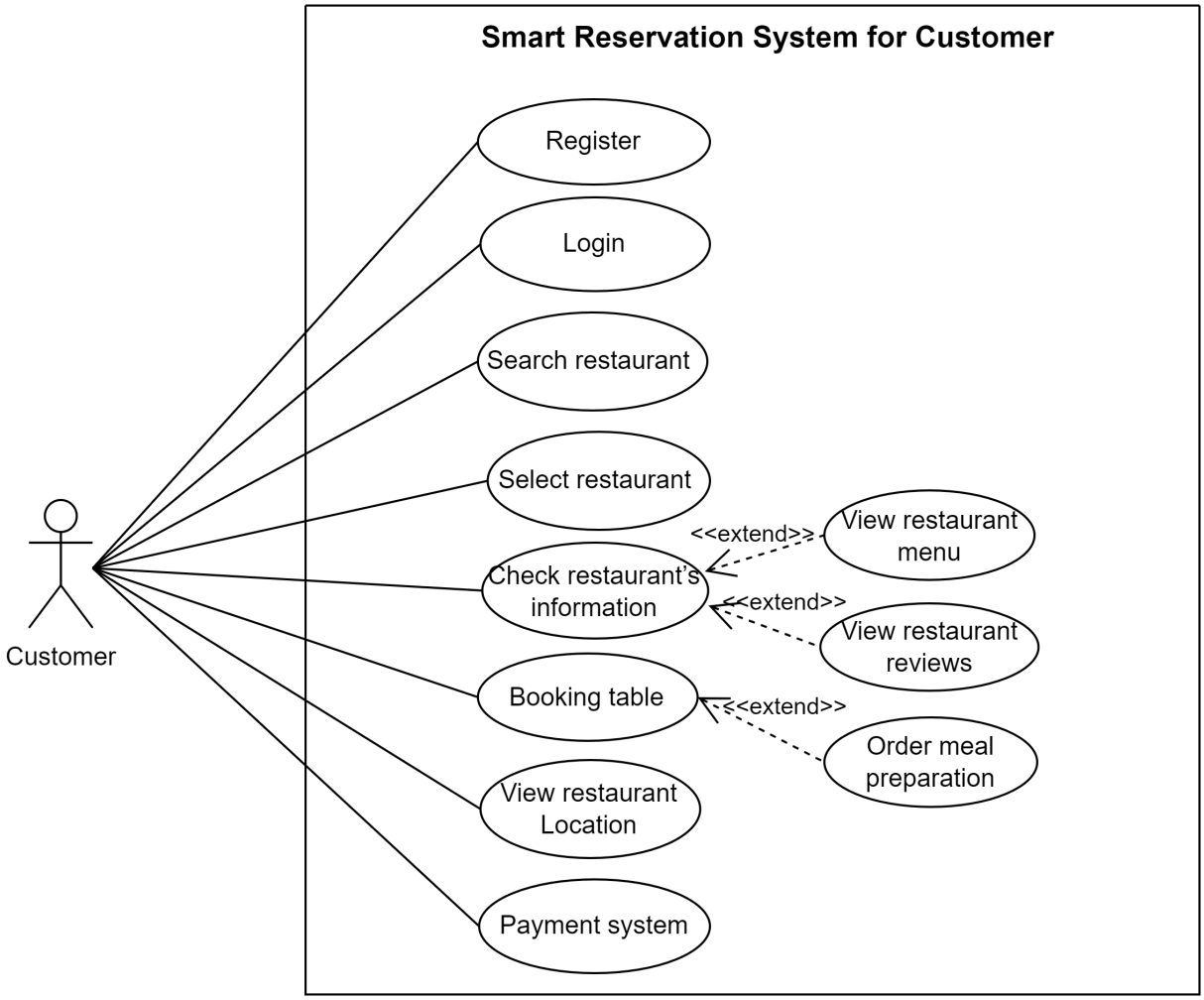


Figure 3.11: The use case diagram for the customer

3.4.2 Data Flow Description

The data flow diagram below shows the flow of information between various elements within the same system for different users.

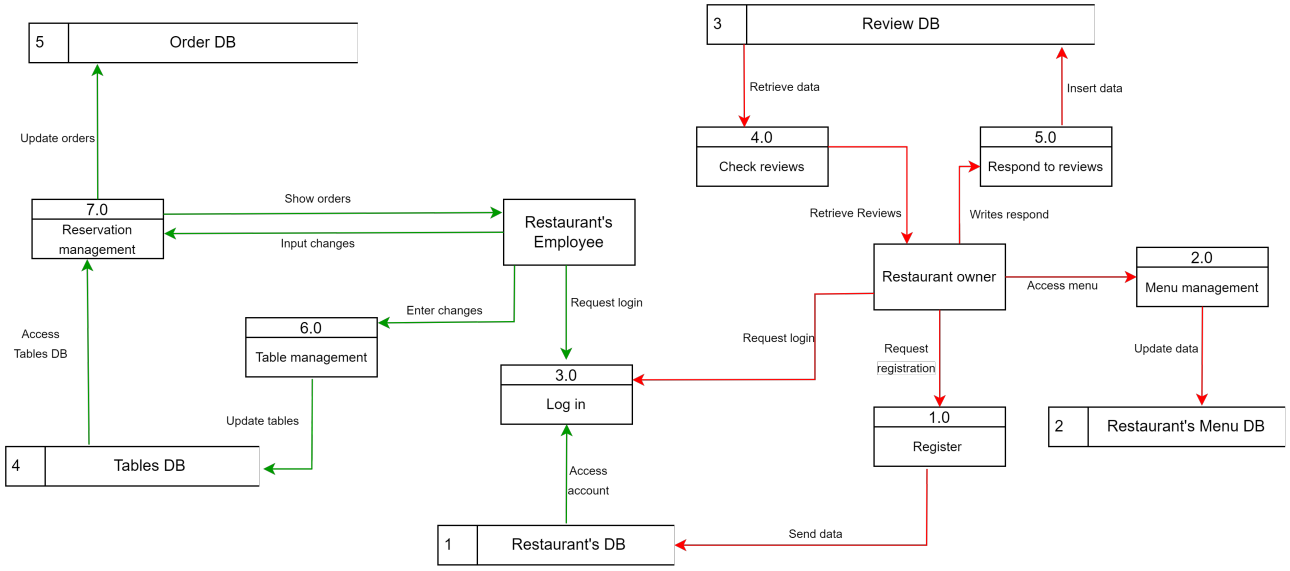


Figure 3.12: The data flow diagram for the restaurant

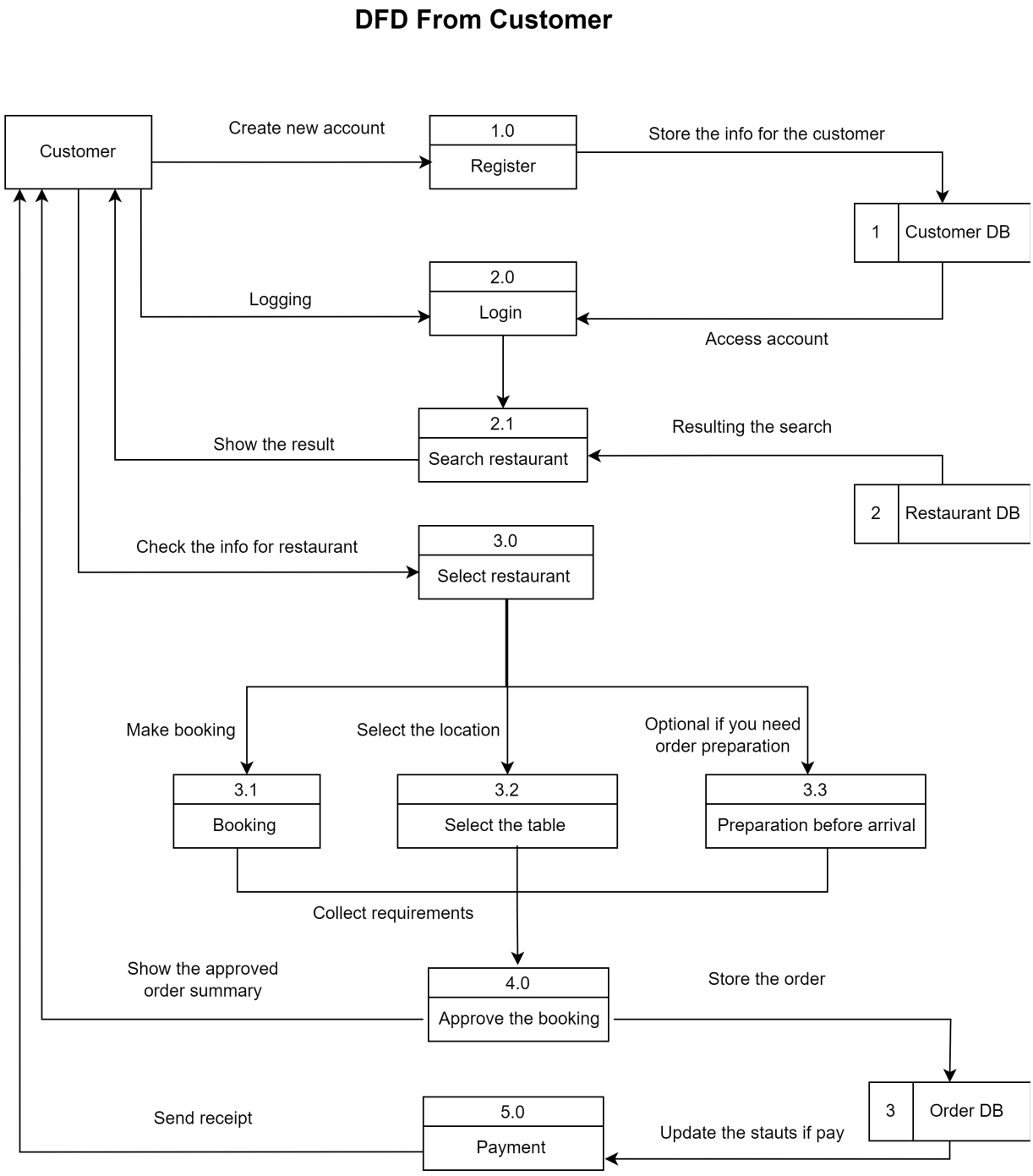


Figure 3.13: The data flow diagram for the customer

Chapter 4

System Design

In this chapter, we created diagrams such as sequence diagram, activity diagram, class diagram, entity relationship diagram, and relational schema diagram. Each diagram is accompanied by an explanation of what it represents in our system.

4.1 Activity Diagram

We created two activity diagrams to demonstrate how some functions are performed in an easily understood way. The first activity diagram shows how a restaurant employee can change the state of a table from available to occupied.

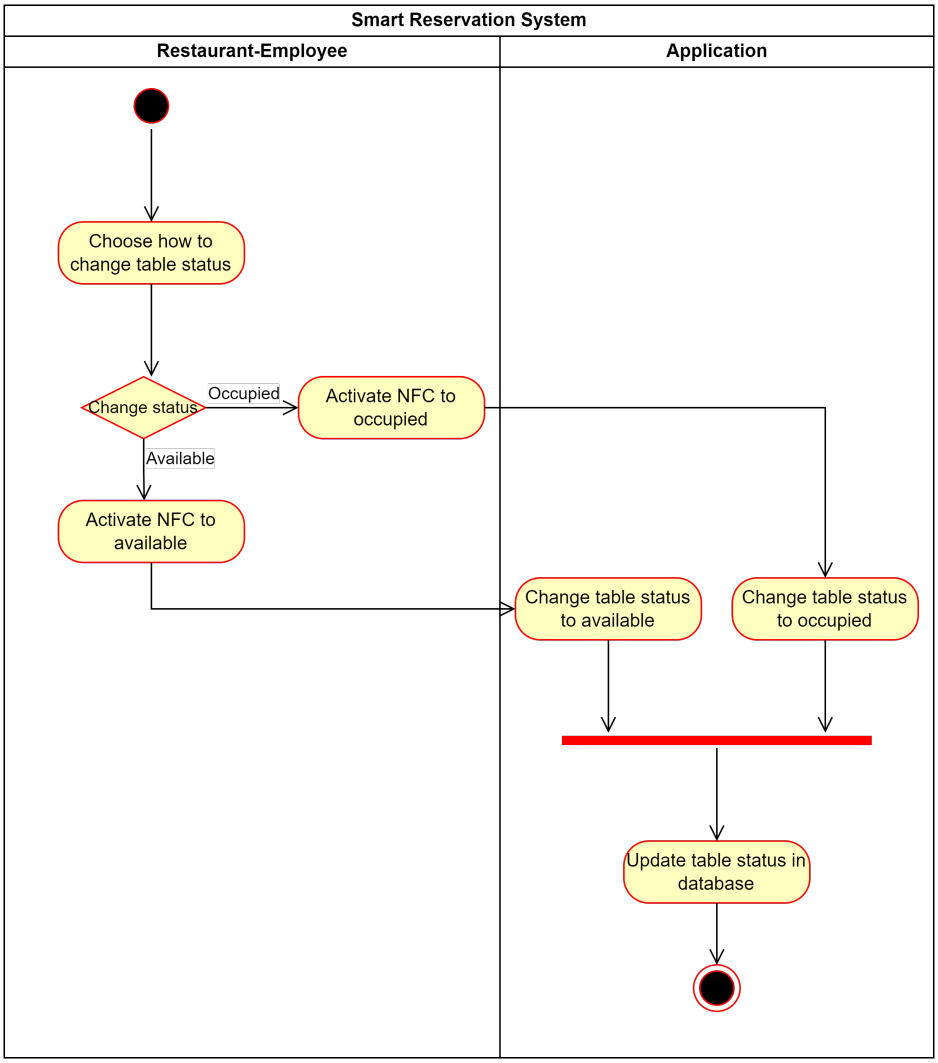


Figure 4.1: Restaurant-Employee changing table status in activity diagram

The second activity diagram shows how a customer makes a reservation at a restaurant.

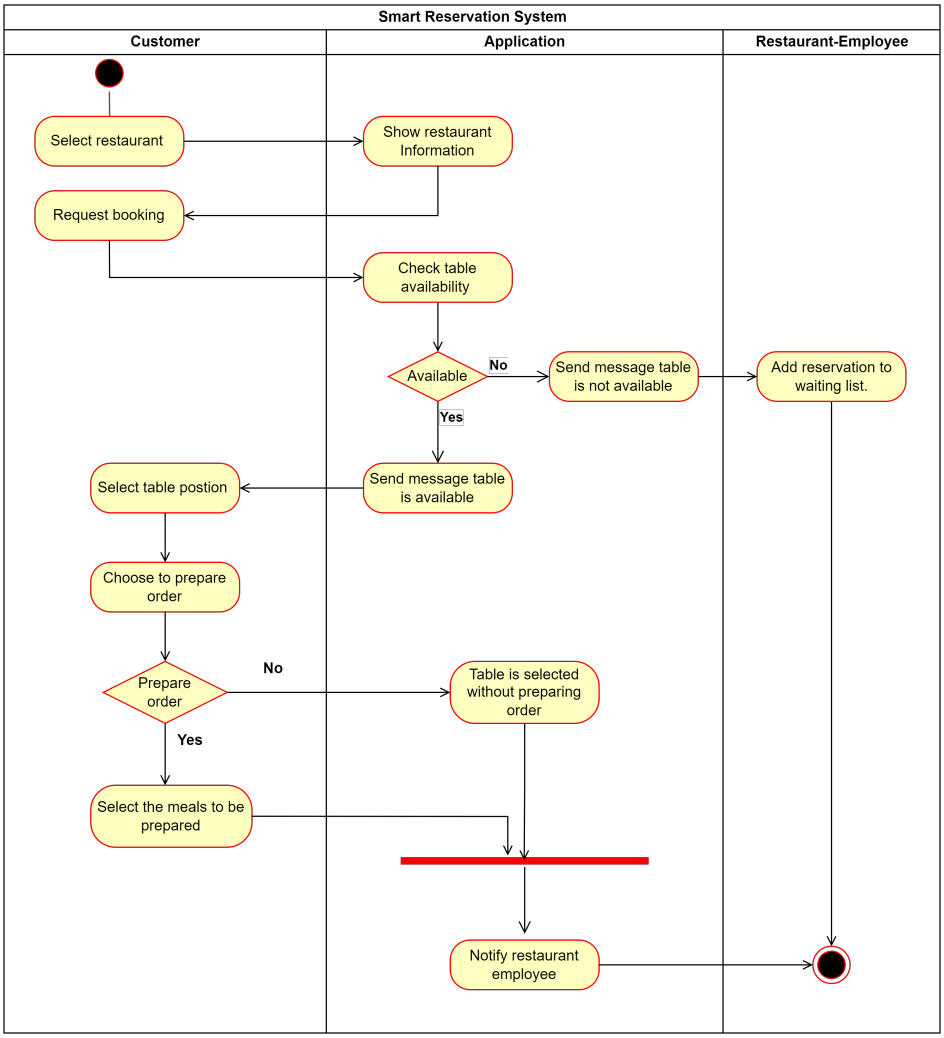


Figure 4.2: Customer selecting restaurant & table in activity iagram

4.2 Sequence Diagram

We created two sequence diagrams to explain how various functions are carried out. The first sequence diagram shows an employee managing tables in the restaurant.

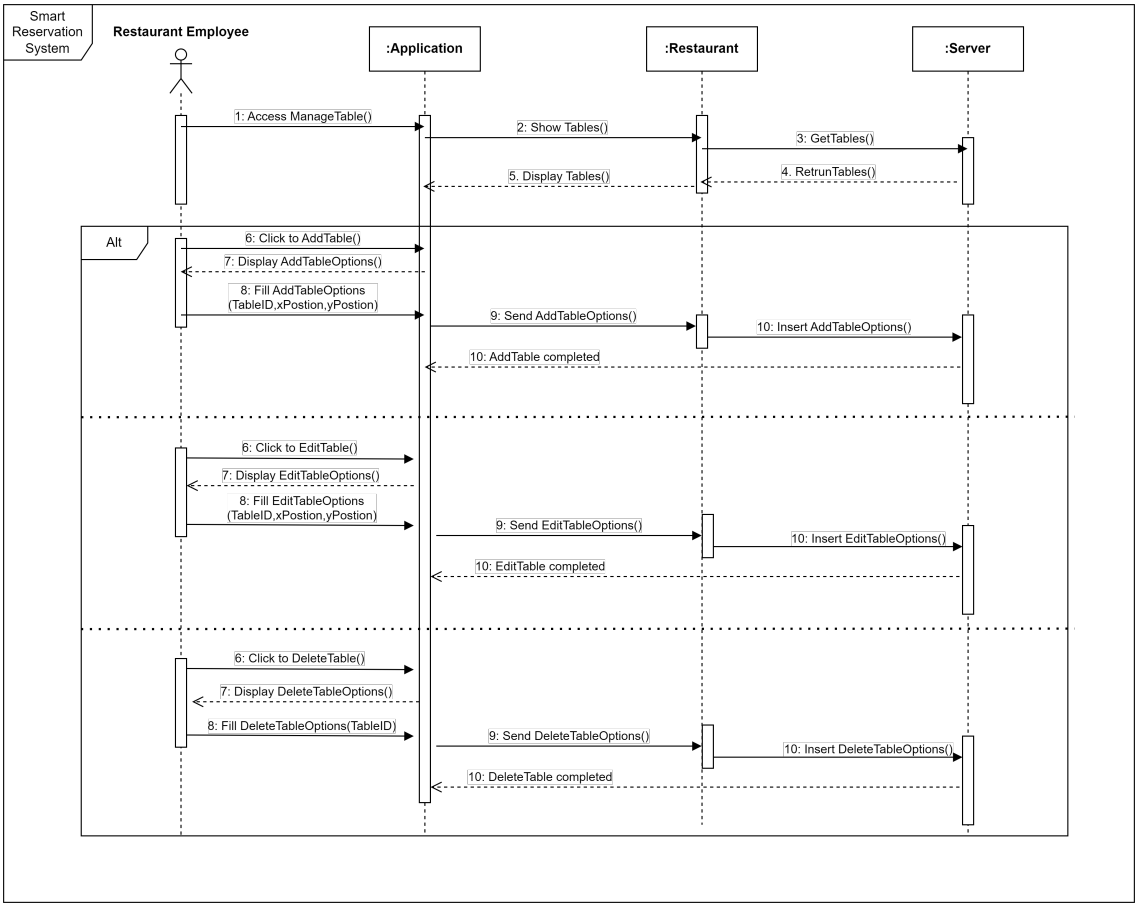


Figure 4.3: Employee changing table status in sequence diagram

The second sequence diagram shows a consumer booking a table at a restaurant.

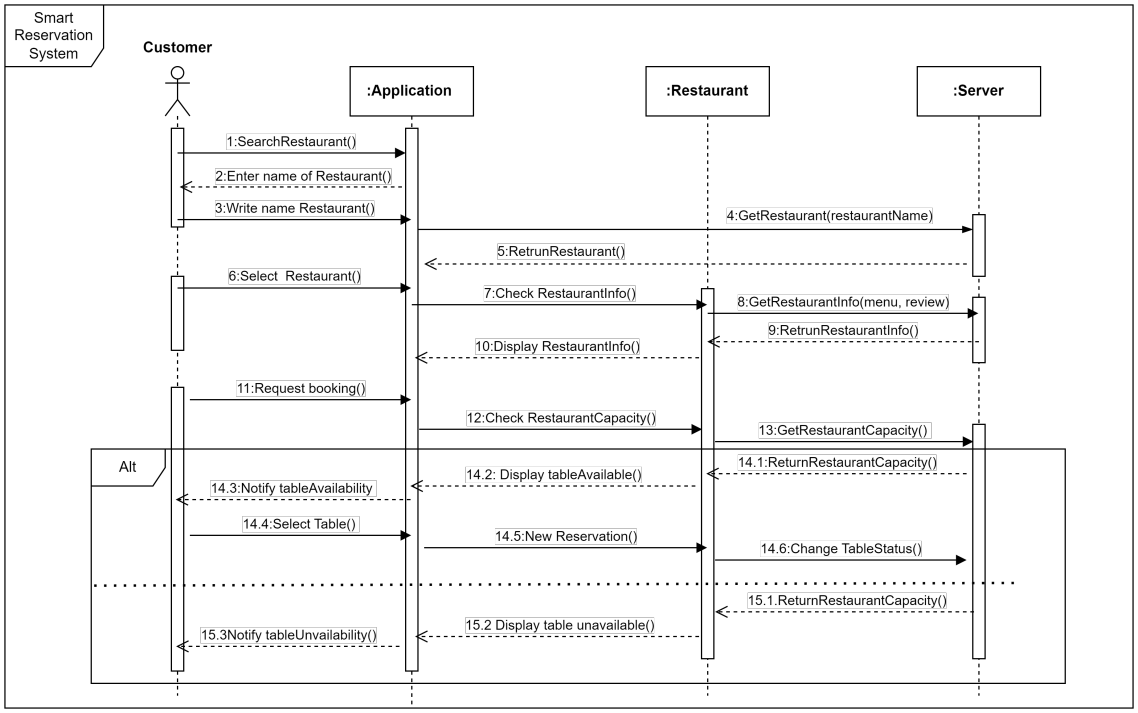


Figure 4.4: Customer booking a table in sequence diagram

4.3 Class Diagram

We created a class diagram to represent the entire system structure. We have written the classes and methods that may be required in our application.

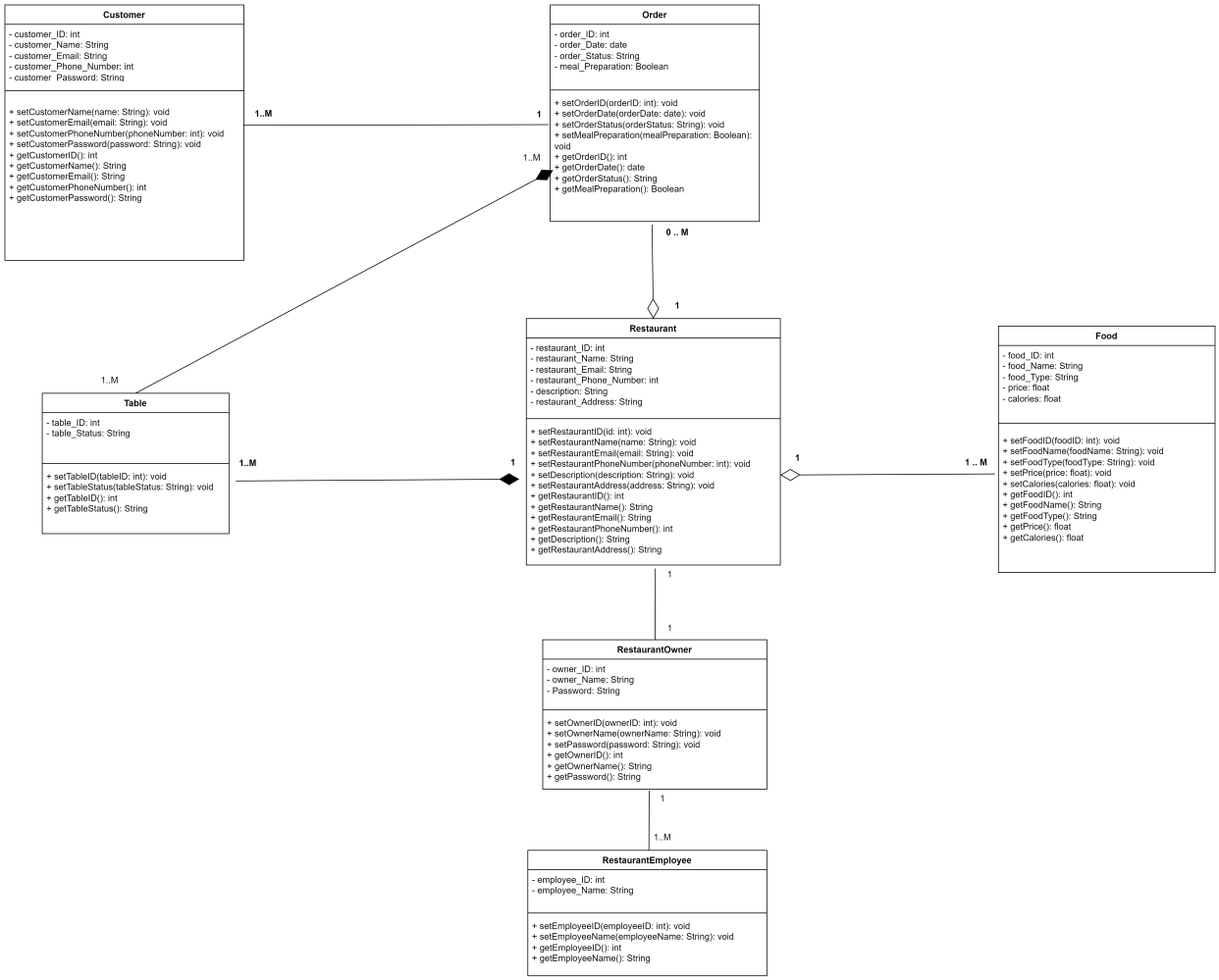


Figure 4.5: Class Diagram in Smart Reservation System

4.4 ER Diagram

We have created an entity relationship diagram and relational schema for the data dictionary for our complete system's data dictionary.

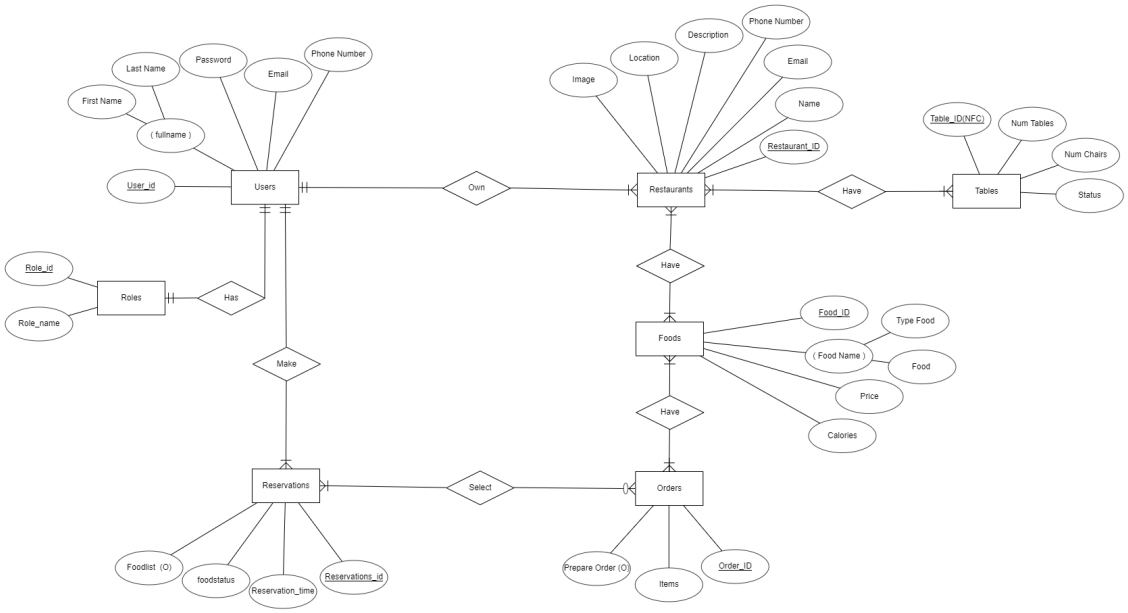


Figure 4.6: Entity relationship diagram for table reservation

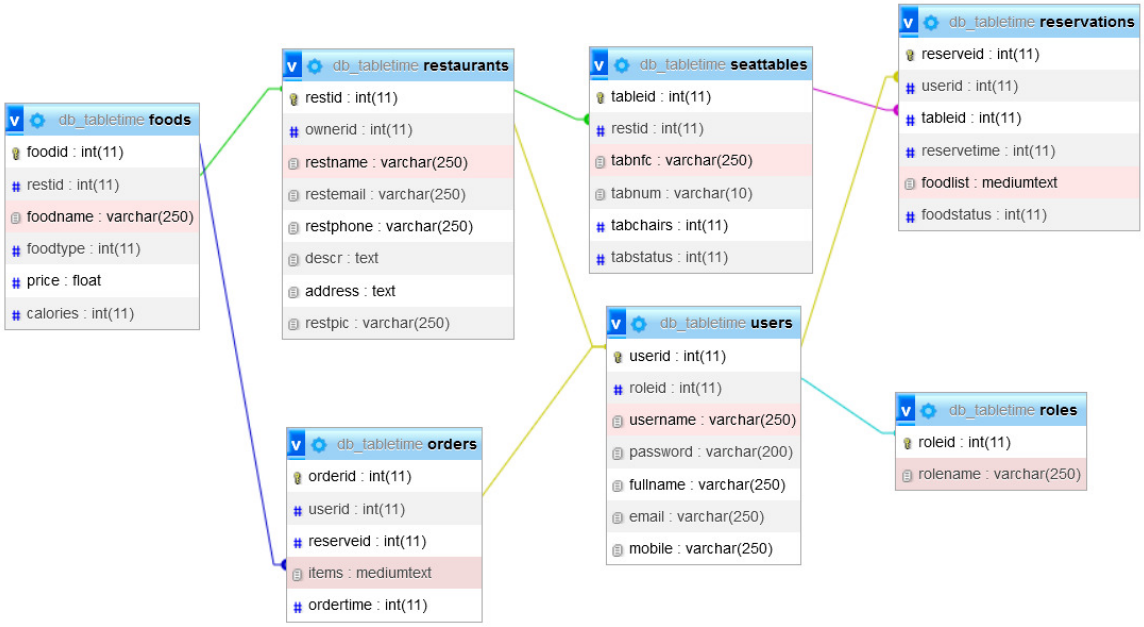


Figure 4.7: Relational schema diagram for table reservation

4.5 Interface Design

We chose to adopt Shneiderman's 8 Golden Rules to make our user interface as user-friendly as possible. The figure below shows our application's home page, where we have applied Shneiderman's 8 Golden Rules, one of which is "Strive for consistency" by making our application layout similar to other applications by having the restaurants visible on the home page and the search for restaurant option visible and familiar with other reservation application's search bars.

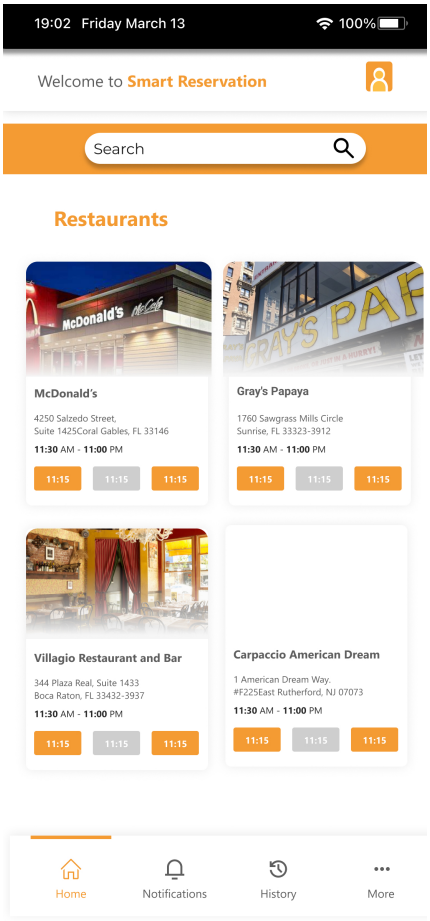


Figure 4.8: Home page customer interface

4.6 Prototype Design

This is the first page that reveals after opening the application; it displays our application's logo and a simple design that hints at what our application is about.

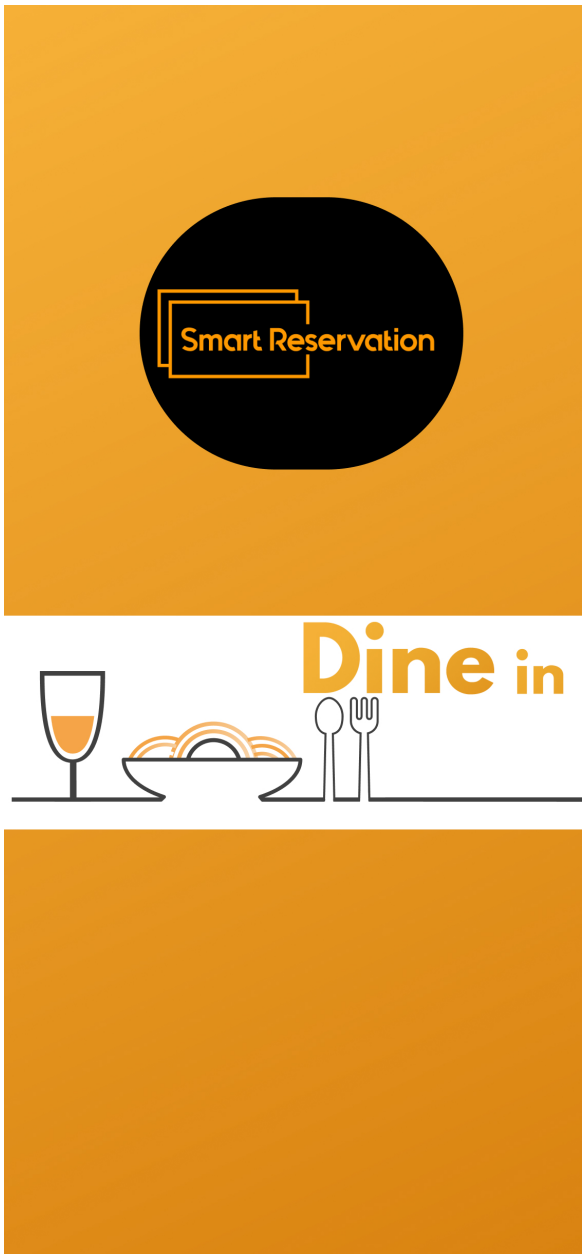


Figure 4.9: Welcome Page in our application

This page takes place after selecting a restaurant; it displays information about the restaurant such as its description, operating hours, and show menu. We can make a reservation on the restaurant's page by selecting a time, date, and number of people then click on select table button.

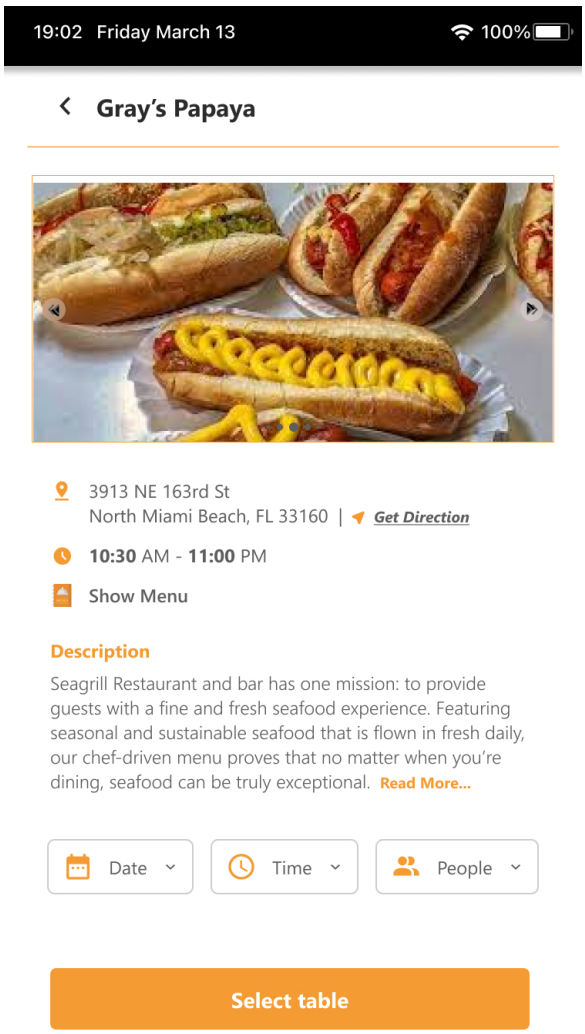


Figure 4.10: Restaurant's page in our application

This page contains a map of the restaurant's interior layout. There are three distinct colors to indicate table status and which table is selected.

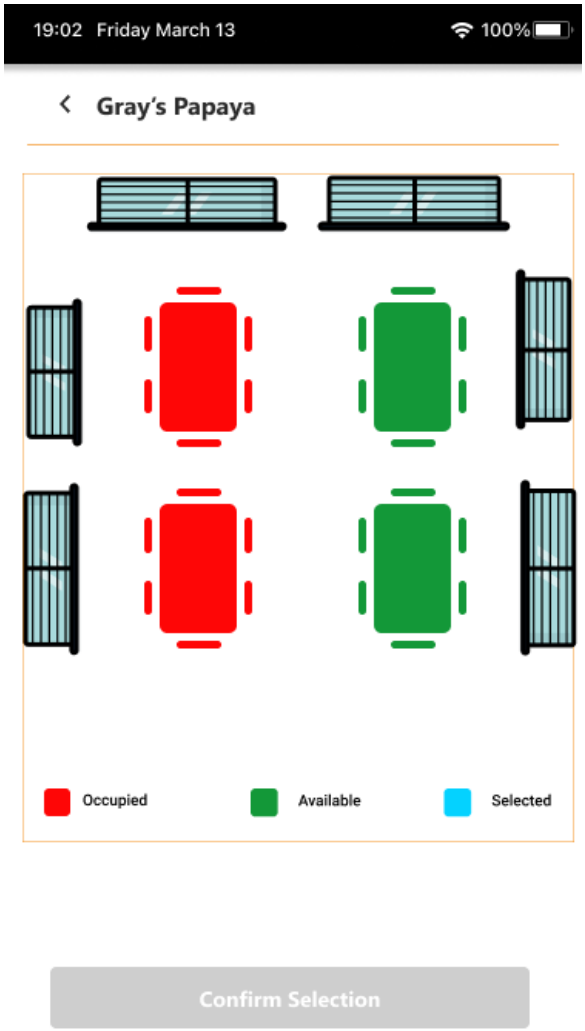


Figure 4.11: Select table in our application

Chapter 5

Implementation

5.1 Tools

This section provides a quick summary of the project's tools and explains their purpose.

5.1.1 Android Studio

Android Studio is an integrated development environment (IDE) created exclusively for Android app development. Android Studio provides seamless integration with Flutter and the Dart programming language. We chose Android Studio because we found it easier and better than others, and its popularity made it easy to find tutorials on how to use it efficiently.

5.1.2 XAMPP

A web server solution stack called XAMPP enables programmers to create and test their applications on a local web server. We determined that testing our application on a local database would be better because it would allow us to make changes to the database more quickly.

5.1.3 Visual Studio Code

Developers use Visual Studio Code as a source code editor. It offers an extensive library of extensions and supports a multitude of programming languages. To make managing the restaurant and orders easier, we used Hypertext Preprocessor (PHP) to make our website work in tandem with a mobile application.

5.1.4 NFC Technology

Devices can share data through Near Field Communication (NFC), a short-range wireless communication technique, when they are brought close to one another or come into contact with one another. Our application efficiently uses NFC technology to improve restaurant order-handling operations.

5.2 Interface description

Figure 5.1: Home Page

The Home Page is the initial page that owners/users see when they open an application. It serves as the beginning point for the user's experience within the application and usually includes a registration option for first-time visitors or a login option for returning users who have already made an account.

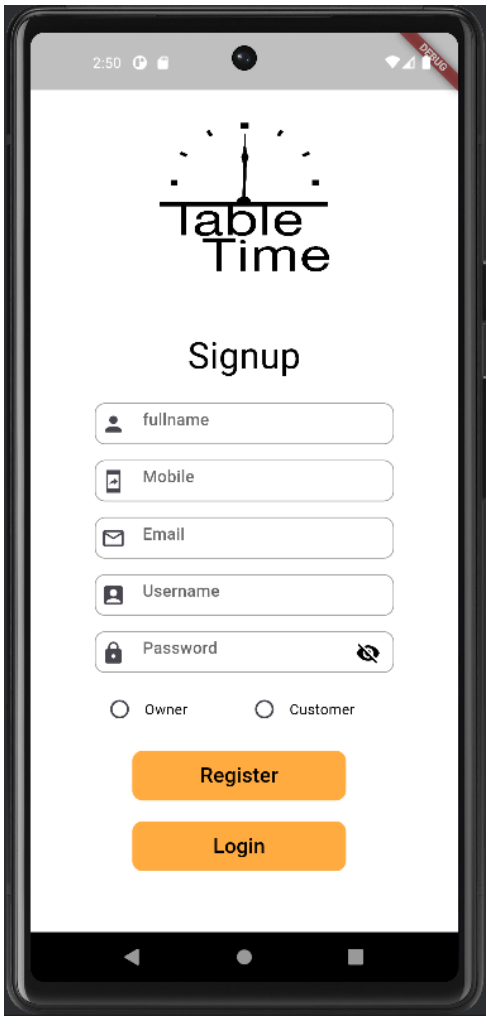


Figure 5.2: Register Page

The Register Page is the page that owners/users see when they decide to register with the application. It is used to create a new account and often includes areas that need personal information such as full name, mobile number, email, username, and password. In addition, users may choose whether to create an account as an owner or a customer.

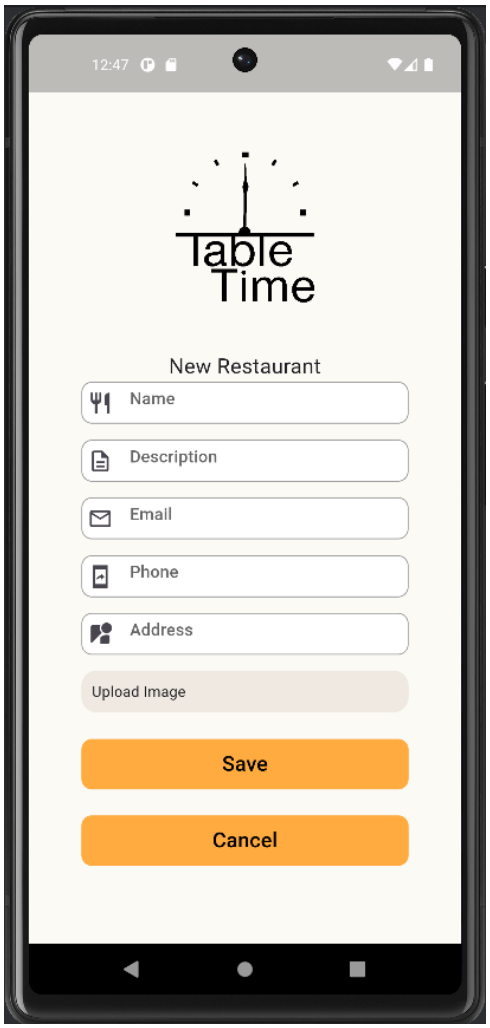


Figure 5.3: Add Restaurant Page

The Add Restaurant Page is where owners can register a new restaurant in the application. This page simplifies the input and arrangement of important restaurant information. Owners are required to provide basic details such as the restaurant's name, email address, phone number, and physical address. After submitting this information, they can add the restaurant to the application's database.

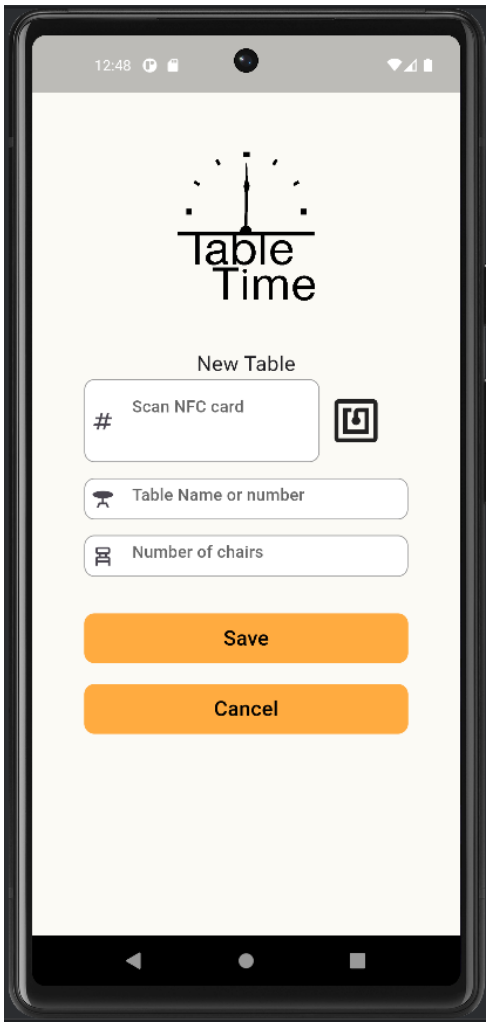


Figure 5.4: Add Table Page

The Add Table Page is where owners may add new tables to their application. It is used to add data to a new table and save it to the database. The required information includes the table's serial number, which may be scanned with an NFC tag reader or input manually, the table number, and the number of chairs. After entering the necessary information, owners should click the save button to store the new table's information and add it to the main list of tables.

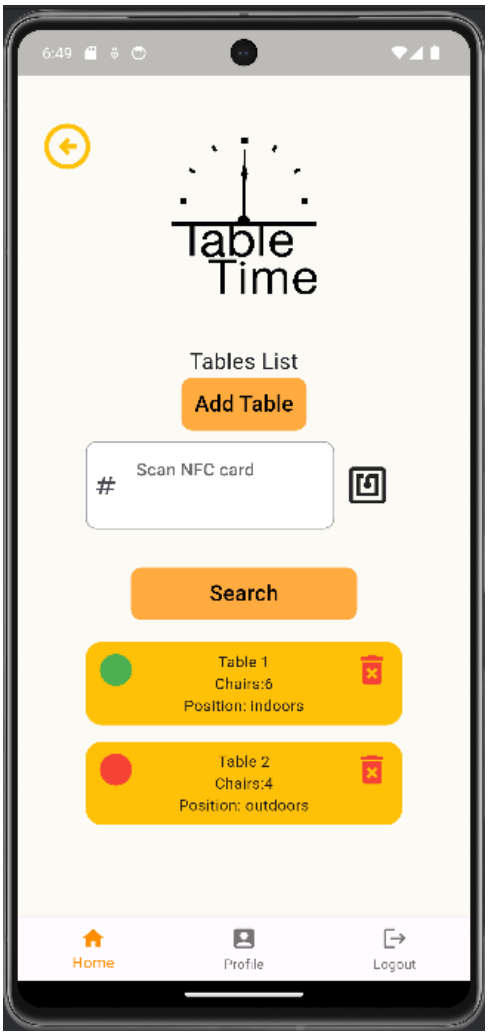


Figure 5.5: Table List Page

The Table List Page shows the number of tables available in the restaurant, as well as their state (available or occupied). This page offers restaurant employees with an overview of the current table configuration, allowing them to handle seating arrangements more efficiently. In the middle of the page, we can see a search option for searching tables by scanning NFC tags. Tables are sorted according to their availability state, allowing personnel to rapidly identify and assign tables to customers who come in.

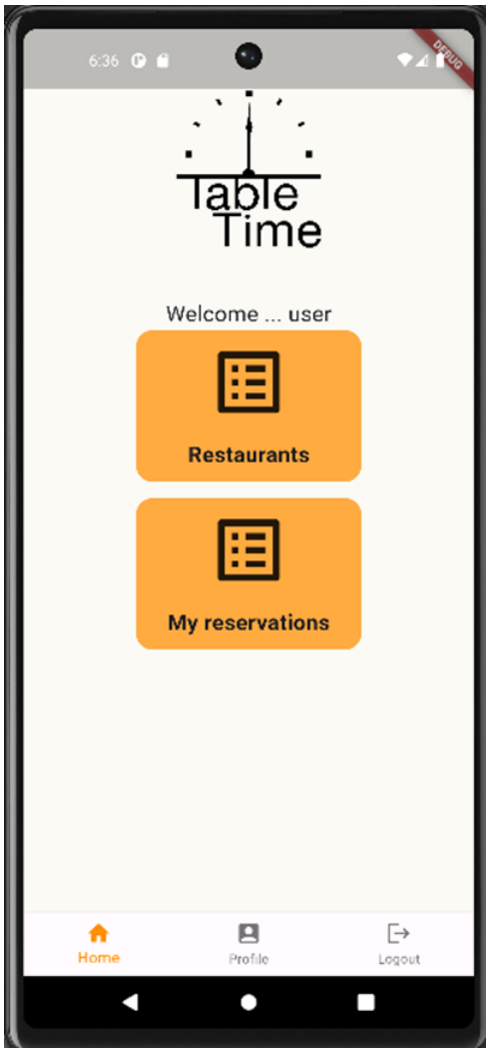


Figure 5.6: Customer Home Page

This is the customer's home page, and it contains 5 options: restaurants to list available restaurants and start reservations, My reservation displays all of the reservations the consumer made. Home page goes to home page, which works on any page, profile is where the consumer edits their profile, and finally log out.

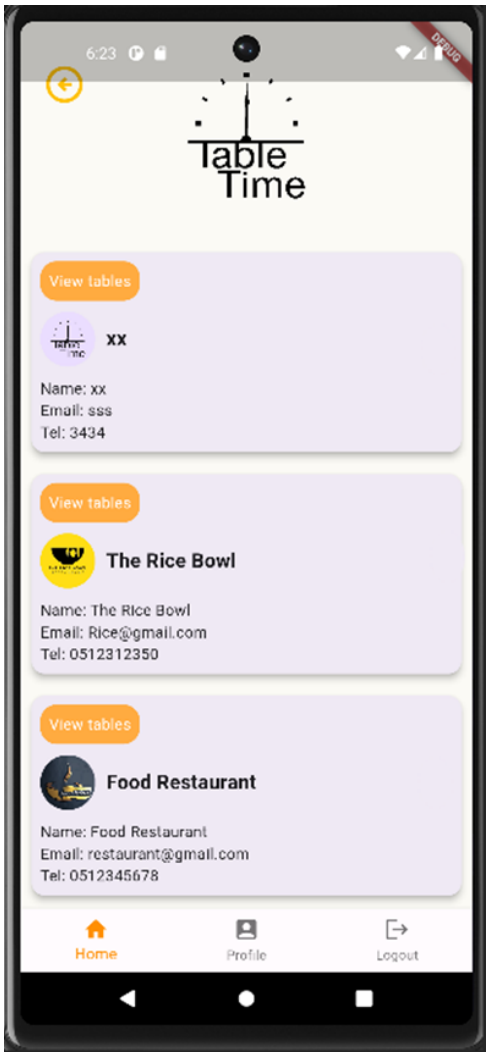


Figure 5.7: Customer Restaurant Page

The restaurant page displays all of the available restaurants in the Table Time application, as well as the name, email, and phone number of each restaurant. To make a reservation, click the View Tables button to view the table list.

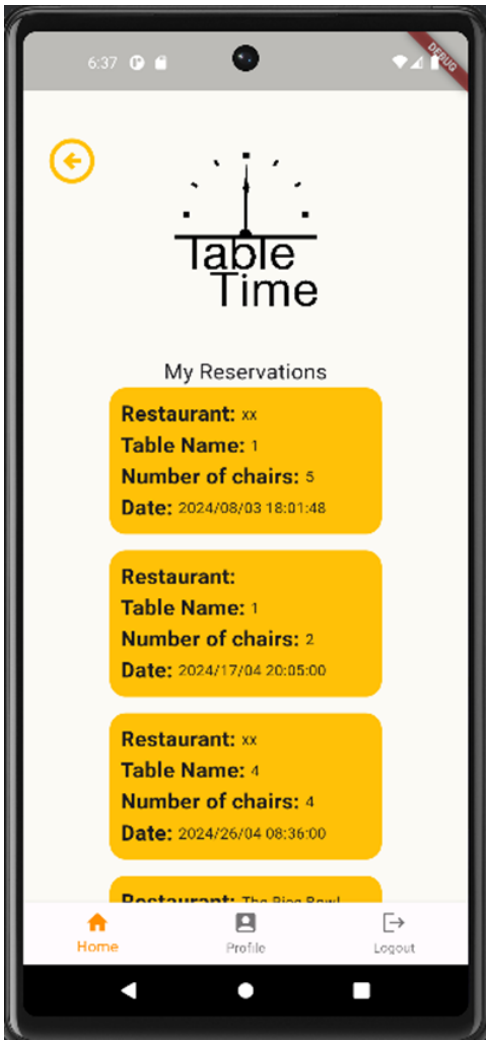


Figure 5.8: My Reservation Page

This page displays all prior and new reservation orders, as well as the restaurant where the user placed the reservation, the table number, the number of chairs, and the date.

5.3 Walkthrough the system

In our system we have different components working together for registration purposes, this section will explain how they function in each component of the system.

```
void createAccount() {
  String fullname = _nameController.value.text.toString();
  String mobile = _mobileController.value.text.toString();
  String email = _emailController.value.text.toString();
  String username = _usernameController.value.text.toString();
  String password = _passwordController.value.text.toString();

  Map<String, dynamic> posts = { "username":username, "password":password,
    "fullname":fullname, "mobile":mobile, "email": email,
    "roleid":_character.toString()};
  dbservices.doPost("register", posts).then((result) {
```

Figure 5.9: Register Part 1

1- Register page in Android Studio:

- The register function consists of controllers which control textboxes.
- The information entered into these textboxes is saved as variables.
- The variables are then posted using a variable named "posts" of type `Map<String, dynamic>`.
- In Dart, a Map is an unordered collection of key-value pairs, where the keys are of type String and the values can be of any type (`dynamic`).
- After posting, the variable "posts" passes through validation processes.

```php
function validatePassword($password){
    $state = true; $passwordErr = '';
    if(!empty($password) && isset( $password )) {
        $state = true;
        if(!preg_match("#[0-9]+#",$password)) {
            $passwordErr = "Your Password Must Contain At Least 1 Number!";
            $state = false;
        }
        elseif(!preg_match("#[A-Z]+#",$password)) {
            $passwordErr = "Your Password Must Contain At Least 1 Capital Letter!";
            $state = false;
        }
        elseif(!preg_match("#[a-z]+#",$password)) {
            $passwordErr = "Your Password Must Contain At Least 1 Lowercase Letter!";
            $state = false;
        }
        elseif(!preg_match("#[\W]+#",$password)) {
            $passwordErr = "Your Password Must Contain At Least 1 Special Character!";
            $state = false;
        }
    } else {
        $passwordErr = "Please enter password   ";
        $state = false;
    }
    $passvalid = array("valid"=>$state, "msg"=>$passwordErr);
    return $passvalid;
}
```

Figure 5.10: Register part 2

2- PHP Registration Validation:

- This system validates passwords according to preset requirements. Password rules are as follows:

    1. The password should not be empty.
    2. Passwords must include at least one number.
    3. Passwords must have at least one capital letter.
    4. Passwords must have at least one lowercase letter.
    5. Passwords must include at least one special character.

- Other validations for fields include username, full name, email, and mobile number.

```
dbservices.doPost("register", posts).then((result) {
  Map<String, dynamic> res = jsonDecode(result);
  var block = res['response'];
  Fluttertoast.showToast(
      msg: block['msg'],
      toastLength: Toast.LENGTH_LONG,
      gravity: ToastGravity.BOTTOM,
      timeInSecForIosWeb: 1,
      backgroundColor: (block['status'] == "done") ? Colors.green : Colors.red,
      textColor: Colors.white,
      fontSize: 16.0
  );
  if (block['status'] == "done") {
    Navigator.push(context,
        MaterialPageRoute(builder: (context) => Login(lang:lang, )));
  }
});
}
```

Figure 5.11: Register Part 3

3- Validation Message and Open Login Page:

- To register, the code sends an HTTP POST request to the "register" server endpoint and stores the information in the "posts" variable of type `Map<String, dynamic>`.
- The server's response is decoded and stored in the "block" variable.
- The `Fluttertoast.showToast` method displays a toast message on the screen, which is retrieved from `"block['msg']"`.
- The toast message's background color is green if `"block['status']"` equals "done", and red otherwise.
- If the registration is successful (when `"block['status']"` returns "done"), the code redirects to the login page.

In our system we have different components working together for login purposes, this section will explain how they function in each component of the system.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    body: Directionality(
      textDirection: TextDirection.ltr,
      child: ListView(
          children: [
        Container(
            margin: const EdgeInsets.fromLTRB(0, 50, 0, 10),
            width: MediaQuery.of(context).size.width * 0.65,
            height: MediaQuery.of(context).size.height * 0.25,
            decoration: const BoxDecoration(
              image: DecorationImage(
                  image: AssetImage('assets/images/logo.png'),
                  fit: BoxFit.contain
              ),  // DecorationImage
            ),  // BoxDecoration
          ),  // Container
        Container(
            width: MediaQuery.of(context).size.width,
            decoration: const BoxDecoration(
            ),  // BoxDecoration
            child: Column(
```

Figure 5.12: Log-in Part 1

1. User Interface Setup:

- After running the application, the user will see the login screen.
- The screen prompts users to enter their login and password, as well as choices for password recovery and navigation.

```
void dologin() {
  String username = _usernameController.text.toString();
  String password = _passwordController.text.toString();

  Map<String, dynamic> posts = {"act":"login", "username":username, "password":password};
```

Figure 5.13: Log-in Part 2

2. Login:

- Enter username and password.
- Passwords can be shown or hidden for added security.
- When you click the "Login" button, the entered credentials are transmitted to the server for verification.

```
dbservices.doPost("login", posts).then((result) {
  Map<String, dynamic> res = jsonDecode(result);
  var block = res['response'];
  Fluttertoast.showToast(
      msg: block['msg'],
      toastLength: Toast.LENGTH_LONG,
      gravity: ToastGravity.BOTTOM,
      timeInSecForIosWeb: 1,
      backgroundColor: (block['status'] == "done") ? Colors.green : Colors.red,
      textColor: Colors.white,
      fontSize: 16.0
  );
```

Figure 5.14: Log-in Part 3

3. Verification Process:

- The application requests the server, including the login and password.
- The server validates credentials and returns the authentication result.
- An alert message indicates if the login attempt was successful or failed.

```
  if (block['status'] == "done") {
    String userid = block['userid'];
    String roleid = block['roleid']; // Admin, Provider, Requester, Cistomerservice
    var rows = block['rows'];

      Navigator.push(context,
          MaterialPageRoute(builder: (context) =>
              dashBoard(userid: userid, username: username, roleid:roleid,  rows:rows, lang:lang)));
  }
});
```

Figure 5.15: Log-in Part 4

4. Navigating to Dashboard:

- When a successful login occurs, the user is sent to the dashboard screen.
- The dashboard panel presents information related to the user's job and permissions.
- The dashboard receives additional data, including user and role IDs, for further customization.

In our system we have different components working together to add new restaurant purposes, this section will explain how they function in each component of the system.

```
Container(
  width: MediaQuery.of(context).size.width * 0.7,
  height: 40,
  margin: const EdgeInsets.fromLTRB(0, 0, 0, 15),
  decoration: BoxDecoration(
    border: Border.all(width: 1, color: Colors.grey),
    color: const Color(0xFFFFFFFF),
    borderRadius: BorderRadius.circular(10.0),
  ),  // BoxDecoration
  child: TextField(
    controller: _nameController,
    decoration: const InputDecoration(
      border: InputBorder.none,
      hintText: "Name",
      icon: Padding(
        padding: EdgeInsets.only(top: 1.0, left: 5,),
        child: Icon(Icons.card_membership),
      ),  // Padding
    ),  // InputDecoration
  ),  // TextField
),  // Container
```

Figure 5.16: Add Restaurant Part 1

1. Input Fields:

- The screen shows input fields for the following information:
  - Users can input the name of the new eatery.
  - Email: Users supply the restaurant's email address.
  - Users provide their phone number to contact the restaurant.
  - Users specify the restaurant's address.

```
// Button Register
Container(
  width: MediaQuery.of(context).size.width * 0.5,
  margin: const EdgeInsets.fromLTRB(0, 20, 10, 20),
  decoration: BoxDecoration(
    color: Colors.orangeAccent,
    borderRadius: BorderRadius.circular(10),
  ),  // BoxDecoration
  child: TextButton(onPressed: () {
    saveRest();
  }, child: const Text('Save',
    style: TextStyle(color: Colors.black, fontSize: 20),),  // Text
  ),  // TextButton
),  // Container
Container(
  width: MediaQuery.of(context).size.width * 0.5,
  margin: const EdgeInsets.fromLTRB(0, 0, 10, 0),
  decoration: BoxDecoration(
    color: Colors.orangeAccent,
    borderRadius: BorderRadius.circular(10),
  ),  // BoxDecoration
  child: TextButton(
    onPressed: () { Navigator.pop(context); },
    child: const Text('Cancel',
      style: TextStyle(color: Colors.black, fontSize: 20),
```

Figure 5.17: Add Restaurant Part 2

2. Save and Cancel Buttons:

- Users can store their input information by clicking the "Save" button.
- Alternatively, users may cancel the registration process by clicking the "Cancel" button.

```
void saveRest() {
  String restname = _nameController.text;
  String email = _emailController.text;
  String phone = _telController.text;
  String address = _addressController.text;

  Map<String, dynamic> posts = {"act": "saveRest", "restname":restname,
    "email": email, "phone":phone, "address":address, "userid": userid};
  dbservices.doPost("saveRest", posts).then((result) {
    Map<String, dynamic> res = jsonDecode(result);
    var block = res['response'];
    if (block['status'] == "done") {
      var rows = block['rows']; // user profile data
```

Figure 5.18: Add Restaurant Part 3

3. Saving Restaurant Information:

- Click the "Save" button to process input restaurant information.
- The correctness of the entered data is verified, and the restaurant information is stored.
- After successfully registering, users are brought to a page that displays the restaurant list.

```
    Navigator.pushReplacement(context,
        MaterialPageRoute(builder: (context) =>
            restaurants(userid: userid, username: username, roleid: roleid,
              rows: rows, lang: lang,)));  // restaurants, MaterialPageRoute
  }
});
```

Figure 5.19: Add Restaurant Part 4

4. Navigation:

- Click the "Cancel" button to cancel a registration and return to the previous screen.
- After successfully registering, users are directed to the page that displays the restaurant list.

In our system we have different components working together to add new tables and chairs purposes, this section will explain how they function in each component of the system.

```
void _tagRead() {
  NfcManager.instance.startSession(onDiscovered: (NfcTag tag) async {
    Map<String, dynamic> tagvalue = tag.data;
    String tagvalueStr = tagvalue.toString();
    var tagAry = tagvalueStr.split("{identifier: ");
    var vals = tagAry[1].split("],");
    String TagVal = vals[0] + "]";
    _nfcController.text = TagVal.trim();
    print(tagvalue);
    NfcManager.instance.stopSession();
  });
}
```

Figure 5.20: Add Table Chairs Part 1

1. Scan the NFC Card or Manually Enter Data:

- Users may scan or manually enter data from an NFC card into the applicable spaces.

```
      child: TextField(
        controller: _tablenumController,
        decoration: const InputDecoration(
          border: InputBorder.none,
          hintText: "Table Name or number",
          icon: Padding(
            padding: EdgeInsets.only(top: 1.0, left: 5,),
            child: Icon(Icons.card_membership),
          ),  // Padding
        ),  // InputDecoration
      ),  // TextField
    ),  // Container
    Container(
      margin: const EdgeInsets.fromLTRB(50, 0, 50, 15),
      width: MediaQuery.of(context).size.width * 0.7,
      height: 40,
      decoration: BoxDecoration(
        border: Border.all(width: 1, color: Colors.grey),
        color: const Color(0xFFFFFFFF),
        borderRadius: BorderRadius.circular(10.0),
      ),  // BoxDecoration
      child: TextField(
        controller: _tabchairController,
```

Figure 5.21: Add Table Chairs Part 2

2. Provide Table Number and Chairs:

- Users provide the table number and chair count for the new table.

```
        child: TextButton(onPressed: () {
          saveTable();
        }, child: const Text('Save',
          style: TextStyle(color: Colors.black, fontSize: 20),),  // Text
        ),  // TextButton
      ),  // Container
      Container(
        width: MediaQuery.of(context).size.width * 0.5,
        margin: const EdgeInsets.fromLTRB(50, 0, 50, 0),
        decoration: BoxDecoration(
          color: Colors.orangeAccent,
          borderRadius: BorderRadius.circular(10),
        ),  // BoxDecoration
        child: TextButton(
          onPressed: () { Navigator.pop(context); },
          child: const Text('Cancel',
            style: TextStyle(color: Colors.black, fontSize: 20),
          ),  // Text
        ),  // TextButton
      ),  // Container
    ],
```

Figure 5.22: Add Table Chairs Part 3

3. Click the "Save" button to Store the Data and Create a New Table:

- Users may save their data by clicking the "Save" button and creating a new table.

```
void saveTable() {
  String nfc = _nfcController.text;
  String tablenum = _tablenumController.text;
  String chairnum = _tabchairController.text;
  var _save = true;
  if(tablenum == ""){
    _save = false;
  }
  if(chairnum == ""){
    _save = false;
  }
```

Figure 5.23: Add Table Chairs Part 4

4. Validate the Data:

- Validate data before saving, including table numbers and chair counts.

```
if(_save) {
  Map<String, dynamic> posts = {
    "act": "saveTable", "nfc": nfc, "restid": restid, "tablenum": tablenum,
    "chairnum": chairnum, "userid": userid
  };
  dbservices.doPost("saveTable", posts).then((result) {
    Map<String, dynamic> res = jsonDecode(result);
    var block = res['response'];
```

Figure 5.24: Add Table Chairs Part 5

5. Processing and Storing Data:

- The data is validated, processed, and saved in the database.

```
dbservices.doPost("saveTable", posts).then((result) {
  Map<String, dynamic> res = jsonDecode(result);
  var block = res['response'];
  if (block['status'] == "error") {
    Fluttertoast.showToast(
        msg: block['msg'],
        toastLength: Toast.LENGTH_LONG,
        gravity: ToastGravity.BOTTOM,
        timeInSecForIosWeb: 1,
        backgroundColor: Colors.red,
        textColor: Colors.white,
        fontSize: 16.0
    );
  }
  if (block['status'] == "done") {
    var rows = block['rows']; // user profile data
```

Figure 5.25: Add Table Chairs Part 6

6. User Feedback Message:

- The user receives a confirmation message upon successful data saving, or an error message if there are any issues.

Chapter 6

Testing

6.1 White-box Testing

The term "white box" refers to “an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems”[10].
The tables below illustrate the numerous scenarios that a customer could perform, as well as the debug results.

Table 6.1: Register Unit Test 1

Test Cases	Scenario	Full Name	Mobile	Email	Username	Password	Role	Expected Result	Actual Result	Conclusion
1	Register new user successfully	Turki A	0599999999	TurkiA@gmail.com	TurkiA	Turki12#	Customer	Account created	Account created	Pass
Debug Results	I/flutter (13088): Posts: {username: TurkiA, password: Turki12#, fullname: Turki A, mobile: 0599999999, email: Turki@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"done","msg":"Account Created"}}									
2	Register new user with a 60 length full name	Alexander Maximilian Ferdinand Bartholomew Harrington-Smythe	0512345678	Alex@gmail.com	Alexander	Alexan1#	Customer	Error message, full name exceeded allowed length	Account created	Fail
Debug Results	I/flutter (13088): Posts: {username: Alexander, password: Alexan1#, fullname: Alexander Maximilian Ferdinand Bartholomew Harrington-Smythe, mobile: 0512345678, email: Alex@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"done","msg":"Account Created"}}									
3	Register new user with more than 600 length full name	More than 600 length name	0523456789	M@gmail.com	longtest	Test1234$	Customer	Error message, full name exceeded allowed length	Account created, but database stored only 250 string length	Fail
Debug Results	I/flutter (13088): Posts: {username: longtest, password: Test1234$, fullname: Wilhelmina Alexandrina Constantina Seraphina Anastasia Beatrice Cecilia Theodora Josephina Henrietta Francesca Genevieve Isabella Gabriella Arabella Valentina Emmanuella Marcella Angelina Rosalina Carolina Felicia Octavia Ophelia Amelia Georgiana Arabella Wilhelmina Victoria Adelaide Eugenia Clementina Sophia Emilia Charlotte Elizabeth Amelia Augusta Louisa Victoria Clementina Matilda Jane Amelia Eleanor Octavia Sophia Georgiana Wilhelmina Beatrice Henrietta Caroline Amelia Philippa Augusta Josephine Cecilia Sophia Charlotte Alexandrina Amelia Victoria Francesca Helena Eugenie Georgiana Amelia Clementina Louisa Victoria Alexandrina Amelia Francesca Charlotte Georgiana Amelia Frederica Charlotte Marie Josepha Sophia Wilhelmina Christina Theodora Elfreda Augusta Maud Maria Elise Hortense Felicite Aurora Ellen Winifred Leopoldina Grace Clementina Elvira Mabel Violet Ethel May Georgina Georgiana Beatrice Matilda Henrietta Louisa Victoria Agatha Christian I/flutter (13088): Response: {"response":{"status":"done","msg":"Account Created"}}									

Table 6.2: Register Unit Test 2

Test Cases	Scenario	Full Name	Mobile	Email	Username	Password	Role	Expected Result	Actual Result	Conclusion
4	Register with null values	null	null	null	null	null	null	Error message	Error Message	Pass
Debug Results	I/flutter (13088): Posts: {username: , password: , fullname: , mobile: , email: , roleid: SingingCharacter.select} I/flutter (13088): Response: {"response":{"status":"error","msg":"Username required"}}									
5	Register without full name	null	0512312333	empty@gmail.com	Empty1	Empty12#	Customer	Error message, full name required	Error message, full name required	Pass
Debug Results	I/flutter (13088): Posts: {username: Empty1, password: Empty12#, fullname: , mobile: 0512312333, email: empty@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"error","msg":"Full name required"}}									
6	Register without phone number	Muhammad Ahmad	null	empty@gmail.com	Empty1	Empty12#	Customer	Error message, mobile required	Error message, mobile required	Pass
Debug Results	I/flutter (13088): Posts: {username: Empty1, password: Empty12#, fullname: Muhammad Ahmad, mobile: , email: empty@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"error","msg":"Mobile required"}}									
7	Register without email	Muhammad Ahmad	0512312333	null	Empty1	Empty12#	Customer	Error message, email required	Error message, email required	Pass
Debug Results	I/flutter (13088): Posts: {username: Empty1, password: Empty12#, fullname: Muhammad Ahmad, mobile: 0512312333, email: , roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"error","msg":"Email required"}}									
8	Register without username	Muhammad Ahmad	0512312333	empty@gmail.com	null	Empty12#	Customer	Error message, full name required	Error message, full name required	Pass
Debug Results	I/flutter (13088): Posts: {username: , password: Empty12#, fullname: Muhammad Ahmad, mobile: 0512312333, email: empty@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"error","msg":"Username required"}}									

Table 6.3: Register Unit Test 3

Test Cases	Scenario	Full Name	Mobile	Email	Username	Password	Role	Expected Result	Actual Result	Conclusion
9	Register without password	Muhammad Ahmad	0512312333	empty@gmail.com	Empty1	null	Customer	Error message, please enter password	Error message, please enter password	Pass
Debug Results	I/flutter (13088): Posts: {username: Empty1, password: , fullname: Muhammad Ahmad, mobile: 0512312333, email: empty@gmail.com, roleid: SingingCharacter.user} I/flutter (13088): Response: {"response":{"status":"error","msg":"Please enter password "}}									
10	Register without role	Muhammad Ahmad	0512312333	empty@gmail.com	Empty1	Empty12#	null	Error message, role required	Error message, role required	Fail
Debug Results	I/flutter (13088): Posts: {username: Empty1, password: Empty112#, fullname: Muhammad Ahmad, mobile: 0512312333, email: empty@gmail.com, roleid: SingingCharacter.select} I/flutter (13088): Response: {"response":{"status":"done","msg":"Account Created"}}									

Table 6.4: Add Restaurant Unit Test

Test Cases	Scenario	Restaurant Name	Email	Phone	Address	Restaurant Image	Expected Result	Actual Result	Conclusion
1	Register new restaurant successfully	Food Restaurant	restaurant@gmail.com	0512345678	Rose street	Png image	Restaurant created	Restaurant created but the application crashed	Fail
Debug Results	I/flutter (22240): Posts: {act: saveRest, restname: Food Restaurant, email: restaurant@gmail.com, phone: 0512345678, address: Rose street, restimg: http://192.168.1.10/tabletime/restimgs/1_1713640699.png, userid: 1} I/flutter (22240): Response: {"response":{"status":"done","msg":"data loaded","rows":[{"restid":"1","ownerid":"1","restname":"xx","restemail":"sss","restphone":"3434","descr":"xas","address":"asdas","restpic":null},{"restid":"6","ownerid":"1","restname":"The Rice Bowl","restemail":"Rice@gmail.com","restphone":"0512312350","descr":null,"address":"Ball street","restpic":"http:\/\/172.20.10.2\/tabletime\/restimgs\/1_1713424726.png"},{"restid":"8","ownerid":"1","restname":"Food Restaurant","restemail":"restaurant@gmail.com","restphone":"0512345678","descr":null,"address":"Rose street","restpic":"http:\/\/192.168.1.10\/tabletime\/restimgs\/1_1713640699.png"},{"restid":"7","ownerid":"1","restname":"","restemail":"","restphone":"","descr":null,"address":"","restpic":""}]}}								
2	Register new restaurant with null values	null	null	null	null	null	Error message	Application crashed but it did create a new restaurant with null values	Fail
Debug Results	I/flutter (13088): Posts: {act: saveRest, restname: , email: , phone: , address: , restimg: , userid: 1} I/flutter (13088): Response: {"response":{"status":"done","msg":"data loaded","rows":[{"restid":"1","ownerid":"1","restname":"xx","restemail":"sss","restphone":"3434","descr":"xas","address":"asdas","restpic":null},{"restid":"6","ownerid":"1","restname":"The Rice Bowl","restemail":"Rice@gmail.com","restphone":"0512312350","descr":null,"address":"Ball street","restpic":"http:\/\/172.20.10.2\/tabletime\/restimgs\/1_1713424726.png"},{"restid":"7","ownerid":"1","restname":"","restemail":"","restphone":"","descr":null,"address":"","restpic":""}]}}								

6.2 Black-box Testing

The term "black box" refers to “a software testing methodology in which the tester analyzes the functionality of an application without a thorough knowledge of its internal design”[11]. The tables below represent the various scenarios the user could go through, as well as the results of the testing.

Table 6.5: Register Acceptance Test Part 1

Test cases	Scenario	Full name	Mobile	Email	Username	Password	Role	Expected Result	Actual Result
1	Entering number as full name	12345	0551234566	sami213@gmail.com	Sami1	Qasw!321	Customer	Register completed.	Fail
Test Screenshot									
2	Entering mobile number less 10 digits	Sami Ali	055123456	sami213@gmail.com	Sami1	Qasw!321	Customer	Error message “Mobile number must be 10 digits”.	Pass
Test Screenshot									

Table 6.6: Register Acceptance Test Part 2

Test cases	Scenario	Full name	Mobile	Email	Username	Password	Role	Expected Result	Actual Result
3	Entering email without '@' symbol	Sami Ali	0551234566	sami213gmail.com	Sami1	Qasw!321	Customer	Error message "Valid Email is required".	Pass
Test Screenshot									
4	Entering already existing username	Salman Said	0597100992	salman.saeed@gmail.com	Salman110	Ar@12345	Customer	Error message "Username already exist".	Pass
Test Screenshot									
5	Entering password without special characters	Salman Said	0597100992	salman.saeed@gmail.com	Salman110	Ar12345	Customer	Error message "You must contain at least 1 special character !"	Pass
Test Screenshot									

Table 6.7: Login Acceptance Test Part 1

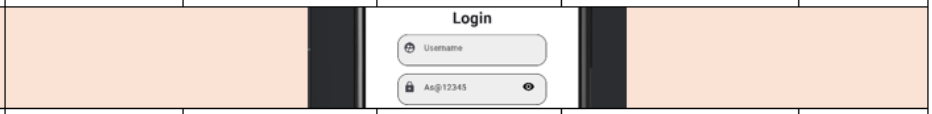
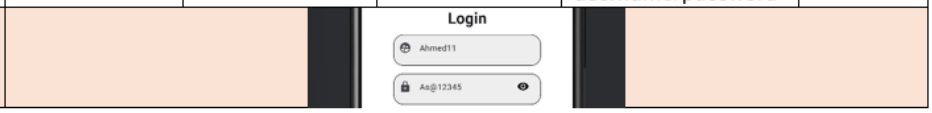
Test cases	Scenario	Username	Password	Expected Result	Actual Result
1	Entering empty username	-	As@12345	Error message indicating empty username.	Pass
Test Screenshot					
2	Entering empty password	Ahmed12	-	Error message indicating empty password.	Pass
Test Screenshot					
3	Entering invalid username	Ahmed11	“As@12345”	Error message “Faild invalid username/password”	Pass
Test Screenshot					

Table 6.8: Login Acceptance Test Part 2

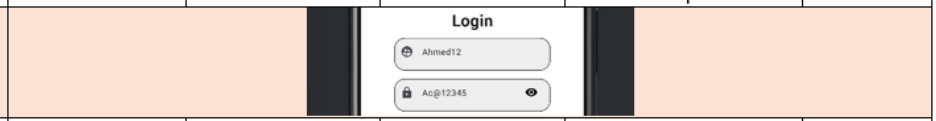
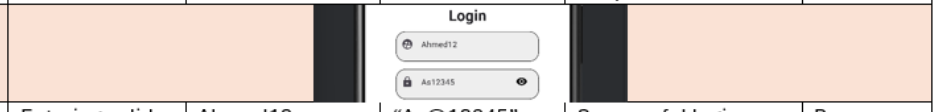
Test cases	Scenario	Username	Password	Expected Result	Actual Result
4	Entering invalid password	Ahmed12	“Ac@12345”	Error message “Faild invalid username/password”	Pass
Test Screenshot					
5	Entering password without special characters	Ahmed12	“As12345”	Error message indicating missing special characters in the password	Fail
Test Screenshot					
6	Entering valid username and password	Ahmed12	“As@12345”	Successful login	Pass
Test Screenshot					

Table 6.9: Add New Restaurant Acceptance Test Part 1

Test Cases	Scenario	Restaurant Name	Email	Phone Number	Location	Image	Expected Result	Actual Result
1	Entering valid restaurant information	Half million 1/2	halfmillion@gmail.com	0120092348	An Nasim District, JBNA3060	Valid	Restaurant added successfully	Fail
Test Screenshot								
2	Entering invalid email	Half million 1/2	halfmilliongmail.com	0120092348	An Nasim District, JBNA3060	Valid	Error message "Valid Email is required".	Fail
Test Screenshot								

Table 6.10: Add New Restaurant Acceptance Test Part 2

Test Cases	Scenario	Restaurant Name	Email	Phone Number	Location	Image	Expected Result	Actual Result
3	Leaving a required field empty	-	halfmillion@gmail.com	0120092348	An Nasim District, JBNA3060	Valid	Error message to fill in all required fields	Pass
Test Screenshot								
4	Adding a restaurant already existing	Half million 1/2	halfmillion@gmail.com	0120092348	Al-Khalidiya Jeddah, 2604-2716	Valid	Error message indicating restaurant already exists	Pass
Test Screenshot								
5	Canceling restaurant addition	Half million 1/2	halfmillion@gmail.com	0120092348	An Nasim District, JBNA3060	Valid	No changes in the database	Pass

Table 6.11: Add New Table Acceptance Test Part 1

Test Cases	Scenario	NFC Serial Number	Table Number	Number of Chairs	Expected Result	Actual Result
1	Entering valid table information.	4,30,117,124,185,42,129	1	4	Table added successfully.	Pass
Test Screenshot						
2	Entering a duplicate NFC serial number.	4,30,117,124,185,42,129	2	2	Error message NFC already taken.	Pass
Test Screenshot						
3	Entering a negative table number	2,11,115,124,185,42,130	-2	2	Error message indicating invalid table number.	Fail
Test Screenshot						
4	Entering zero chairs	2,11,115,124,185,42,130	3	0	Error message indicating number of chairs cannot be zero.	Fail
Test Screenshot						

Table 6.12: Add New Table Acceptance Test Part 2

Test Cases	Scenario	NFC Serial Number	Table Number	Number of Chairs	Expected Result	Actual Result
5	Leaving NFC serial number field empty	-	4	6	Error message indicating NFC serial number is required.	Fail
Test Screenshot						
6	Leaving table number field empty	12,8,213,124,185,42,130	-	2	Error message indicating table number is required.	Pass
Test Screenshot						
7	Leaving number of chairs field empty	12,8,213,124,185,42,130	5	-	Error message indicating number of chairs is required.	Pass
Test Screenshot						
8	Adding a table with a large NFC serial number	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17, 18,19,20,21,22,23,24,25,26,27,28,29,30, 31,32,33,34,35,36,37,38,39,40,	6	2	Error message indicating NFC serial number is too large.	Fail
Test Screenshot						

Table 6.13: Update Profile Acceptance Test Part 1

Test Cases	Scenario	Full Name	Mobile Number	Email	Username	Password	Expected Result	Actual Result
1	Update profile with valid data.	Sami Ail	0551234566	sami213@gmail.com	Sami_123	Asd@1234	Profile updated successfully and user directed to confirm page.	Pass
Test Screenshot								
2	Update profile with invalid email	Sami Ail	0551234	Sami213gmail.com	Sami_123	Asd@1234	Error message indicating invalid data and profile not updated.	Pass
Test Screenshot								
3	Cancel from profile update page	-	-	-	-	-	Go directly to the home page after the cancellation.	Pass

Table 6.14: Update Profile Acceptance Test Part 2

Test Cases	Scenario	Full Name	Mobile Number	Email	Username	Password	Expected Result	Actual Result
4	Update profile without entering Full Name	Sami	0551234566	sami213@gmail.com	Sami_123	Asd@1234	Error message indicating Full Name is required.	Fail
Test Screenshot								
5	Update profile without entering Mobile Number	Sami Ail	0551234566	sami213@gmail.com	Sami_123	-	Profile updated successfully and user directed to confirm page.	Pass
Test Screenshot								

Chapter 7

Conclusion

7.1 Problems and Difficulties

One of the biggest challenges in developing the application is a lack of online resources connected to the Table Time application concept, as there has been no application that uses NFC technology for restaurant table reservations.

Another challenge is creating a dynamic restaurant interior map from the top perspective to manage all of the table placement to reflect various restaurant interior layouts. Creating a dynamic restaurant interior gives customers an overview of how the restaurant layout will look, which is why it is crucial to implement.

The survey had a low participation rate among restaurant owners since it was performed by personally visiting them, explaining the survey's objective, and displaying our application before collecting their responses.

Finally, implementing a payment system is problematic since it is necessary to have an end-user license agreement and a privacy policy before implementing a payment system. End-user license agreements and privacy policies must be established by professionals to follow the government's regulations.

7.2 Findings

Many potential customers are interested in a restaurant reservation application, but restaurant owners have some worries. According to the survey results, restaurant owners are not opposed to adding a reservation application to their reservation system if the issues are addressed in our application. One of the primary challenges is dealing with impolite customers who do not arrive at their reservation time, initiate conflicts with restaurant workers, and then leave poor reviews.

To address this issue, we must prioritize customer support, end-user license agreements, and privacy rules, which will aid in protecting the rights of both customers and restaurant owners.

7.3 Future Work

Table Time's mission is to provide a new way of managing restaurant reservations that is easier and more effective than other reservation software. To accomplish that ambition, the upcoming development will center on polishing and publishing the Table Time application.

To improve the Table Time app, the table list will be updated with more additions and details to reflect different restaurant interiors. Before launching Table Time, the database will be uploaded to the cloud, and the End User License Agreement and Privacy Policy will be written by professionals in accordance with government regulations.

Due to the short time available for conducting a survey of restaurant owners, the number of survey participants was low; following larger surveys across several restaurant types will be necessary for comprehensive feedback, and we plan to gather these surveys physically, as we did with the previous survey.

The Table Time system will then be integrated into Foodics to function on cashier devices before being published on Google Play and the App Store.

7.4 Conclusion

Restaurant reservations are considered an essential service in any restaurant. Traditional restaurant reservations are no longer viable. The Table Time application was developed using NFC technology to make restaurant reservations more efficient while offering a pleasant experience for customers. Positive survey responses demonstrate substantial consumer support for the Table Time idea. Table Time will be further developed and refined before being released on Google Play and App Store

Bibliography

[1] A. Dennis, B. H. Wixom, and D. Tegarden, *Systems Analysis and Design with UML, 5th Edition*. Wiley, 2015.

[2] Thamer Alomirini, "Skipro," https://skipro.app/, 2023, version 5.0.0, Accessed on 10/08/2023.

[3] Y. Bunashi, "Requeue," https://requeue.net/, 2023, version 3.2.8, Accessed on 10/08/2023.

[4] H. S. Company, "Wddk," https://wddk.sa/, 2023, version 2.6, Accessed in 10/08/2023.

[5] M. Algadhibi, "My table," https://mytable.sa/, 2023, version 6.7.8, Accessed 10/08/2023.

[6] L. Rosencrance, "Software requirements specification (srs)," *Software Quality*, September 10 2019. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/software-requirements-specification

[7] AltexSoft. (2021, July) Functional and non-functional requirements: Specification and types. Accessed: November15, 2023. [Online]. Available: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/

[8] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.

[9] QAT Global, "Detailing the external interfaces in the software requirements," https://qat.com/external-interfacessoftware-requirements/, 2023, published: September 25, 2023, Accessed: November 15, 2023.

[10] Imperva. (2023, December) What is white box testing: Types & techniques for code coverage. Learning Center. [Online]. Available: https://www.imperva.com/learn/application-security/white-box-testing/

[11] H. Ashtari. (2022, September) Black box testing vs. white box testing. Spiceworks Inc. [Online]. Available: https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/