Advances in Computing 2023, 13(1): 9-36
DOI: 10.5923/j.ac.20231301.02

Takeaway Menu System for Online Meal Delivery Services

Alanoud A. Alqahtani

MSc Cloud Computing, School of Computing and Mathematical Sciences, University of Leicester, England

Abstract Online meal delivery services have grown in popularity as they allow individuals to order food from the comfort of their own homes. COVID has limited the number of offline businesses, closing many physical stores. The online food delivery market is flooded with plenty of options, making it difficult for consumers to decide which one to use. B2B platforms have become increasingly popular, solving the lockout problem and presenting other challenges. The Takeaway Menu system is a web app that allows family-run businesses to advertise their services on the platform and expose them to a large audience. Similarly, consumers have a rich platform from which they can purchase without worrying about the kitchen's quality and hygiene because the system ensures that only licensed suppliers are registered. The project features a micro-service design, with the back-end written in Java Spring-boot and Node.js and the client-side written in React.js. Asynchronous messaging is handled by RabbitMQ,while Zipkin handles service tracking. This paper begins by explaining the project's necessity and motivation. Then it goes over the needs and specifications, which are prioritised. Next is the planning section, where the technology stack chosen for the project is briefly detailed, along with specifics of the process model used, database design, and software modeling. After that, the application's development and testing are discussed. Finally, the paper concludes with information regarding the assessment procedure, performance metrics, and recommendations based on the evaluation results.

Keywords Online Meal Delivery Services

1. Introduction

E-commerce is not a novel idea. In truth, e-commerce has been around for almost four decades. Amazon, one of the pioneers of e-commerce, began as an online book retailer. Amazon is now the world's largest e-commerce organisation. People purchase everything from electrical devices to veggies and groceries online from the convenience of their own homes (Kim, 2021). This was not the case not long ago. In 2019, due to lockdown methods to counter the COVID pandemic, people could not access their offices, workshops, and booths. Private, family-run businesses were closingdown because they had no way of doing business, which meant many people could not meet their fundamental financial needs.

Consequently, online B2B platforms grew popular because they provided a platform from which people could conduct their businesses, despite the COVID restrictions (Kumar et al., 2021). As a result, more and more productive families shifted their businesses to such platforms, and there wasa significant increase in online transaction patterns, as seen in Figure 1 below.

According to Edinson Trends (2022), internet purchases in the United Kingdom decreased in March and April 2020 before increasing by 39% in May. When comparing April 2021 to April 2020 figures, UK transactions increased by 171% (Edinson Trends, 2022). However, as impressive as these figures appear, many suppliers are dissatisfied with these platforms, primarily due to the tight restrictions and high commission rates, which primarily impact small, family-run businesses operating from their homes, who lack negotiating power with the platforms (Ozili & Arun, 2022). Conventional businesses, where customers would visit physical stores to purchase, were highly profitable before online B2B platforms emerged. Still, as these online B2B platforms developed their dominance in the market, they tilted the scales in their favour, resulting in very little profit for conventional food business owners (Ozili & Arun, 2022). As a result, our project aims to offer solutions to these challenges so hardworking, family-run businesses may benefit from increased profits and business growth.

1.1. Background

As previously stated, the meal delivery sector is quickly expanding. As of 2022, the food industry's market value, worldwide, is $215.69 billion (Njunina, 2022). With such rapid expansion, many individuals are trying to break into the

* Corresponding author:
zmsx1@hotmail.com (Alanoud A. Alqahtani)
Received: Feb. 21, 2023; Accepted: Mar. 8, 2023; Published: Mar. 15, 2023
Published online at http://journal.sapub.org/ac

food sector. Although the fast expansion of the food delivery sector is beneficial in terms of employment creation, it has also given rise to several essential concerns. For example, due to many service providers being on board, platforms have been overburdened and unable to fully satisfy service providers (Poon & Tung, 2022). Also, platform providers' lack of or inability to monitor the food production process means that food preparationin unsanitary conditions could be an issue (Poon & Tung, 2022).

Furthermore, due to the increased pressure caused by the increasing number of customers, users have experienced poor customer service in the form of delayed replies, unresponsive websites, service outages, etc. (Poon & Tung, 2022). This dissertation was inspired by these challenges and seeks to fix or reduce the problems to deliver a better experience.

1.2. Aim and Objectives

This project aims to bring consumers and providers together on a single platform where providers can promote their services and customers can quickly access them. The web app is designed to be user-friendly for all ages. The program will provide a platform for family-run businesses to sell their meals to many clients while effortlessly maintaining order inflow and delivery quality.

1.2.1. The Aim

A country's economy relies heavily on small enterprises, in-house workshops, and other forms of entrepreneurship. Most countries worldwide were utterly unprepared when COVID arrived (Ozili & Arun, 2022). The government had no strategy or idea of how to keep these small industries viable, and neither did the individuals who worked in them (Ozili & Arun, 2022). As a result, businesses suffered significantly. To be prepared for any future epidemic or pandemic, it is vital to have tools that will keep businesses afloat and be a route to growth. This project established a B2B platform to provide such a solution. This platform will be handy not only now but also in theevent of a similar future event, such as COVID.

Furthermore, this project seeks to provide efficient and optimised solutions to the previously mentioned challenges by introducing features such as a maximum serving capacity handler, which ensures that providers do not receive more orders than they can serve while also ensuring that the customer experience is not degraded. There is also a feedback mechanism that collects client comments on individual products of the order. The feedback incorporates information regarding the meal products and the delivery experience. These comments regarding food products are presented underneath the appropriate item on the menu to assist customers in making informed decisions and to provide a positive customer experience.

1.2.2. Objectives

To accomplish the objectives mentioned earlier, the project will produce a web app allowing users to access operations based on their client, provider, or administrator role. The following are the deliverables:

- Development of an interactive web application that will empower customers by giving them control over the process of order placement and providing them extra information tomake information-based decision.

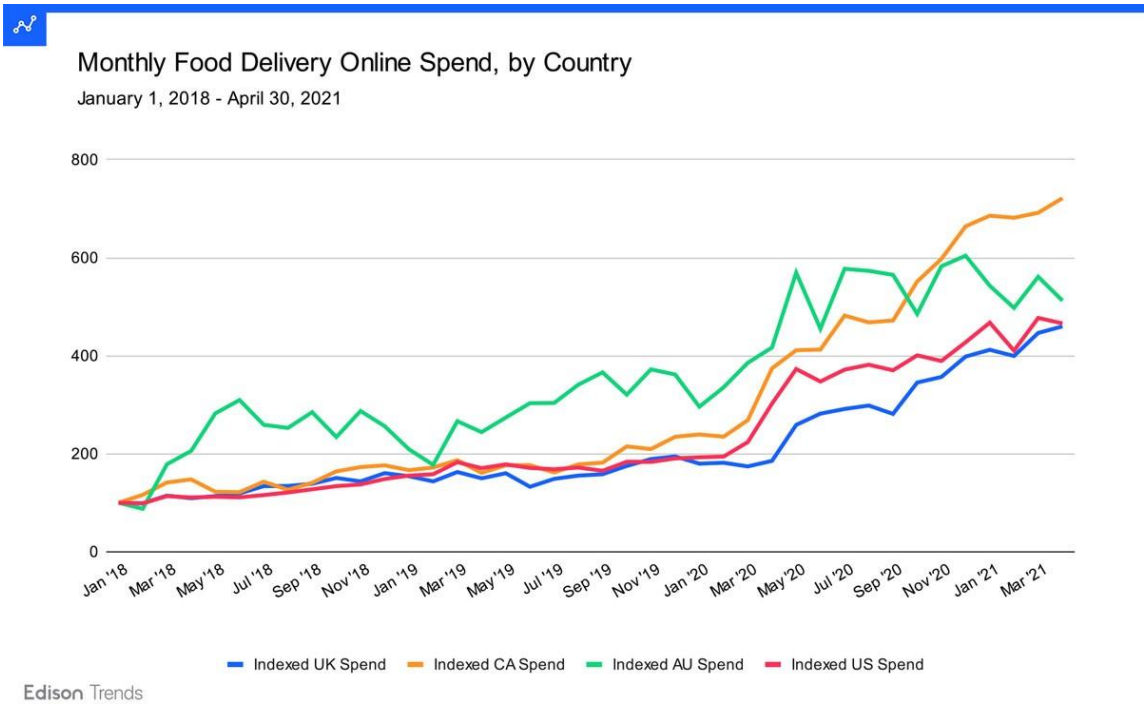


Figure 1. Monthly Food Delivery Online Spend by Country (Edinson Trends, 2022)

- Development of an interactive web application that will enable productive families to reach their target audience and showcase their services in a systematic way.

2. Background Literature

This chapter discusses the background reading used to inform our decisions regarding the project's architecture, design, and technology.

2.1. Monolithic

Most programming languages offer some form of functionality to split a complex project into manageable modules. Still, they always produce a single executable unit called a monolith that runs on the same system sharing resources such as CPU, RAM, Database, and Storage. As a result, it is challenging to run separate components of a monolith independently (Blinowski et al., 2022). Managing a monolith gets increasingly challenging as its size and complexity grow due to the following reasons:

- Even for a small change, the whole monolith needs to be restarted, which takes considerable effort and time (Kalske et al., 2018).
- The code becomes highly complex and tangled no matter how well it is managed (Blinowski et al., 2022).
- It becomes increasingly difficult to debug such applications, and the chance of introducing new bugs when fixing the previous ones becomes very high (Blinowski et al., 2022).
- As explained above, a monolith is a single executable unit and needs to be deployed on the same machine; this introduces poor utilisation of the resources where either an expensive configuration is installed for some modules or some compromise is made by chosen sub-optimal configurations (Kalske et al., 2018).
- Even if the load is increased on a single monolith module, the only way to scale the application is to create a new instance and split the load by introducing a load balancer. This limits scalability (Blinowski et al., 2022).

A monolith application requires all modules to use the same technology stack and further locks the stack for future developments (Blinowski et al., 2022).

2.2. Micro-Services

A micro-service is a small service that fulfills a single business responsibility (Kalske et al., 2018). A micro-service architecture utilises many such services to perform a substantial business task (Kalske et al., 2018). A micro-service should be autonomous and independent, having a technology stack, data, and deployment procedures. It should follow the single responsibility principle. Although the size of a micro-service is still a topic of discussion, it is generally accepted that fine- grained micro-services are preferable to larger ones (Kalske et al., 2018). Following are some keycharacteristics of a micro-service:

- The name 'micro-service' suggests that when a service becomes large and complex it should be decomposed into two or more services because it is far easier to maintain andscale a small service (Kalske et al., 2018).
- A micro-service should be loosely coupled, meaning it should be independent. The communication can only be done through the exposed simplifies or upgrades a micro- service easily, without affecting other services (Kalske et al., 2018).
- Micro-services enable continuous development and delivery as they enable many small services to be independently deployable without affecting any other service (Taibi et al., 2018).

Micro-services are becoming the new norm of software development, but they come with their challenges, some of which are highlighted below:

- One of the biggest challenges when moving toward micro-service architecture is to decompose and refactor the monolith into several micro-services that require considerable effort and expertise (Kalske et al., 2018).
- Micro-service architecture prefers separate databases for separate services, which introduces data consistency problems that require novel strategies to provide correct and updated data to all services (Blinowski et al., 2022).
- Since API calls over the network replace the method invocation in a monolith, developers should consider the design of a micro-service architecture to minimise the API calls to other services (Blinowski et al., 2022).

2.3. Eureka Server

In a micro-service architecture, several independent services are up and running (Kalske et al., 2018). For availability purposes, there might be several instances of the same service up and running, and it can be complicated to keep track of each instance's port so that a successful request can be made (Kalske et al., 2018). This is where Eureka-server comes in. It is an application that allows the instances to register themselves to Eureka-server to keep the information about the I.P. address and port of the service instance. It is also known as a Discovery Server as it helps in servicediscovery (Kalske et al., 2018).

2.4. Distributed Tracing and Zipkin

In micro-services, the process of debugging is complicated. Tracing only a single request is daunting as developers must trace it across multiple services. This is where Distributed Tracing comes in, as it identifies the exact point or points that are causing problems ("Architecture · OpenZipkin," 2022). Zipkin is one of the most widely used distributed tracing system tools, developed using Google paper by Twitter ("Architecture OpenZipkin", 2022). Zipkin consists of four components, namely Zipkin collector, storage, Zipkin query service, and web U.I., responsiblefor collecting tracing data, storing the traced data, providing API for fetching the stored data, and providing an

easy method for displaying fetched data on the user interface respectively ("Architecture · OpenZipkin", 2022). So, to shield ourselves from future failures, we need to have data available to us. For this purpose, we have incorporated Zipkin in the project as it is more suitable for website applications ("Architecture ·OpenZipkin", 2022).

2.5. Docker

To run an application, we need the runtime environment of an application, all the dependencies, libraries, configuration files, and all other binaries. If any of these are missing a file, there are deployment issues, and the application will not run. So, a container is something that bundles all these files in a single package to ease the process of deployment (Jangla, 2019). Moreover, the performance of a docker container is much better than virtual machines in terms of memory throughput, CPU performance, operation speed measurement, load test, and Disk I/O operations (Potdar et al., 2020). With Docker, installation and configuration of issues of dependencies like PostgreSQL, RabbitMQ server, and Zipkin were taken care of by using images provided on the docker site.

2.6. Messaging Queue and RabbitMQ

Messaging queues help store data and share data between different producers and consumers. They are super helpful in handling data streams without compromising response time, as the producer, after sending a message, does not wait for the reply and moves on to produce the following message. Also, they are very effective in asynchronous processing (Fu et al., 2021).

RabbitMQ is an open-source message broker system that allows messages to be sent over TCP connections (Sharvari & Sowmya, 2019). RabbitMQ provides out-of-the-box support for several standardised protocols like delay and priority queuing (Fu et al., 2021). The most important feature or functionality of RabbitMQ is its support for the Advanced Messaging Queue Protocol (AMQP). It also provides options for exchange types depending on our needs (Madhu & Sunanda, 2019). Our project uses it for asynchronous communications between notification services and other services.

2.7. Relevant Applications

E-commerce is not new and has been here for more than four decades; however, e-commerce has primarily been used by business giants. Smaller businesses were not as focused on having an onlinepresence and did not have a budget to have a dedicated I.T. team to develop and maintain their application. As COVID hit the world, people were forced to stay at home and keep a safe distance. This situation gave rise to the popularity of online business platforms. This section focuses on the need for online business platforms and their importance. It also discusses the existing systems and their shortcomings, which inspired this project.

2.7.1. Uber Eats

Uber Eats is one of the world's largest online food delivery platforms. It was launched in August 2014 by its parent company Uber. Uber Eats was initially launched with the name Uber Fresh, but later, in 2015, it was renamed Uber Eats and launched its order application separately from the riding application of Uber (Carson, 2022). As of February 2019, Uber Eats was on its way to delivering food for an estimated $10 billion worldwide (Carson, 2022). As Figure 2 shows, since 2016, the food delivery market share of Uber Eats has grown from 5% to 25%.

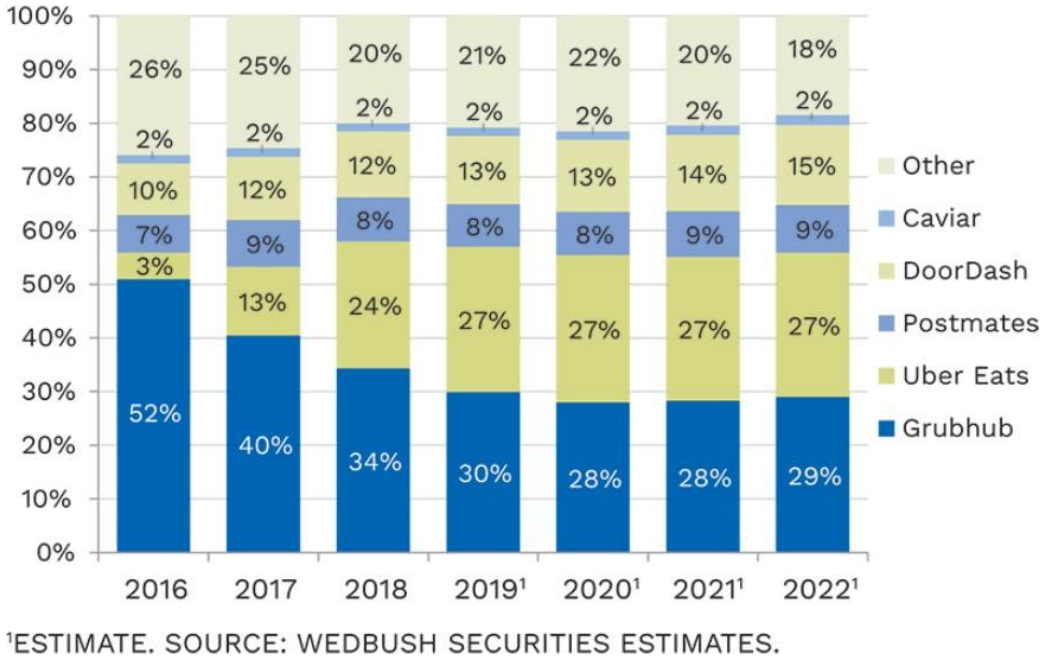


Figure 2. Market share for different food delivery services (Carson, 2022)

2.7.2. Other Food Delivery Service Applications

The other four main delivery service applications are Postmates, DoorDash, Seamless, and Grubhub (Carson, 2022). Each food delivery service has its strengths and weaknesses, so consumers need to consider which one would be the best fit for their needs before signing up.

Postmates

Postmates delivers merchandise using its human dispatch network infrastructure, allowing users to purchase items from any place at any given time and location (Ridesharing Driver, 2021). Postmates' couriers, like Uber drivers, receive order notifications on their handsets, and the corporation uses GPS to match demand and supply as quickly as possible (Ridesharing Driver, 2021). The primary distinction is that Postmates delivers from almost any eatery or store, whereas Uber Eats exclusively delivers from associated eateries (Ridesharing Driver, 2021). Although the prices are comparable, Postmates delivers better deals to its subscribers than Uber Eats.

DoorDash

Most delivery service systems, in principle, offer a relatively user-friendly interface. DoorDash, on the other hand, offers a broader selection of food categories and more comprehensive sifting on factors like pricing, duration of delivery, and more (Kamp, 2020). Furthermore, the company actively integrates technological innovations and exploits consumer analytics to deliver merit to its associate eateries, resulting in their participation in the network (Kamp, 2020).

Grubhub

Grubhub's corporate strategy is centred on billing eateries a percentage for each order made (Bushnell, 2022). Grubhub's system offers distribution solutions to establishments in various market segments. Grubhub has a broader network of countrywide affiliates than DoorDash, enabling them to provide better competitive prices (Synder, 2020). Grubhub also provides a broader array of deals, with various incentives for discounted shipping available on the firm's Perks Webpage (Synder, 2020). As a competitive edge, GrubHub has incorporated a ranking system that grades eateries based on their ratings (Synder, 2020). Users assess eateries based on the meals' calibre and overall service.

Seamless

Although they are different firms, GrubHub and Seamless are nearly identical (Hyrecar, 2021). The primary distinction is that clients may visit an individual entity's trademarked application and website (Hyrecar, 2021). However, for the delivery team that supports the supply chain, the distinctions vanish. Drivers often enroll to drive for GrubHub and Seamless because the two firms offer similar services (Hyrecar, 2021). Furthermore, if an eatery is accessible on Seamless, it is similarly offered on GrubHub. Therefore, operations and services are identical (Hyrecar, 2021).

2.7.3. Comparison

Comparing the four food delivery service applications, DoorDash has the most comprehensive menu, with more than 350 restaurants in its database, while Grubhub has the widest variety of cuisines, with over 20,000 restaurants represented (Rayome, 2022). However, Seamless offers the best customer ratings, with a satisfaction rate of 98% (Rayome, 2022). Finally, Postmates has the lowest delivery fee of all the food delivery service applications, reviewed at $2.75 for orders within 1 mile of the delivery address (Carson, 2022). Based on these factors, DoorDash appears to be the best food delivery service application. However, all the applications offer great value for money and can be used to order anything from food to home goods.

While Uber Eats is growing exponentially, there are some significant issues caused by Uber Eats policy. For instance, since 2016, Uber Eats has enforced, through their terms, the right to deduct the total charge of an order from a restaurant when there is a problem with the meal, even when the restaurant is not at fault ("Uber Eats amends its contracts", 2022). As of March 2022, a groupof diners has filed a lawsuit against Grubhub, Uber Eats, and Postmates over U.S. restaurant prices. The diners claim that the 'no-price competition' policy has prohibited restaurants from charging lower prices for dine-in or takeout orders (Stempel, 2022). Also, smaller restaurants have not been able to enjoy growth and progress in terms of profit due to the higher commission of the platform(Li et al., 2020).

The online food industry has generated many employment opportunities for professionals, including cooks, delivery riders, supporting staff, software developers, etc. The industry giants, such as Eleme in China, Swiggy in India, and the US-based Uber Eats, employ thousands of employees (Li et al., 2020). Although these jobs are suitable for the economy and help low-income people support their families, poor working conditions are an issue. For example, delivery riders experience high workloads, traffic danger, and unrealistic delivery deadlines. Due to these conditions, there is often low job satisfaction and high attrition rates (Li et al., 2020).

2.8. Observations and Proposed Solution

This chapter carefully highlighted the importance of online food delivery platforms due to the immense amount of revenue they are generating. Despite this revenue generation, significant issues must be addressed, given the importance of this industry. The major issues identified here can be hostile policies and bad customer experiences. To solve bad customer experiences, several features have been introduced to enhance good customer experience, such as maximum serving capacity, which lets the provider specify how many orders it can handle efficiently at a time. The customer is informed that the restaurant is temporarily unavailable if the queue is full. The customer can either try later or schedule

their order. This feature will ensure that there are no delayed or late deliveries and customers do not have to wait for longer than they were informed it would take to prepare the order. Our app will also show the ingredients information from the provider, categorising them into required and *optional*. The customer is shown the ingredients list before placing the order, and if they are allergic to any ingredient, they can opt to exclude it fromthe order if it is in the *optional* category. If it is a *must*, then the customer cannot exclude it as it will affect the taste of the dish, and providers do not want bad reviews about their dishes. Moreover, intelligent feedback takes reviews and ratings for each item, not the overall order. This feedback will help other customers make more informed buying decisions.

3. Software Requirements Specification (SRS)

Software requirement specifications (SRS) is a document that aims to discover, explore, and document the project's nature, purpose, functional and non-functional requirements, scope, and hardware and software requirements. This document serves as a manual for the project to be developed and helps keep all the stakeholders and teams on the same page.

3.1. User Requirements

User requirements specify what the user wants the application to do and how the application should behave. Below are the user requirements categorised as essential, recommended, and optional, indicating their priority.

3.1.1. Essential Requirements

Essential requirements are requirements that must be met for the system to be in a functional state. If even a single essential requirement is not implemented, the system will not be able to serve its purpose. Below are the essential requirements of this project:

Client Interface

1. Ability to register using the registration portal.
2. Ability to authenticate the user and perform the login.
3. Get all the provider's information and display it meaningfully to place an order.
4. Perform transactions on the placement of orders.
5. Order tracking for both customer and provider.
6. Search filter for a customised search query.
7. Enforcement of feedback from customers about restaurant and food items
8. Display of feedback by customers along with restaurant and food items.

Provider Interface

1. Ability to create menu.
2. Ability to view and edit menu.
3. Ability to receive the order.
4. Dashboard to view placed orders.
5. Dashboard to view statistics
6. Search filter for a customised search query.
7. Dashboard to view feedback from customers.

3.1.2. Recommended Requirements

Recommended requirements are requirements that enhance the application functionality.The recommended requirements for this project are stated below:

Client Interface

1. Show recommendations to a user based on the latest trends.
2. Show recommendations to a user based on user order history.
3. The facility of the system's digital wallet.
4. Show recommendations to users of top-rated providers and those with similar taste profiles.
5. Promotions and voucher integrations.

3.1.3. Optional Requirements

Optional requirements are not necessary for the system's end goal but add to or enhance the system's productivity. Below are the optional requirements of this project:

1. Support for multiple online transaction methods.
2. Mobile applications for Android and iOS.
3. Ability to generate a report to view performance based on a defined period.

4. Methodology

PLANNING AND DESIGN

When starting a software engineering project, most engineers often leave out some of the most critical steps in the software engineering life cycle in their urge to start developing the project. These processes are the cornerstones of scalable, maintainable, and future-proof work. These phases are planning and designing. The design represents a higher model of abstraction that includes all the research related to technology stacks, architectural design, and different types of diagrams. Every design choice will impact the development phase, so every decision made here should be based on correct reasoning instead of naively adopting the prevalent standards of the industry.

Planning is another crucial phase of the life cycle that outputs a series of modular tasks and a realistic timeline that is followed throughout the process.

4.1. Modeling Tools

During the design phase, many tools used to refine and polish the ideas concretely on paper. These tools help document the whole design and planning process and provide a better understanding of the entire process ahead of starting. The following tools were used in the design phase of this project:

- **Lucid-Char:** a web-based diagramming application to create different UML diagrams, database designs, Etc.
- **Draw.io:** a web-based application for creating UML diagrams, ERD, database design, Etc.

It was used to create an architecture diagram in our case.

4.2. Software Modelling

4.2.1. Use Case Diagram

A use case diagram can help facilitate understanding the different operations a user can perform and how different users interact with the system and each other. Figure 3, below, depicts all the micro-services and how they interact with each other and the user.

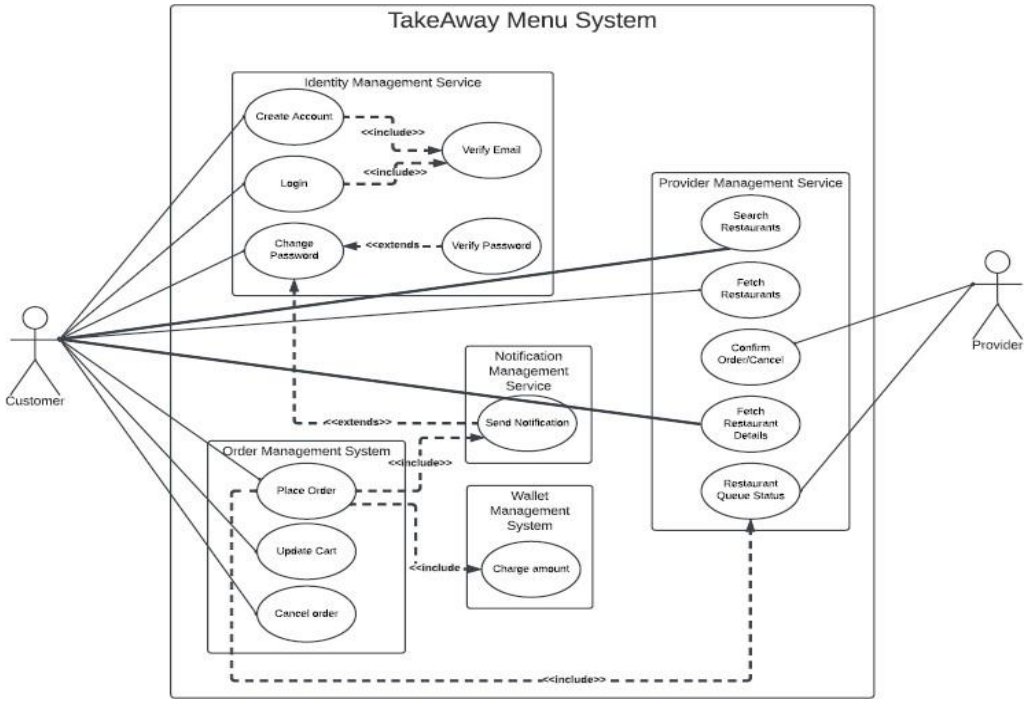


Figure 3. TWM use case diagram

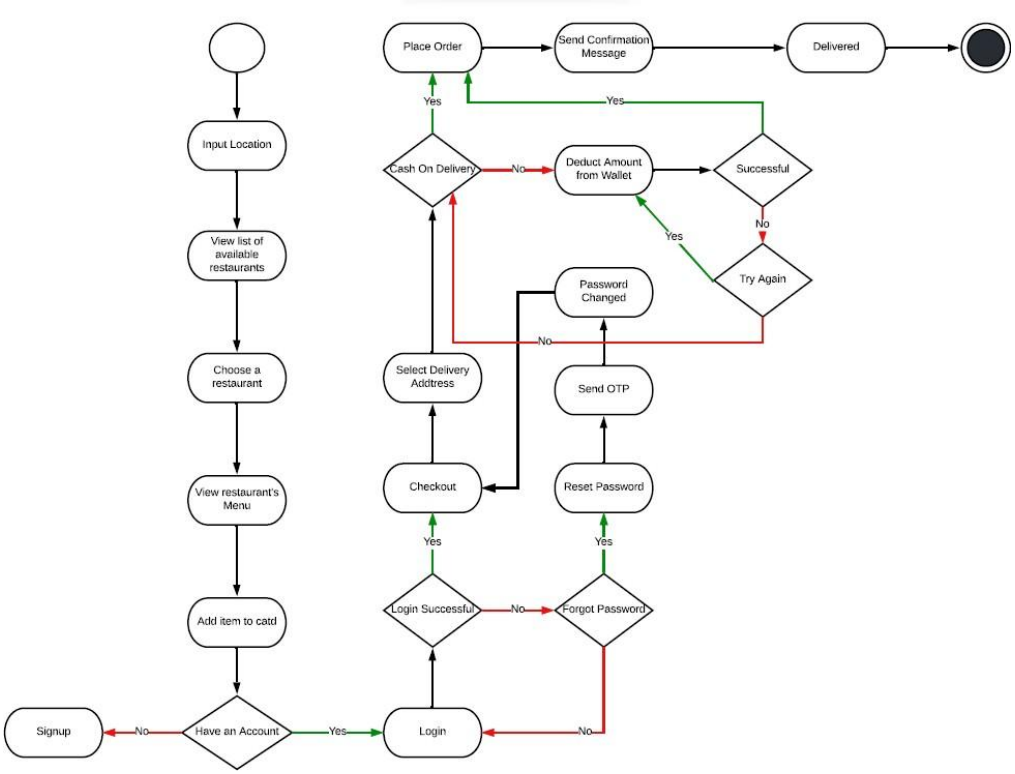


Figure 4. Activity diagram for Client Interface

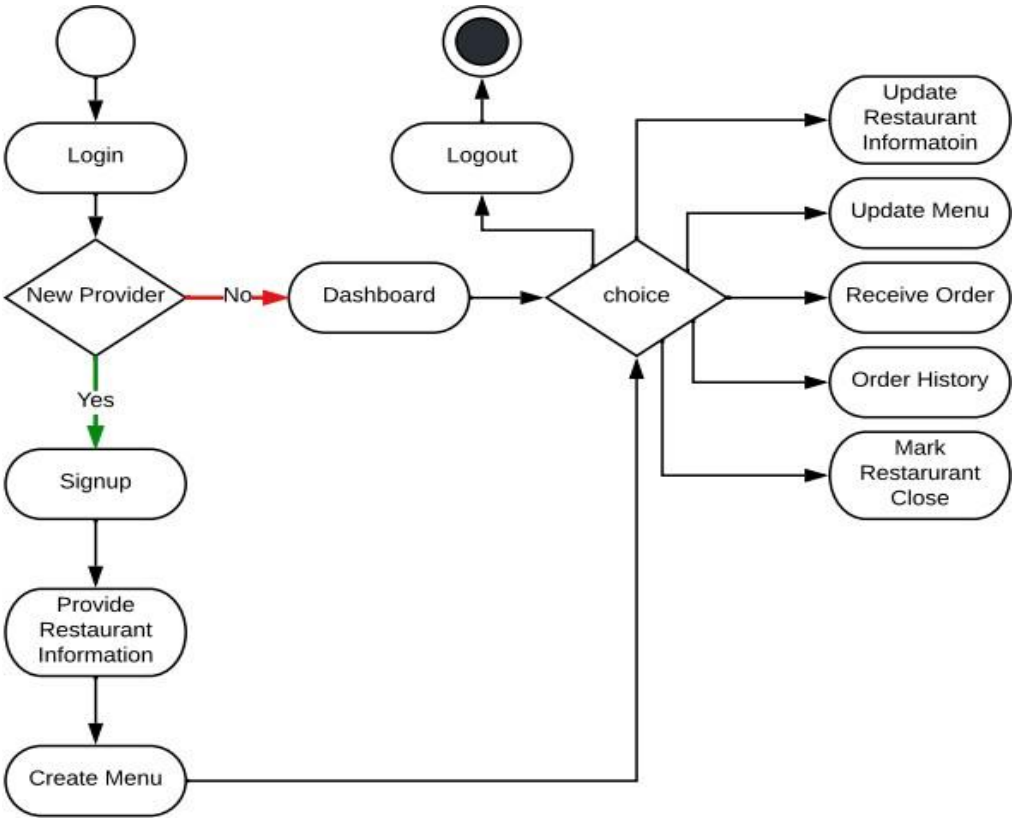


Figure 5. Activity diagram for Provider Interface

4.2.2. Activity Diagram

Users perform a sequence of operations to get to a particular state when they are using the application. Figures 4 and 5, below, illustrate how both the client and provider can get to a particular state and how it can go further, essentially describing the dynamic aspects of the system.

4.3. Process Model

For the Takeaway menu system, an iterative approach was used, see Fig 6. This model starts with planning and design, where the system is decomposed into several components developed incrementally, meaning that a working prototype is developed after each increment. This model applies the waterfall model incrementally. Due to the adoption of micro-services architecture in our system, the iterative model is inherently the best choice of a software model as the system is already divided into independent micro-services, which can be developed and presented independently.

4.4. User Stories

This section contains some of the basic requirements for the app, whose prioritisation is based onthe Moscow method. Users are divided into two types interacting with web apps, i.e., Customers and Providers. Customers will use the web app to view different restaurants and order food, whereas providers will use it to manage their restaurants and receive orders, see Fig 7.

Customers' core requirements will be searching restaurants based on their location, viewing different restaurants, exploring a restaurant's menu, adding items to a cart, ordering food, and providing a review, which will help future customers get the best food.

In contrast, providers will mainly use the web app to manage their restaurants, create and update their menus, and receive orders. They can also set their order queue, the total number of orders therestaurant can serve at a time. They will also be able to mark their restaurant as closed or offline for whatever reason.

4.5. Database Design

A database design provides a high-level overview of the system and its relationships. A robust database design is necessary for system to work efficiently and optimally. One of the essential decisions to make when designing a database is whether to use an SQL or no-SQL database design. Since the data the system receives is structured, the schema was based on an SQL design. PostgreSQL was used as the database management system of choice as it is one of the most popular, robust, and stable databases.

Since the system is based on micro-services architecture, a separate database is used for all servicesto achieve one of the most basic requirements of micro-services architecture: loose coupling among the services.

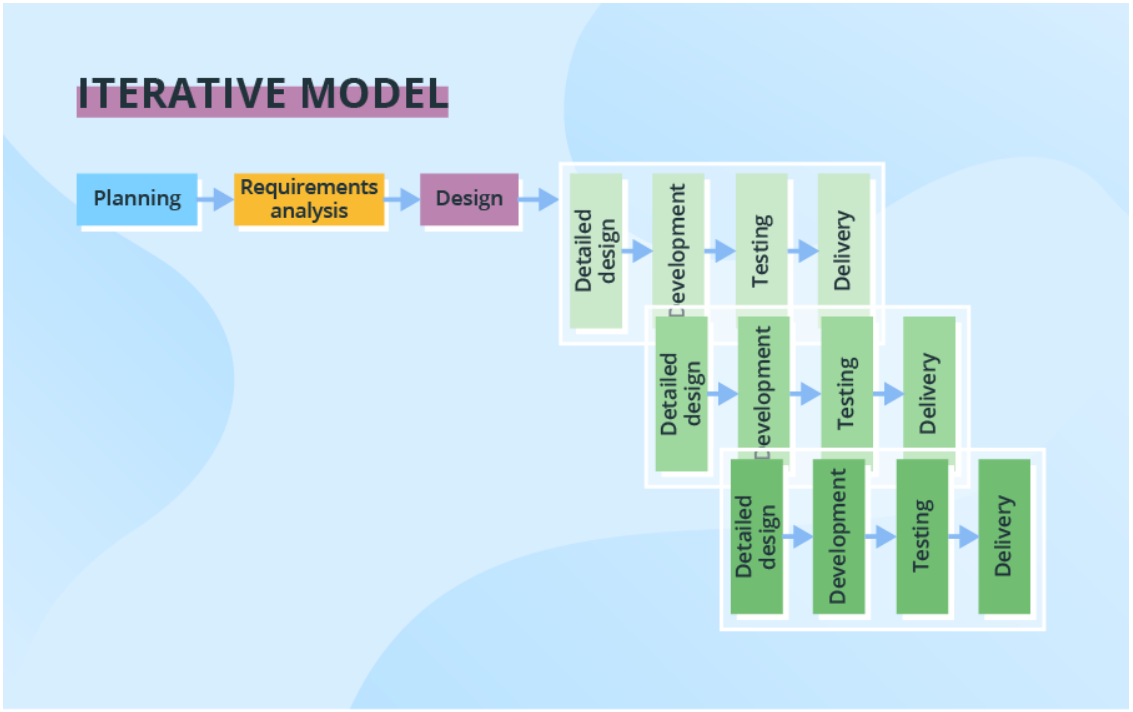


Figure 6. Iterative Process Model for Software Development

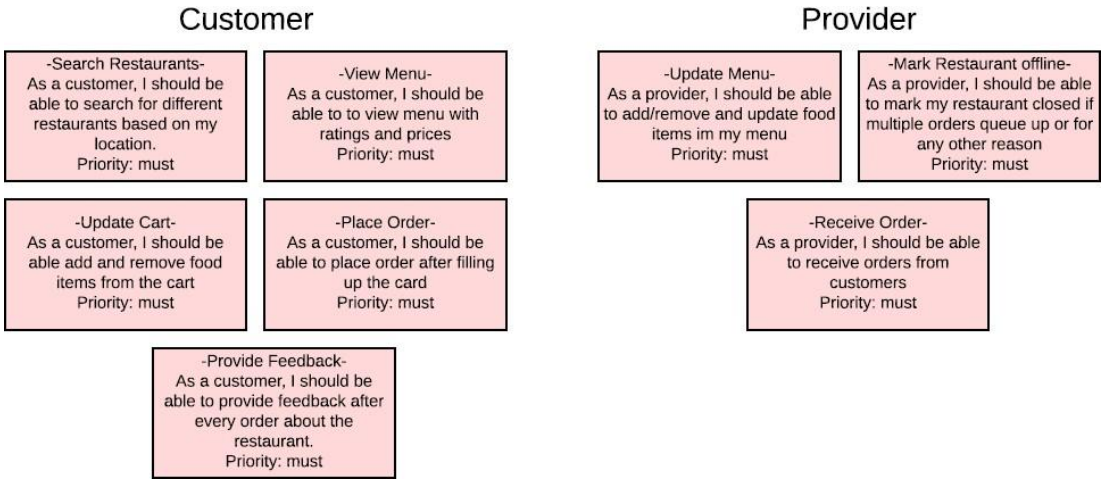


Figure 7. User Stories

4.5.1. IMS Database

The identity management service (IMS) is responsible for maintaining the basic information of the user, whether a customer or a provider. Fig 8 shows the IMS database design.

- ***User*:** This table contains all the essential information of a user. An auto-incremented integer value is a primary key and unique user id.
- ***Address*:** These tables contain all the addresses of a user (there may be more than one). Therefore, there is a one-to-many relationship between the USER and ADDRESS table. An auto-incremented integer value is used as a primary key. This table is connected to the USER table by a unique user id, which is used as a foreign key in this table. It is also connected to the TOWN table, contains a town id as a foreign key, and has a one-to-one relationship with the TOWN table.
- ***Town*:** This table contains information about a particular area of an address identified by a unique integer. This table has a many-to-one relationship with the CITY table and contains the city id as a foreign key. The data in this table is primarily static.
- ***City*:** This table contains the city part of an address identified by a unique integer as a primary key.

4.5.2. PMS DATABASE

The provider management system (PMS) database is responsible for maintaining the provider- specific data (Fig 9).

- ***Provider Info*:** This table contains the information specific to a restaurant uniquely identified by an integer

as a primary key. This table has a one-to-many relationship with CONTACT and PROVIDER_MENU_ITEM.

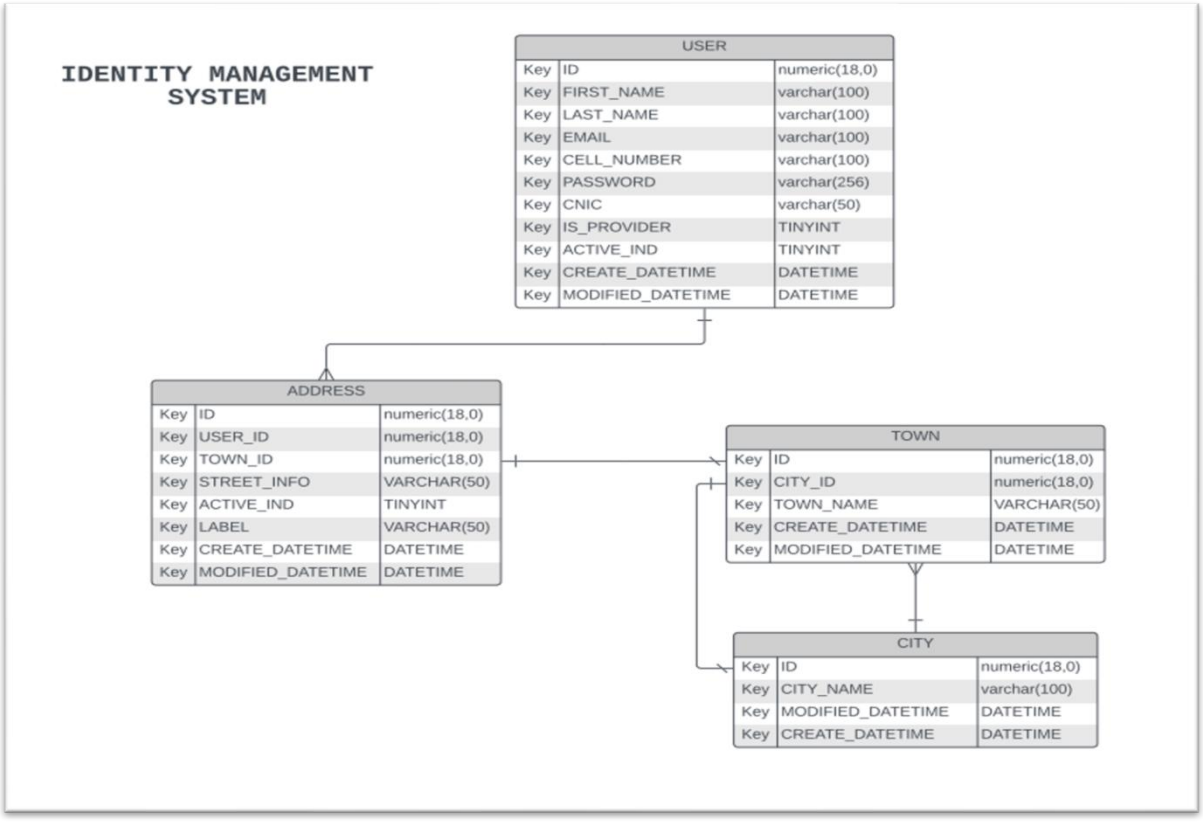


Figure 8. IMS Database design

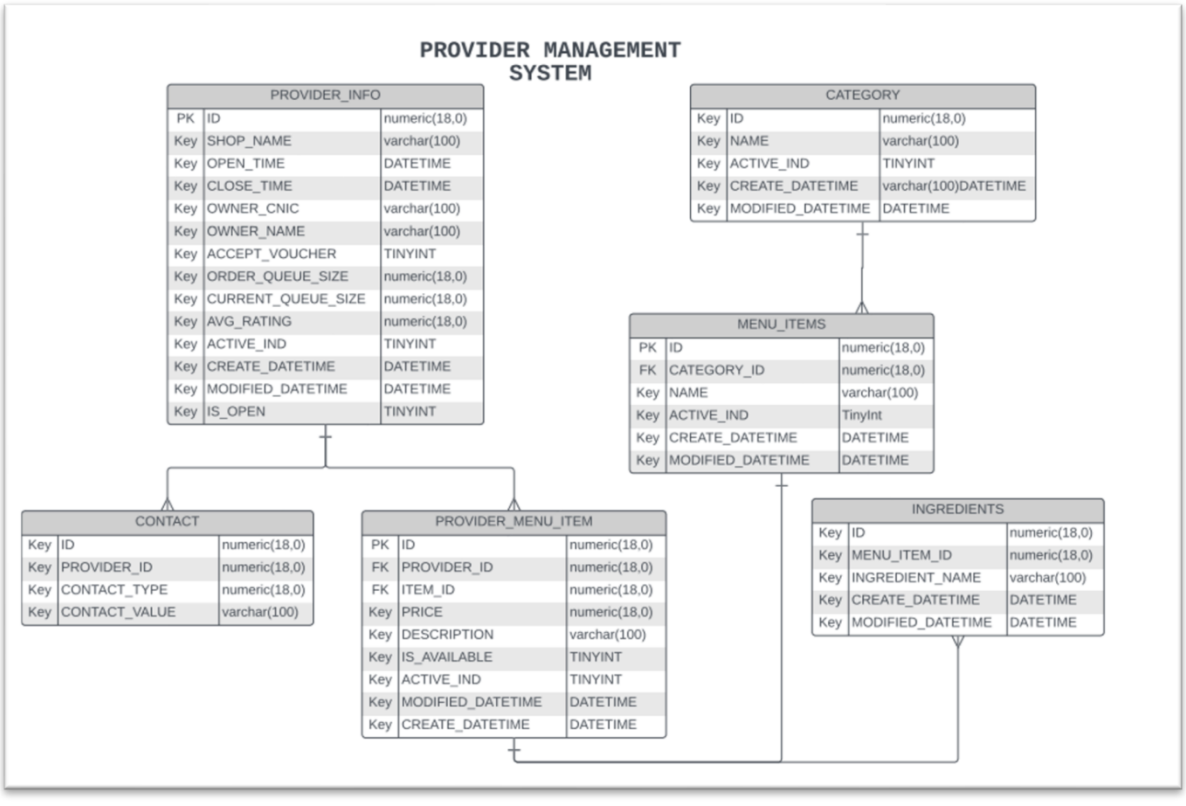


Figure 9. PMS Database design

- ***Contact*:** This table contains all a restaurant's contact details and the provider id as a foreignkey.
- ***Provider_Menu_Item*:** This table contains a restaurant's menu where a single row represents a single mm. It contains the provider id as a foreign key. This table has a one- to-one relationship with the MENU_ITEMS table with menu id as a foreign key and a one- to-many relationship with the INGREDIENT table.
- ***Menu_Items*:** This table contains a predefined set of menu items that the platform supports having many-to-one relationships with the CATEGORY table, with category id as a foreignkey.
- ***Ingredients*:** This table contains all the ingredients related to a provider menu item having the provider menu item id as a foreign key from the PROVIDER_MENU_ITEM table.
- ***Category*:** This table contains predefined food types that the platform supports.

4.5.3. OMS Database

The order management system database is responsible for maintaining currently active orders and the history of completed orders (Fig 10).

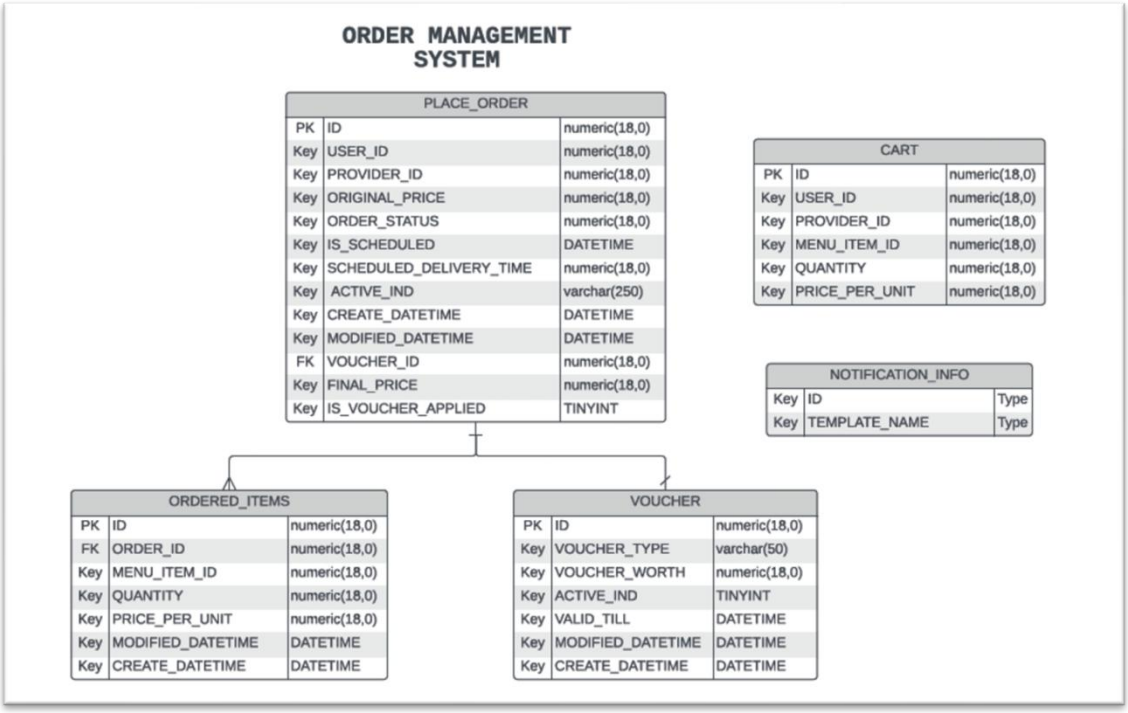


Figure 10. OMS Database design

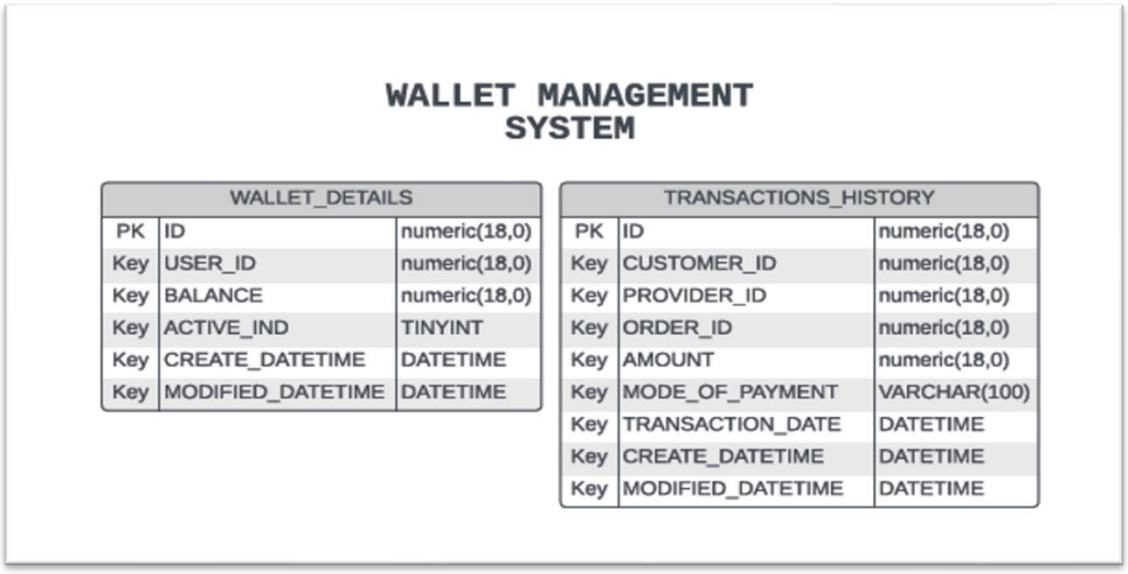


Figure 11. WMS Database design

- ***Order*:** This table contains all the currently active orders and the history of completed orders. An integer order id uniquely identifies the orders. This table has a one-to-one relationship with the VOUCHER table, having voucher id as a foreign key.
- ***Ordered_Item*:** This table contains all the items placed in a single order. It has a many-to-one relationship with the order table, having the order id as a foreign key.
- ***Voucher*:** This table contains information about vouchers applied to different orders.
- ***Cart*:** This table contains the items present in a cart when the customer is in the processof ordering food.

4.5.4. WMS DATABASE

The wallet management system databse contains information about users' e-wallets andtransaction history (Fig 11).

- ***Wallet_Details*:** This table contains the available balance in the user's e-wallet.
- ***Transaction_History*:** This table contains the transaction histories of all users.

4.5.5. NMS DATABASE

The notification management system uses the NMS database (Fig 12) and records all differenttypes.

- ***Notification_Template*:** This table contains the notification templates for sending emailsand SMS.

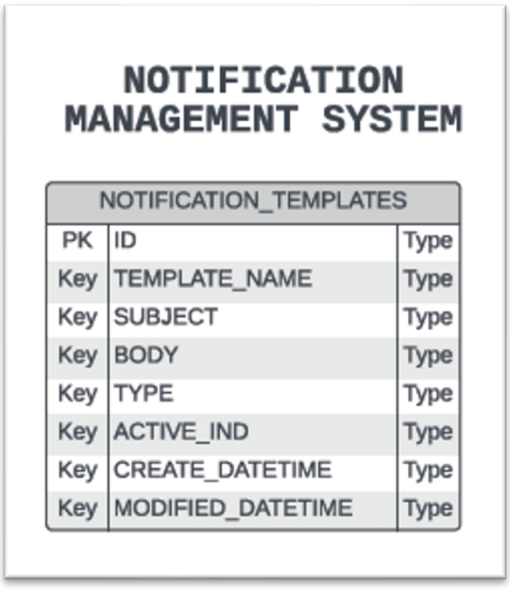


Figure 12. NMS Database

5. Implementation

This chapter covers the project system implementation, including the tools used in this project, which are further divided into development tools, services, and software tools. It also discusses the system architecture, what technologies are being used, and on which architecture the system is built. It will highlight the server implementation and some code snippets to understand the structure and implementation better.

5.1. Technologies Stack

5.1.1. Spring Boot

The Spring Boot framework is quite famous for building web applications at an enterprise level. It provides excellent flexibility in developing applications in SOA architecture. It provides several starter projects to jump-start the development, along with several brilliant features like dependency management, auto-configuration, and built-in CRUD handling (Guntupally et al., 2018).

5.1.2. Node.Js

One of the most in-use and demanded server-side development environments in the JavaScript space is Node.js, a JavaScript environment used to develop high-performance applications and reply to event-driven programming models with asynchronous behavior (Tilkov & Vinoski, 2010). Node.js has become successful because of its ability to handle high concurrency and a large amount of data compared to other languages such as PHP and Python, which handle much fewer requests. After some experiments, the results showed that Node.js is lightweight and efficient (Lei et al., 2014). Node.js also eliminates developers' complexity in learning multiple languages for front-end and back-end development. A developer confident with JavaScript can learn a few more back-end functionalities and become a back-end developer. Also, Node.js is free, another reason for its global appeal (Shah & Soomro, 2017).

5.1.3. Express.Js

A framework should increase the productivity and speed of a project for software professionals because the frameworks are used and updated by numerous other developers and contributors, andthe outcomes are frequently of higher quality. Programmers that develop everything from the ground up will ultimately establish their own framework, a highly personalised one. Regarding frameworks, Node.js is a relatively new platform (unlike Ruby and Java). However, Express.js has already established itself as a leader, with the majority of Node.js apps using use the Express.js framework (Mardan, 2018).

Express is a flexible application framework for Node.js that offers a robust set of functionalities for both web and mobile applications. For this reason, Express.js offers flexibility to the developers for the development of web applications (Mardan, 2018).

5.1.4. React.Js

React.js is a free and open-source library of JavaScript that is mainly used to develop user interfaces. It provides a way to handle the view layer of the application of MVC architecture. It also allows the creation of components that are reusable

pieces of code, resulting in faster development and fewer bugs. React was introduced by Facebook (now known as the Meta), which also maintains it, along with a community of individual developers.

5.2. Tools Used

This section will cover the tools used to develop the User Identity system. The tools are divided into languages and frameworks, external libraries for utility functions, and software to manage thecode. The categories of the tools used are described in this section.

5.2.1. Development Tools

Table 1 details the server-side tools used in the project, while Table 2 details the client-sidedevelopment tools.

Server-Side

Table 1. Server-Side Development tools

COMPONENT	TOOL	TYPE	PURPOSE
SERVER	Node.js	Environment	The environment in which the JavaScriptlanguage runs
	JavaScript	Language	Back-end language usedfor development
	Express.js	Framework	Node.js framework forbuilding APIs
	Knex.js	Module	Schema designing andconnecting to PostgreSQL database
	Bcrypt.js	Module	For password hashing andpassword validating
	JWT Tokens	Module	JWT implementation inNodejs
	Java	Language	Back-end language usedfor development
	Spring Boot	Framework	Java framework forbuilding APIs

Client-Side

Table 2. Client-Side Development tools

COMPONENT	TOOL	TYPE	PURPOSE
CLIENT	React.js	Framework	React is an open- source framework ofJavaScript used to build U.I. interfaces through components.
	HTML5	Markup Language	Used to define the structure of a web page using markuptags.
	CSS3	Style Sheet Language	Used to define thepresentation of anHTML document.
	Bootstrap	Framework	Open-source framework to speedup styling process.

5.2.2. Services

In addition to the development tools mentioned above, the system also uses some other external services, as shown in Table 3.

Table 3. Service used

SERVICE	PURPOSE
Git	Version control
GitHub	Code backup
Docker	Package applications in a container
Zipkin	Monitoring and tracing APIs call
Eureka Server	Service registry
Lombok	Boilerplate code

5.3. Software Tools

Some additional software tools were also used alongside the services and development tools previously mentioned. These are listed in Table 4:

Table 4. Software Tools

SOFTWARE	PURPOSE
Visual Studio Code	The text editor writes code, debugs, and runs testcases.
DBeaver	Database tool to manage data.
PgAdmin	Web-based GUI database tool to interact and createa database.
IntelliJ Idea	The most popular IDE for Java, as it provides manyfeatures and helps in development.
Postman	Helps send requests to the back-end for testingpurposes.

5.4. System Architecture

Our system architecture's tech stack includes Node.js, Express.js, Java, and PostgreSQL as a database. The reason for choosing Java and Node.js as back-end languages is that Java is very popular in building applications using micro-service architecture, whereas Node.js is currently the most popular and preferred choice among developers for

web applications. PostgreSQL is also known as the most advanced open-source database, and due to its advanced features and open- source nature, it was chosen for the database.

Maintaining and scaling projects is one of the hardest things in software development. Generally, every project starts as a monolithic application with perfectly defined modules with single responsibilities; as new features are added, the codebase becomes more extensive and lacks structure and design. This project was designed keeping the micro-services architecture in mind. Although designing the micro-services architecture and finding the right set of services is challenging, it provides significant advantages over monolithic architecture.

The system is divided into six micro-services based on the services' work (Fig 13). These are Identity Management Service, Provider Management Service, Order Management, Notification Management Service, Wallet Management Service, and Recommender Management Service.

Identity Management Service:

This service is responsible for all account-related activities. It has all the basic user information, and all requests related to account creation, authentication, authorisation, profile information updates, and password changes are routed towards this service. This service uses the JWT Web Token for authentication. It delegates the task of sending notifications to the Notification Service.

Provider Management Service:

This service provides the front end with all the restaurant-related data, including the restaurants, menus, and prices. It also provides APIs for the provider to manage its restaurants and perform tasks like creating/updating, creating menus, and accepting orders. It receives orders from the Order Service and passes them to the correct provider.

Order Management Service:

This service receives all the orders, processes them, and stores the order history. It also maintains a separate cart for each customer. It receives the order from the customer and places it in the RabbitMQ queue for Provider Service to consume it. It also delegates the work of sending order confirmation emails and SMS to the Notification Service.

Wallet Management Service:

This system also provides a wallet for cashless transactions. Wallet Service is responsible for maintaining the wallets of all customers, keeping the wallet in the correct and stable state, and maintaining the transaction history.

Notification Management Service:

This relatively simple service is only responsible for sending different types of notifications to customers and providers through emails and SMS.

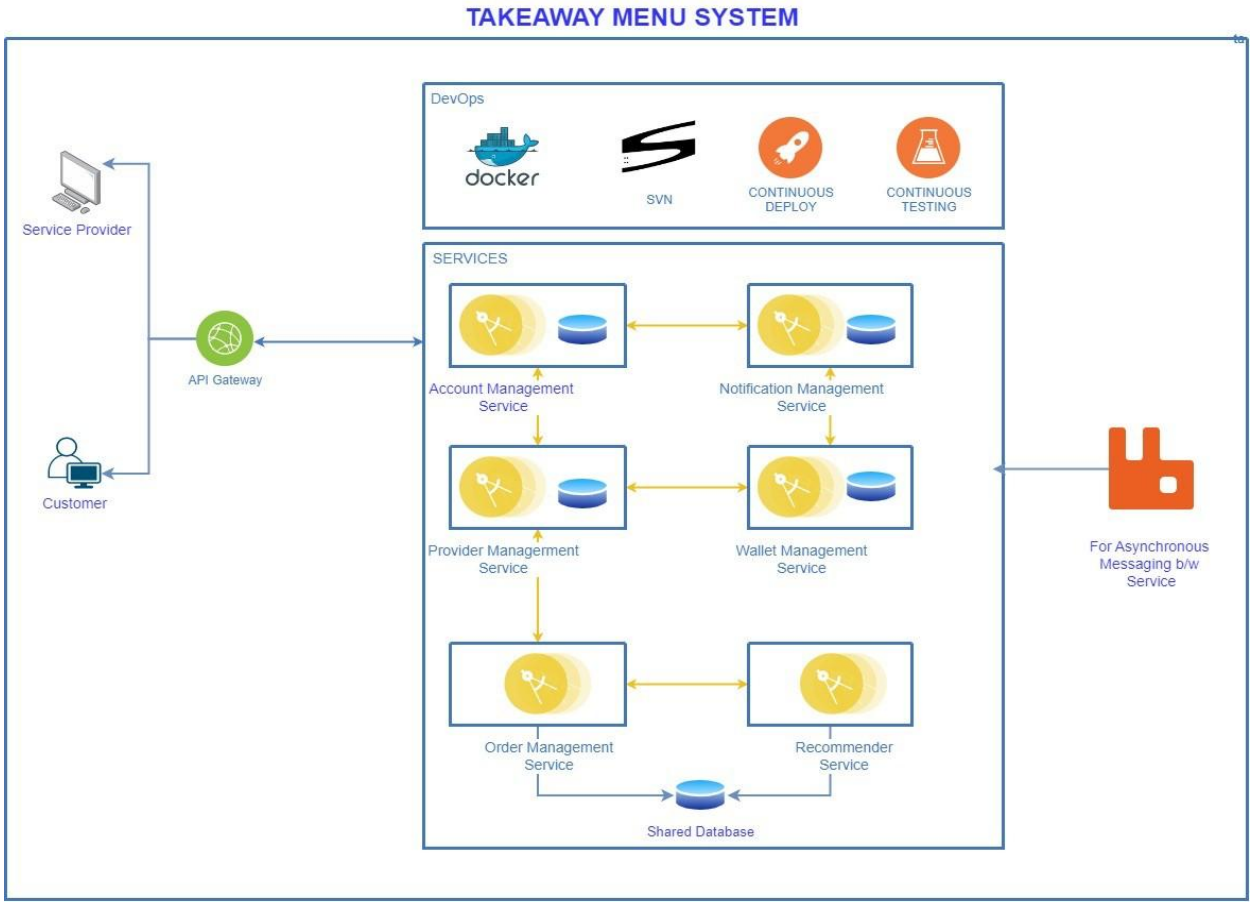


Figure 13. TMS Architecture Diagram

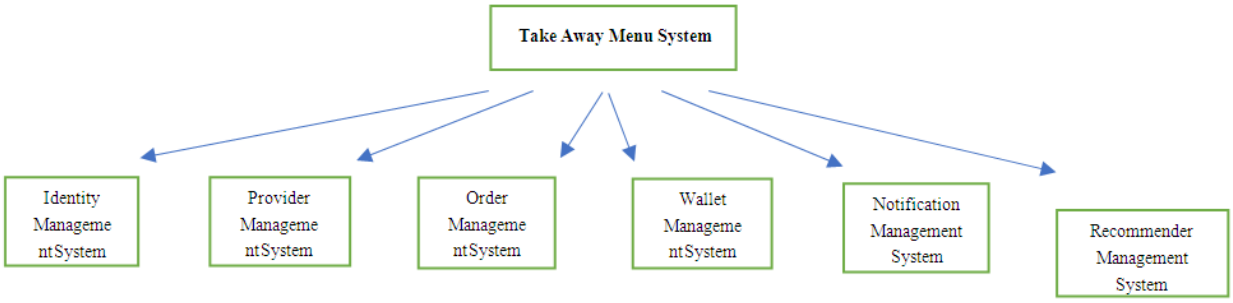


Figure 14. Server Implementation

Recommender Management Service:

This service is responsible for extracting different insights from the order data and providing customers with exciting recommendations based on their preferences. It can recommend trending foods among restaurants, most bought food from a restaurant, food to order next based on personal preference, and many others.

5.5. Server Implementation

The server was designed using the REST API, completely decoupled from the front end; it communicates only through the API. The server was developed on Node.js using Express and onJava using Spring Boot as a framework.

Figure 14 shows the structure of the project. The structure is comprised of several services built on Nodejs and Spring Boot having their own tasks to perform. The identitymanagement service is responsible for managing all the user related functionalities such as creating a user, updating its profile, changing credentials, and managing user authentication. The provider management service is responsible for managing the provider's related tasks, which in this case is the shop owner. Several functionalities are handled in this service, some of which are receiving orders and dispatching them. Next, we have the order management service, which contains all the functionalities related to the orders that are being placed and manages their history. The wallet management service is where a user's wallet is handled and contains an amount that allows the user to pay for their order. Another very important service is the recommendation service, which provides suggestions for where a user form could order from.

5.5.1. Request Cycle

In Node.js we have separate files for defining routes where end points are defined. When a requestis made it lands on the routes file, finds the associated endpoints, then its linked controller is called, which directs the call to the service file in which the business logic is defined. If data manipulation is needed, then knex is used to interact with the database. When the request is completed it is then sent back in the reverse order of its arrival.

In Java routes are mapped on top of the controller class with endpoinst mapped on the methods. When a request first arrives at the controller, the controller calls the service class to execute the business logic. In case of database interactions, repositories are called within the service file to fetch or update data. On completion of the request, a response is sent back to the controller, which then sends it back to the client. Figure 15, below, is a diagrammatic representation of the flow of request within an application.

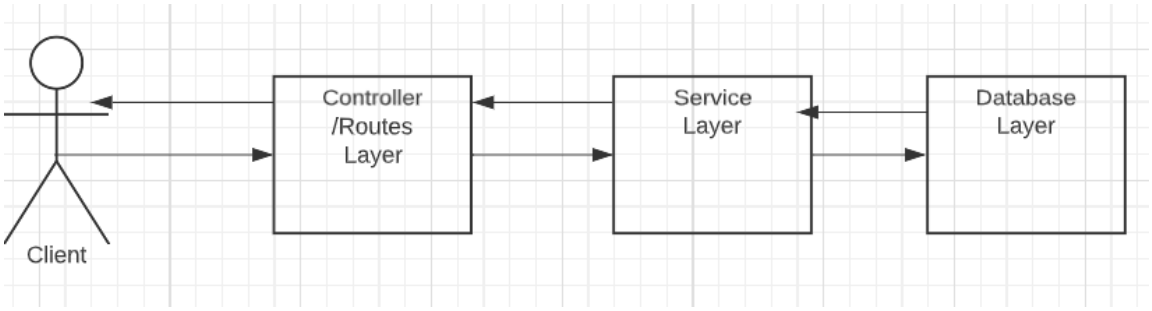


Figure 15. Request cycle diagram

5.5.2. Node.Js Implementation

```
const express = require('express');
const router = express.Router();        You, 3 weeks ago • Created routes and controllers …

const CustomerController = require('../../controllers/CustomerController');
const controller = new CustomerController();

// All the routes for customer is here
router.post('/checkEmailAndCell', controller.checkEmailAndCell.bind(controller));
router.post('/signup', controller.signUp.bind(controller));
router.post('/login', controller.login.bind(controller));
router.post('/sendCode', controller.sendCode.bind(controller));
router.post('/forgetPassword', controller.forgetPassword.bind(controller));


module.exports = router
```

In the above code snippet, we are defining the endpoints of the application. Firstly, the express andcontroller is imported in the routes file then the instances are created for the router, which is usedto route the request to the appropriate controller, and for the controller, which calls the controller function. The controller here is simply responsible for receiving the request and responding back.Each controller is derived from a base controller.

```
async checkEmailAndCell(data) {       You, 3 weeks ago • Signup API done …
    try{

        // checking email if it exist in database or not
        console.log(">>>>>>>>>> Checking email");
        let emailExists = await this.userUtils.checkEmail(data.email);
        if(emailExists){
            console.log("Provided email already exists");
            throw new Exception(STATUS_CODES.BAD_REQUEST, RESPONSE_MESSAGES.ERROR_EMAIL_EXISTS);
        }

        // checking cell number if it exist in database or not
        console.log(">>>>>>>>>> Checking cell number");
        let cellNoExists = await this.userUtils.checkCellNo(data.cellNumber);
        if(cellNoExists){
            console.log("Provided cell number already exists");
            throw new Exception(STATUS_CODES.BAD_REQUEST, RESPONSE_MESSAGES.ERROR_CELLNO_EXISTS);
        }

        // calls otp service to get OTP

        let otp = 45879;

        // returning OTP
        return otp;
    }
    catch(err){
        // returning the error to the controller
        throw err;
    }


}
```

In the above code snippet, we are defining the service logic. The service is where the main login of the function is. In the above snippet the service is checking the email and cell number against the database and if it already exists it throws an error and if not then an OTP is generated. The structure of the service is such that it contains some utility functions that are responsible for formatting the request and calling the model where the database query is placed.

```
/**
 * Checks email in the database
 * @param {*} email
 * @returns
 */
async checkEmail(email){       You, 3 weeks ago • Signup API done …

    // Getting email from DB
    let userEmail = await userModel.getEmail(email);

    // Returning email if found in DB otherwise null
    return userEmail ?? false;
}
```

In the above code snippet, a utility function is defined that is responsible for the validation of theemail. This function check finds the email address in the database and if found returns that email,otherwise it returns a null response.

```
/**
 * This function is responsible to if the provided email and cell nu,ber already exists in database or not
 * @param {*} req request object
 * @param {*} res response object
 */
async checkEmailAndCell(req, res){      You, last week • Send code and forget password api done …
    try{

        console.log(">>>>>>>>>>>>>>>>>>>>>>>> CHECK EMAIL API <<<<<<<<<<<<<<<<<<<<<<<<");

        console.log(">>>>>> BODY: ", JSON.stringify(req.body));

        // calling checkEmailAndCell function inside user service
        let response = await this.userService.checkEmailAndCell(req.body);

        // returning response to the user
        res.success(STATUS_CODES.SUCCESS, RESPONSE_MESSAGE.SUCCESS, {otp: response})
    }
    catch(err){
        // consoling the err
        console.log(err);
        //handling the error
        this.handleExceptions(err, res);
    }


}
```

In the above image we are defining our controller logic. The controller is responsible for receiving the request and responding back. The controller also receives two parameter requests and responses. The request object contains the complete incoming data and the response object is responsible to respond back.

```
You, last week | 1 author (You)
const db = require('../../database/dbObject');

💡 Checking if the provided email exists in DB or not
const getEmail = async (email) => db('USER').select('ID').where({EMAIL: email}).first();       You,

// Get user from database through cell number
const getCellNo = async (cellNo) => db('USER').select('ID').where({CELL_NUMBER: cellNo}).first();

// inserts user in database
const insertUser = async (dataObj) => db('USER').insert(dataObj).returning('*');

// Get user from database through email
const getUserByEmail = async (email) => db.select('*').from('USER').where({EMAIL: email}).first();

// Getting user from DB by it user id
const getUserById = async (id) => db.select('*').from('USER').where({ID: id}).first();

// Updating user by user id
const updateById = async (id, obj) => db("USER").where({ID: id}).update(obj);

// Updating user by cell number
const updateByMobile = async (mobile, obj) => db("USER").where({CELL_NUMBER: mobile}).update(obj);


module.exports = {
    getEmail,
    getCellNo,
    insertUser,
    getUserByEmail,
    getUserById,
    updateById,
    updateByMobile
}
```

The above code snippet consists of data layer logic. These functions accept some parameters and queries from the database and return the responses. Knex query builder is used for database queriesbecause of its simplicity and fast execution.

5.5.3. Java Implementation

```java
package com.takeawaymenusystem.orderservice.order;

import ...

@Slf4j
@RestController
@RequestMapping("tms/oms/order/")
public record OrderController(OrderService orderService) {

    @GetMapping("getOrdersByUserId/{userId}")
    public ServiceResponse<List<OrderResponseDto>> getOrderByUserId(@PathVariable("userId") Long userId){
        ServiceResponse<List<OrderResponseDto>> response = new ServiceResponse<>();
        try{
            response.setStatusCode(ResponseCode.OK);
            response.setResult(orderService.getOrderHistoryByUserId(userId));
        }catch (BusinessException ex){
            response.setStatusCode(ex.getErrorCode());
            response.setMessage(ex.getMessage());
        }
        return response;
    }
```

```java
@Slf4j
@RestController
@RequestMapping("tms/oms/cart/")
public record CartController(CartService cartService) {

    @PostMapping("addToCart")
    public ServiceResponse<CartResponseDto> addItemToCart(@RequestBody CartUpdateRequestDto cartUpdateRequest){
        ServiceResponse<CartResponseDto> response = new ServiceResponse<>();
        try{
            log.info("here at controller");
            response.setStatusCode(ResponseCode.OK);
            response.setResult(cartService.updateCart(cartUpdateRequest));
        }catch (BusinessException ex){
            response.setStatusCode(ex.getErrorCode());
            response.setMessage(ex.getMessage());
        }
        return response;
    }
```

In the above code snippets, the controller of the order and cart services is shown, which is the end point that is exposed externally. All the requests first arrive at the controller, then based on the request mapping the appropriate service function is called, which performs the business logic and returns some response. The response is then sent back to the client from the controller.

```java
public CartResponseDto updateCart(CartUpdateRequestDto request) throws BusinessException{

    CartResponseDto responseDto;
    Cart cart = cartRepository.findByUserId(request.getUserId());
    if(cart == null) {
        Cart userCart = Cart.builder()
                .userId(request.getUserId())
                .providerId(request.getProviderId())
                .shopName((request.getShopName()))
                .cartItemList(request.getCartItemList())
                .createDateTime(LocalDate.now())
                .modifiedDateTime(LocalDate.now())
                .build();
        cartRepository.save(userCart);
        responseDto = mapper.map(userCart, CartResponseDto.class);
        responseDto.setCreateDateTime(LocalDate.now());
        return responseDto;
    } else {
        cartRepository.deleteById(cart.getCartId());
        Cart userCart = mapper.map(request, Cart.class);
        cartRepository.save(userCart);
        responseDto = mapper.map(userCart, CartResponseDto.class);
        responseDto.setModifiedDateTime(LocalDate.now());
        return responseDto;
    }
}
```

The above code snippet shows the implementation of the update process of a user's cart. In SpringData JPA, a repository is used for database operation. A cart repository is implemented that is responsible for interacting with the database. The cart repository is checked and if the cart does not exist then it creates a new one, otherwise it removes any previous cart information and updatesit with new data.

```
public OrderResponseDto placeOrder(OrderRequestDto requestDto) throws BusinessException{
    OrderResponseDto responseDto = new OrderResponseDto();
    ServiceResponse<Boolean> hasEnoughCredit = new ServiceResponse<>();
    Cart cart = cartRepository.findByUserId(requestDto.getUserId());
    log.info("get cart {}",cart);
    if (cart != null) {
        if(requestDto.getModeOfPayment().equals(ModeOfPayment.THROUGH_WALLET)){
            hasEnoughCredit = walletClient.hasCredit(requestDto.getUserId(), requestDto.getCost());
            if(hasEnoughCredit.getResult()) {
                responseDto = performPlaceOrder(requestDto,cart);
            }
            else{
                throw new BusinessException(ErrorCode.INSUFFICIENT_CREDIT,"Insufficient credit to complete the order");
            }
        }else if (requestDto.getModeOfPayment().equals(ModeOfPayment.CASH_ON_DELIVERY)){
            responseDto = performPlaceOrder(requestDto,cart);
        }
        cartRepository.deleteById(cart.getCartId());
    }else{
        throw new BusinessException(ErrorCode.EMPTY_CART,"Cart does not exist");
    }
    return responseDto;
}
```

The code shown in the above snippet is of utmost importance as it is responsible for the placementof the order. The process of placing an order starts with checking if the customer has any items in the cart. If it does, then the next step is to check the method of payment. If the method of payment is cash on delivery, then the order is placed but if the method of payment is through a digital wallet, then a request is sent to the wallet service using the Open Feign client to validate the account balance of the user. If the wallet service responds with ok, then the order is placed otherwise an appropriate response is sent back.

5.6. Front-End Implementation

The front-end of the app is built using react, a library for building web-applications. The reactlibrary can be used to build fast, optimised applications and it has good support for the styles.

The front-end code contains the component structure; there is a main component to which all the other components are imported and used. Each component is responsible for a particular task.

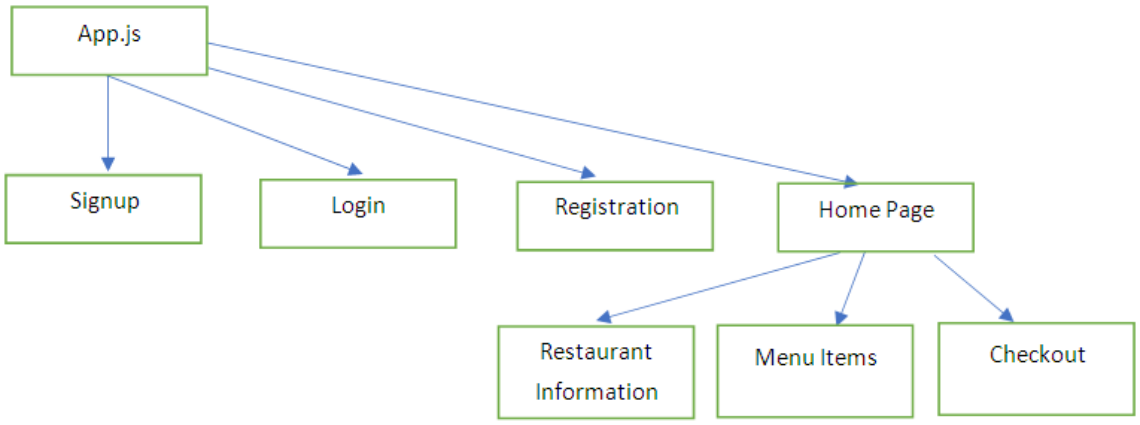
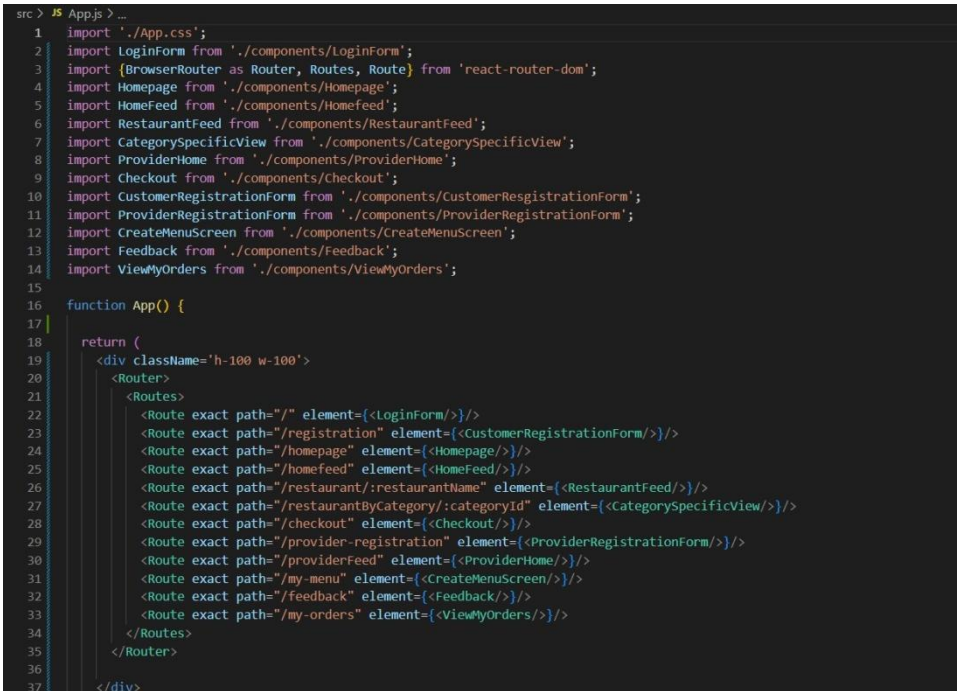


Figure 16. Basic front-end code structure

Figure 16 shows how the front-end part is divided. All the components are imported and bootedfrom the main component that is app.js.



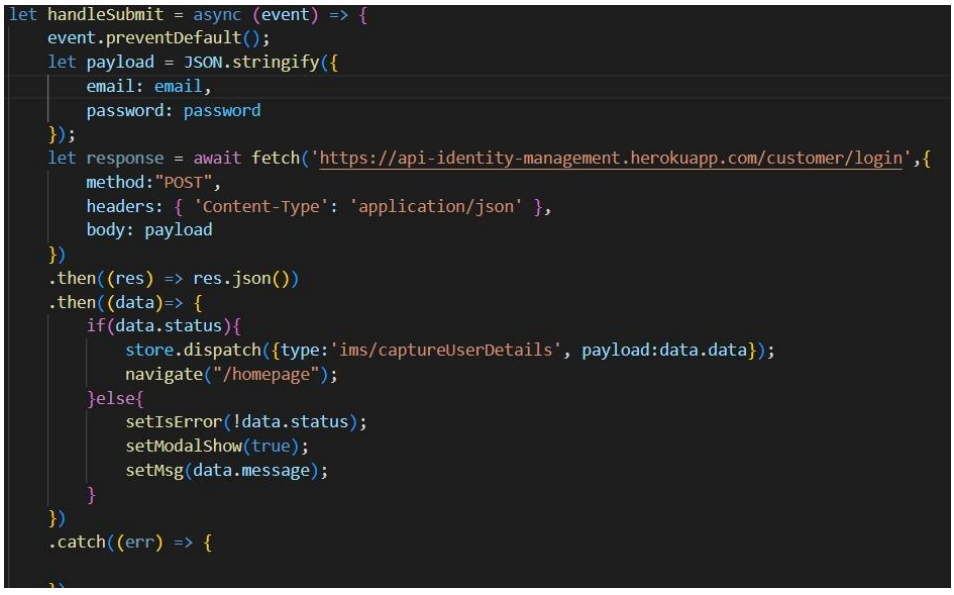
```
src > JS App.js > ...
import './App.css';
import LoginForm from './components/LoginForm';
import {BrowserRouter as Router, Routes, Route} from 'react-router-dom';
import Homepage from './components/Homepage';
import HomeFeed from './components/Homefeed';
import RestaurantFeed from './components/RestaurantFeed';
import CategorySpecificView from './components/CategorySpecificView';
import ProviderHome from './components/ProviderHome';
import Checkout from './components/Checkout';
import CustomerRegistrationForm from './components/CustomerResgistrationForm';
import ProviderRegistrationForm from './components/ProviderRegistrationForm';
import CreateMenuScreen from './components/CreateMenuScreen';
import Feedback from './components/Feedback';
import ViewMyOrders from './components/ViewMyOrders';

function App() {

  return (
    <div className='h-100 w-100'>
      <Router>
        <Routes>
          <Route exact path="/" element={<LoginForm/>}/>
          <Route exact path="/registration" element={<CustomerRegistrationForm/>}/>
          <Route exact path="/homepage" element={<Homepage/>}/>
          <Route exact path="/homefeed" element={<HomeFeed/>}/>
          <Route exact path="/restaurant/:restaurantName" element={<RestaurantFeed/>}/>
          <Route exact path="/restaurantByCategory/:categoryId" element={<CategorySpecificView/>}/>
          <Route exact path="/checkout" element={<Checkout/>}/>
          <Route exact path="/provider-registration" element={<ProviderRegistrationForm/>}/>
          <Route exact path="/providerFeed" element={<ProviderHome/>}/>
          <Route exact path="/my-menu" element={<CreateMenuScreen/>}/>
          <Route exact path="/feedback" element={<Feedback/>}/>
          <Route exact path="/my-orders" element={<ViewMyOrders/>}/>
        </Routes>
      </Router>

    </div>
```

This code snippet is from the app.js and this is where all the components are rendered and all routes are defined as it is the root node of the tree hierarchy of the application.



```
let handleSubmit = async (event) => {
    event.preventDefault();
    let payload = JSON.stringify({
        email: email,
        password: password
    });
    let response = await fetch('https://api-identity-management.herokuapp.com/customer/login',{
        method:"POST",
        headers: { 'Content-Type': 'application/json' },
        body: payload
    })
    .then((res) => res.json())
    .then((data)=> {
        if(data.status){
            store.dispatch({type:'ims/captureUserDetails', payload:data.data});
            navigate("/homepage");
        }else{
            setIsError(!data.status);
            setModalShow(true);
            setMsg(data.message);
        }
    })
    .catch((err) => {
```

The above code snippet is for the login component where the request is sent to the server withthe login credentials.

```
export default function appReducer(state=initialState, action){
    switch (action.type){
        case 'cart/addToCart':
            return {
                ...state,
                cart: {
                    ...state.cart,
                    cartItemList: addToCart(state.cart.cartItemList,action.payload)
                }
            }
        case 'cart/removeFromCart':
            return {
                ...state,
                cart: {
                    ...state.cart,
                    cartItemList : removeFromCart(state.cart.cartItemList,action.payload)
                }
            }
        case 'cart/incrementItem':
            return{
                ...state,
                cart: {
                    ...state.cart,
                    cartItemList: incrementItemCount(state.cart.cartItemList, action.payload)
                }
            }
        case 'cart/decrementItem':
            return{
                ...state,
                cart: {
                    ...state.cart,
                    cartItemList: decrementItemCounter(state.cart.cartItemList, action.payload)
                }
            }
```

The above code snippet shows the implementation of the reducer function, which is used to maintain the state of the application. Redux is used because of the difficulty of sharing data among components. In parent-child hierarchy, data can be passed down as props and sent back but when components are siblings or have multiple layers data synchronisation and maintainability becomes challenging. For this reason, Redux is used. The reducer function shown above is responsible for mutating the state of the application.

```
src > JS index.js > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
6  import store from './store';
7  import {Provider} from 'react-redux';
8
9  const root = ReactDOM.createRoot(document.getElementById('root'));
10
11 root.render(
12     <Provider store={store}>
13       <App/>
14     </Provider>
15 );
16
17 // If you want to start measuring performance in your app, pass a function
18 // to log results (for example: reportWebVitals(console.log))
19 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();
21
```

The above screenshot shows the initialisation of the store, which is where the state of the application is kept. The store functions as the single source of truth.

```
let incrementCounter = (e) => {
  console.log(e.target);
  store.dispatch({type:'cart/incrementItem', payload: e.target.value});
}

let decrementCounter = (e) => {
  console.log(e.target.value)
  store.dispatch({type:'cart/decrementItem', payload: e.target.value});
}
```

In the above screenshot, implementation of increment and decrement operations on the cart is shown. The data in the cart is stored in the application state because it is needed to be shared amongdifferent components. So, to update the state of the cart, the dispatch method of the store is invoked, which takes type and action as arguments. Type tells it what operation to perform on the state and on which section of the state. Action provides the new data that is needed to update thestate.

6. Discussion

TESTING

This chapter will cover the testing scenarios implemented for the two major components of the project i.e., the front-end side of the TMS and the server part. These components were tested carefully to eliminate any possible errors and bugs in the system. The front-end part was tested to ensure that there were no design issues and all buttons and redirections work as required. The server was tested to prevent all the server side issues, bugs, and code crashes.

6.1. Front-End

Our application consists of several vital screens that are necessary to provide a unique and enhanced user experience. To provide a great user experience it is of the utmost importance that the screens are well tested and behave as expected. For this purpose, we opted for black box testing; this is done by navigating to the screen to be tested and checking that it has loaded successfully and that there are no errors in the API calls. Then the functionality provided by the screen is tested by providing input data or clicking the relevant button, depending on the interface, and validating whether the response on user input is behaving as expected. After all the screens were tested individually, then the overall flow of the application was tested by navigating to different pages and validating that the correct state was maintained across all the components and, on any event, that appropriate actions were called to update the state of application. The most difficult part of frontend development is to maintain the overall state of application and make sure that the integrity of state is maintained. There are several components that need access to the state of the application to correctly display information. For example, the information in a user's cart is used by cart, checkout, home feed, and restaurant components. Following this, it was tested to ensure that data manipulation is efficiently handled. Below are the snapshots of the screens indicating the tests that have been done on them:

Case 1: If wrong credentials are entered, an appropriate message is shown.

Case 2: When a provider has entered details of its item to be added to the menu and submitted them then the item is added into the database and a message is shown indicating the success of theoperation.

Case 3: On landing on the menu page a provider can successfully view the items in the menu. The provider also has an option to edit the data of any item in the menu.

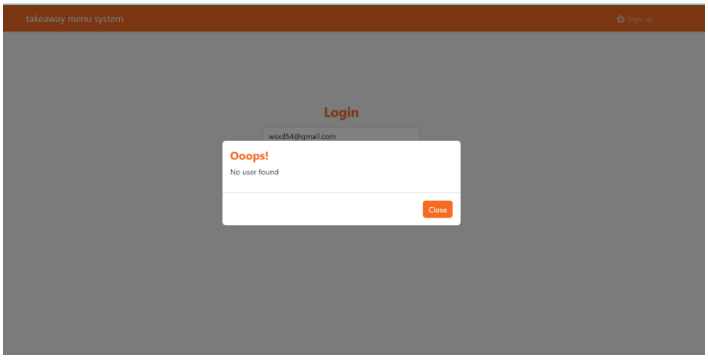
Case 4: Successful integration of map within the application. With the help of map marker, a usercan pinpoint its location.

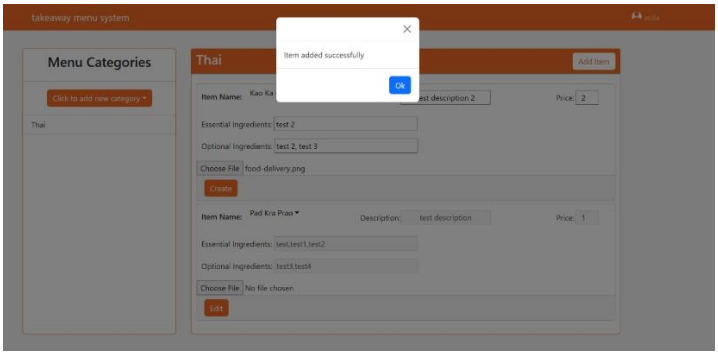
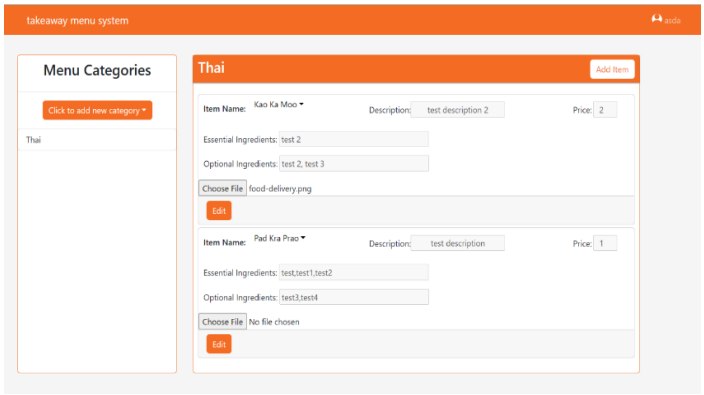
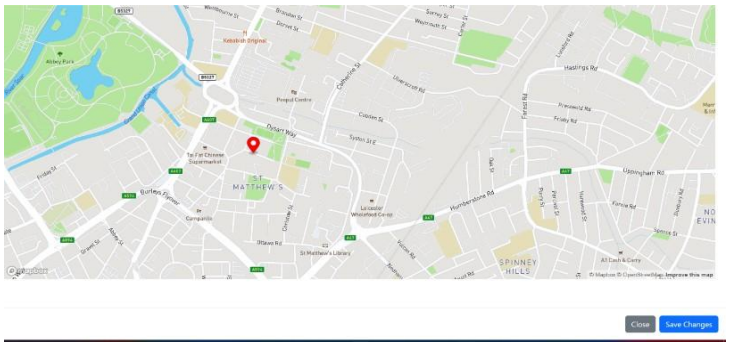
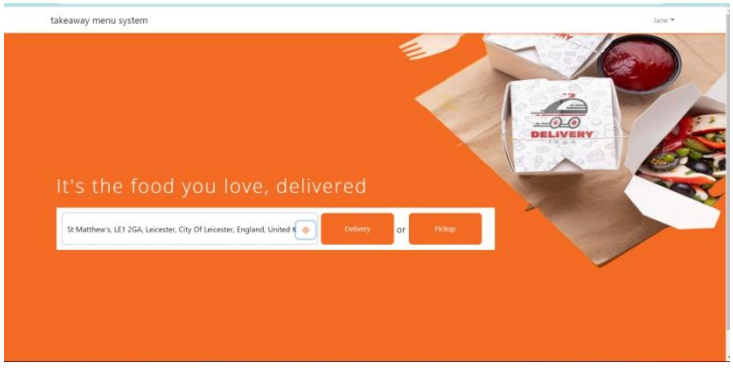
Case 5: As marked by the marker on the map, location is successfully shown in the delivery address tab which will be used to bring up the nearby restaurants

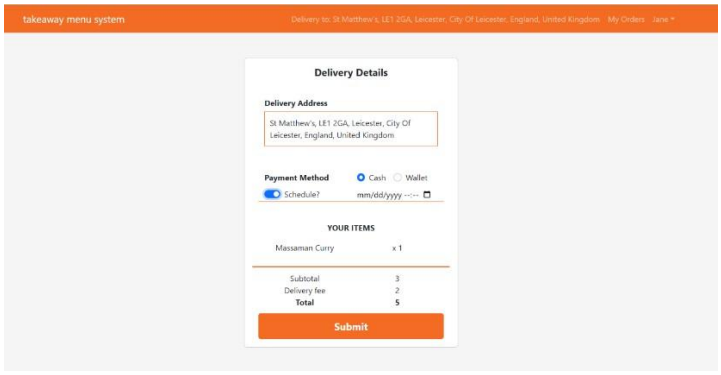
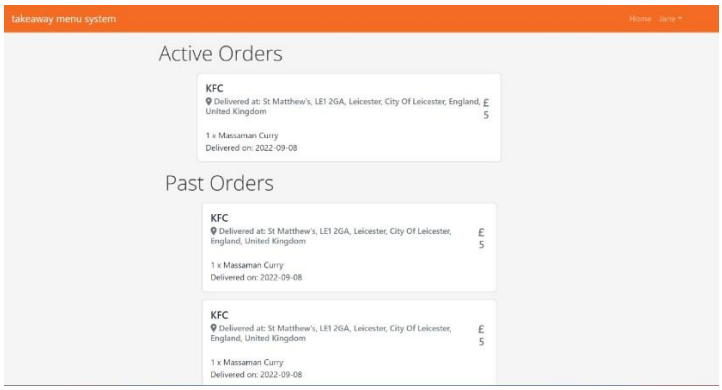
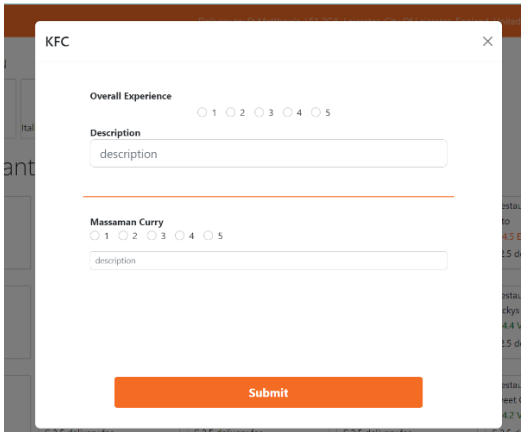
Case 6: The user can successfully view the items in the cart, the delivery address, the mode of payment, and the scheduling option on the checkout screen.

Case 7: On successful placement of the order, the user is redirected to the view order page where the order history of the user is shown. This screen also confirms that the order is successfully placed because active orders are shown at the very top.

Case 8: After the order is placed, when the customer re-logs in a feedback modal is shown where an item-based review is taken from the customer that captures fine-grained feedback.

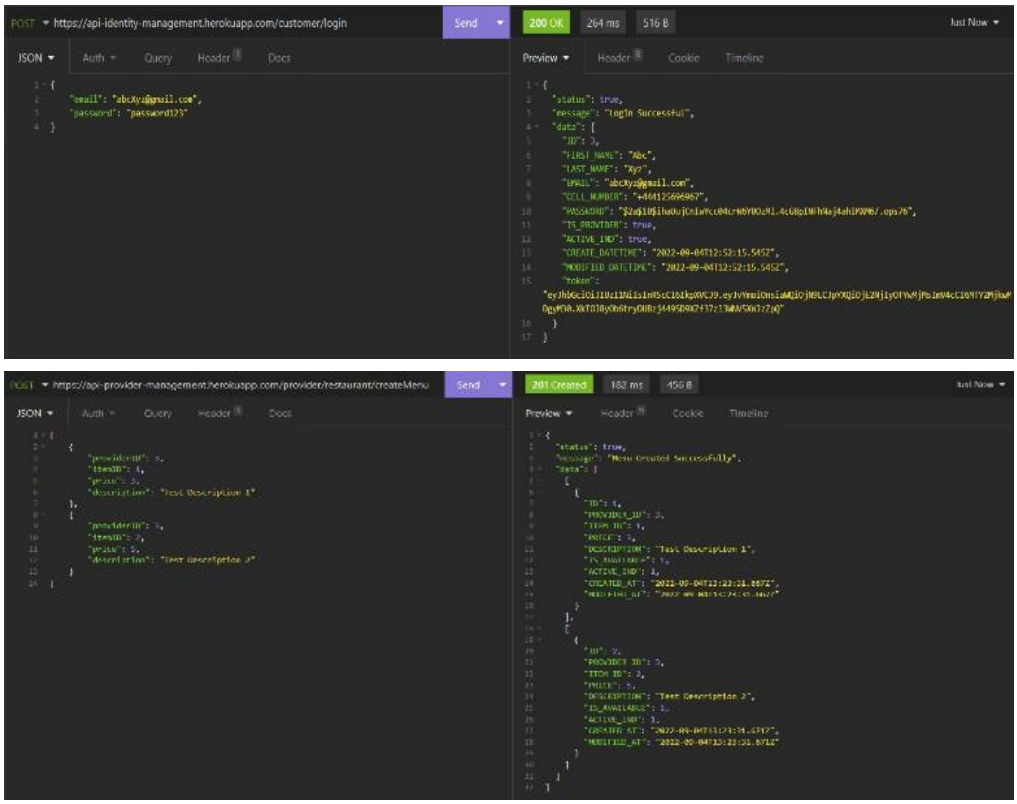
Case	Pass / Fail	Screenshots
1	Pass	

2	Pass	
3	Pass	
4	Pass	
5	Pass	

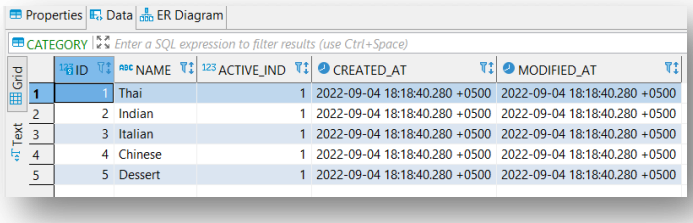
6	Pass	
7	Pass	
8	Pass	

6.2. Server

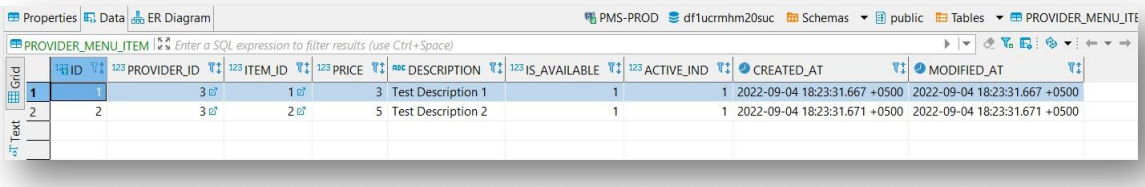
Black box testing was used to test the backend services. Black box testing tests the behaviour of the service with no concern about its implementation. It is done to check whether an application is behaving as expected under different scenarios. Since the backend was developed in two different languages i.e., Java and Node.js, there are some differences in language on the implementation level as each has its own way of doing things. End-to-end testing is done through insomnia. Insomnia is a REST client that provides features like security helpers, environment variables, and an easy to use interface to test APIs. Insomnia was used to test the server by triggering the API by using the JWT Authentication method. Authentication was tested by including the JWT Token inthe API header.



Testing the backend APIs with Insomnia and validating the response



	ID	NAME	ACTIVE_IND	CREATED_AT	MODIFIED_AT
1	1	Thai	1	2022-09-04 18:18:40.280 +0500	2022-09-04 18:18:40.280 +0500
2	2	Indian	1	2022-09-04 18:18:40.280 +0500	2022-09-04 18:18:40.280 +0500
3	3	Italian	1	2022-09-04 18:18:40.280 +0500	2022-09-04 18:18:40.280 +0500
4	4	Chinese	1	2022-09-04 18:18:40.280 +0500	2022-09-04 18:18:40.280 +0500
5	5	Dessert	1	2022-09-04 18:18:40.280 +0500	2022-09-04 18:18:40.280 +0500



	ID	PROVIDER_ID	ITEM_ID	PRICE	DESCRIPTION	IS_AVAILABLE	ACTIVE_IND	CREATED_AT	MODIFIED_AT
1	1	3	1	3	Test Description 1	1	1	2022-09-04 18:23:31.667 +0500	2022-09-04 18:23:31.667 +0500
2	2	3	2	5	Test Description 2	1	1	2022-09-04 18:23:31.671 +0500	2022-09-04 18:23:31.671 +0500

The Postgres database tables of the are tested by monitoring if accurate records are being storedand updated.

7. Evaluation

This application sought to present information to its users to simplify navigating through a website and ordering food, therefore the best method for evaluation is to ask a broad range of people to use and test the application and record the feedback in the form of a questionnaire.

Due to the time constraints of the project, only two days were allocated to perform the activity. During this time, different friends and family were asked to use the app and record their responses. People were carefully picked to not only include phone savvy individuals but also senior citizen that may find it more difficult to use their phone and navigate through different websites.

The following questions were presented:

Q1) How was the overall look and feel of the website?

Q2) How hard or easy was it to navigate through the website?

Q3) What were the features that you particularly liked or disliked on the website.

Most of the reviewers gave positive feedback about the look and feel of the website. It was also noted that non-technical users were more comfortable with using a rather simple interface. On the other hand, some of the more technical users praised the simple design but preferred a more decorated UI. Most of the users found that the website design and its navigation were intuitive.

The most praised of all the included features was the option of including or excluding some ingredients, which made it possible to customise the food according to a customer's liking. It was also noted that, although other services allow a note on the order, often it is either ignored by the restaurant or becomes difficult to manage in case of large orders. Incorporating this option with every food item makes it manageable, easy to add for customers, and easy to view for providers. Several other features were also well received including schedule order, marking a restaurant closed, and the rating system, not only for restaurants but also for food items.

The most repeated recommendation was to develop separate mobile apps for customers and providers. It was also suggested that a mobile app for riders would make it more convenient to track the rider during delivery. Another request was to include credit/debit card support for payment and cashless transactions.

8. Limitations and Recommendations

In this chapter, we will discuss the limitations and future enhancements for the TMS (Take AwayMenu System).

8.1. Limitations

Currently, the TMS is ready to accept orders and fulfill them but it still contains some limitations, which can be addressed in future versions.

- **No Support for Voucher:**

Vouchers play a vital part for any platform to engage its customers. Numerous platforms have benefitted from using this technique for customer engagement. Though this is an important feature to have for a platform, it comes with multiple challenges and is not easy to implement. It requires tons of development to function properly from managing records of each user to classifying users so qualifies to avail them. Unfortunately, TMS doesn't have this feature now because of such complexities and time frame constraints.

- **No Twilio Support for Notifications:**

Customers like to receive real-time notifications that keep them posted about the progress of their order; this also enhances their user experience as well. Although, TMS is designed to be user friendly and interactive it still lacks the support for a notification service. One of the more renowned services that can be integrated with TMS for notifications is Twilio butthe integration could not be performed due to a technical issue with their API thatdeactivated the account.

- **Lack of Delivery Tracking:**

TMS is currently limited to a web application and does not have a mobile application so it is not possible to track the delivery agent. The order is marked as completed once the customer confirms from his account, but it would provide a greater user experience when orders are marked delivered on completion of delivery by the rider.

8.2. Recommendations

While TMS is both useful and effective at solving food ordering challenges, some improvements can be made in the future versions. Some of these recommendations are as follows:

- **TMS Mobile Application:**

The creation of mobile apps has recently experienced tremendous growth. With an increasing number of people using smartphones and tablets to access the Internet, developing mobile apps has the unique capacity to reach a lot of potential customers. There are many advantages to building a platform's mobile application, such as reinforcing the brand, increasing visibility, and increasing accessibility. Mobile apps are at the forefront of the developmental drive as our society continues to transition toward a mobile-centric community. A personalised business mobile app can help that business succeed in the future by putting it in the hands of new customers. Therefore, it is highly recommended TMS develops its own mobile app, which will definitely benefit the overall TMS idea and provide a better user experience. This would also solve one of the limitations, as orders could be tracked as the delivery agents will have the app installed on their devices through which traceability of the order will be possible.

- **Online Payments Functionality:**

Businesses can now enjoy a variety of new advantages and benefits thanks to e-payment systems, providing them the competitive edge they need to stand out. There are several benefits of having online payments some of which are as follows:

  - o Reduced Transaction Cost
  - o Secure
  - o Save Time
  - o Real-time reflection of payments
  - o Complete visibility of all transactions
  - o No risk of fraud

TMS system will incorporate this feature by integrating the payment gateways through which the user can pay online without any hassle to manage the physical cash issues.

- **Reports**

To monitor the business trajectory and see how much revenue it is generating it is important to see and analyse

reports. TMS is here to help solve the service provider and create value in the market, therefore another very important feature in a future version is the ability for providers to generate reports through which they can analyse their business and adjust their strategies accordingly.

9. Conclusions

In today's world we have a lot of applications that claim to provide a platform for users to dobusiness, but since the market is so competitive the newer solutions are just overcrowding the business space without providing actual value. In this project, some major issues of such applications are addressed. This project has successfully created a web application that is providing a feature-rich user experience to customers by putting them in the driver's seat.

The objectives of the customer role were addressed by developing an easy-to-use web application in react.js with bootstrapping using react-bootstrap to let customers view the targeted reviews about a particular item, and schedule orders according to their needs. The most important objective achieved was to let the customer decide which of the ingredients they do not want to include and to show the item specific view on the provider's end so that customer requests are not neglected or ignored.

Moreover, from the provider end, the objectives were accomplished by letting family-run businesses become a part of this system, but also ensuring that only serious businesses that maintain good hygiene standards in their workplace can participate by making government-issued licenses mandatory.

REFERENCES

[1] Archivebay.com. 2022. *OpenZipkin Â· A distributed tracing system - Archived 2022-08-30*. [online] Available at: https://archivebay.com/site/zipkin.io--2020-06-04 11-31-51. [Accessed 30 August 2022].

[2] Australian Competition and Consumer Commission. 2022. *Uber Eats amends its contracts*. [online] Available at: https://www.accc.gov.au/media-release/uber-eats-amends-its-contracts [Accessed 30 August 2022].

[3] Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. Micro-service Architecture: A Performance and Scalability Evaluation. *IEEE Access*, *10*, 20357-20374.

[4] Bushnell, M. (2022, June 29). *Should Your Restaurant Be on Grubhub?* Retrieved August 26, 2022, from https://www.business.com: https://www.business.com/articles/is-grubhub-good-for-restaurants/.

[5] Carson, B. (2022). *Uber's Secret Gold Mine: How Uber Eats Is Turning Into A Billion- Dollar Business To Rival Grubhub*. Forbes. Retrieved August 26, 2022, from https://www.forbes.com/sites/bizcarson/2019/02/06/ubers-secret-gold-mine-how-uber-eats-is-turning-into-a-billion-dollar-business-to-rival-grubhub/?sh=4d38ef311fa9.

[6] Edison trends, 2022. [online] Available at: https://www.trustradius.com/products/edison-trends/pricing [Accessed 30 August 2022].

[7] Fu, G., Zhang, Y., & Yu, G. (2021). A Fair Comparison of Message Queuing Systems. *IEEE Access*, *9*, 421-432. https://doi.org/10.1109/access.2020.3046503.

[8] Hyrecar. (2021, April 6). *What's The Difference Between Seamless And GrubHub?* Retrieved August 26, 2022, from https://www.hyrecar.com: https://www.hyrecar.com/blog/whats-the-difference-between-seamless-and-grubhub/#main-differences-between-grubhub-and-seamless.

[9] Jangla, K. (2019). *Accelerating Development Velocity Using Docker*. Apress.

[10] Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges when moving from monolith to micro-service architecture. *Current Trends In Web Engineering*, 32-47. https://doi.org/10.1007/978-3-319-74433-9_3.

[11] Kamp, L. (2020, November 28). *How DoorDash Built the Most Incredible Go-to-market Playbook Ever*. Retrieved August 26, 2020, from https://larskamp.medium.co: https://larskamp.medium.com/how-doordash-built-the-most-incredible-go-to-market-playbook-ever-5e8f1d58f6cd.

[12] K. Guntupally, R. Devarakonda and K. Kehoe, "Spring Boot based REST API to Improve Data Quality Report Generation for Big Scientific Data: ARM Data Center Example," 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 5328-5329, doi: 10.1109/BigData.2018. 8621924.

[13] Kim, K. (2021). Amazon-induced price discrimination under the Robinson–Patman ACT. *Columbia Law Review*, *121*(6), 160-185.

[14] Kumar, V., Alshazly, H., Idris, S. A., & Bourouis, S. (2021). Evaluating the impact of Covid-19 on society, environment, economy, and education. *Sustainability*, *13*(24), 13642.

[15] Lei, K., Ma, Y., & Tan, Z. (2014). Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. *2014 IEEE 17Th International Conference On Computational Science And Engineering*. Doi: 10.1109/cse.2014.142.

[16] Li, C., Mirosa, M., & Bremer, P. (2020). Review of online food delivery platforms and their impacts on sustainability. *Sustainability*, *12*(14), 5528. https://doi.org/10.3390/su12145528.

[17] Madhu, M., & Sunanda, D. (2019). Distributing messages using Rabbitmq with advanced message exchanges. *International Journal Of Research Studies In Computer Science And Engineering*, *6*(2).https://doi.org/10.20431/2349-4859.0602004.

[18] Mallanna, S., & Devika, M. (2020). Distributed request tracing using Zipkin and Spring Boot Sleuth. *International Journal Of Computer Applications*, *175*(12), 35-37. https://doi.org/10.5120/ijca2020920617.

[19] Mardan, A. (2018). Using Express.js to create Node.js web apps. *Practical Node.Js*, 51-87. doi: 10.1007/978-1-4842-3039-8_2a.

[20] Njunina, V. (2022). *Food Safety Trends 2022: How Has the Food Industry Changed?* Fooddocs.com. Retrieved August 26, 2022, fromhttps://www.fooddocs.com/post/food-safety-tr

ends.

[21] Ozili, P. K., & Arun, T. (2022). Spillover of COVID-19: Impact on the global economy. *Managing Inflation and Supply Chain Disruptions in the GlobalEconomy* (pp. 41-61). IGI Global.

[22] Poon, W. C., & Tung, S. E. H. (2022). The rise of online food delivery culture during theCOVID-19 pandemic: An analysis of intention and its associated risk. *European Journal of Management and Business Economics*.

[23] Potdar, A., Narayan, D. G., Kengond, S., & Mulla, M. (2020). Performance evaluation of Docker container and virtual machine. *Procedia Computer Science*, *171*, 1419-1428. https://doi.org/10.1016/j.procs.2020.04.152.

[24] Rayome, A. (2022). *Best Food Delivery Services for 2022*. CNET. Retrieved August 26, 2022, from https://www.cnet.com/tech/services-and-software/best-food-delivery-service/.

[25] Shah, H., & Soomro, T. (2017). Node.js Challenges in Implementation. *Global Journal Of Computer Science*, *17*(2). Retrieved from https://www.researchgate.net/publication/318310544_Nodejs_Challenges_in_Implementation.

[26] Sharvari, T., & Sowmya, N. (2019). A study on modern Messaging Systems- Kafka, RabbitMQ, and NATS Streaming. Retrieved August 26, 2022, from https://www.researchgate.net/publication/337856004_A_study_on_Modern_Messaging_Systems-_Kafka_RabbitMQ_and_NATS_Streaming.

[27] Ridesharing Driver. (2021, September 14). *Here's The Big Differences Between Postmates vs. Uber Eats!* Retrieved August 26, 2022, from https://www.ridesharingdriver.com: https://www.ridesharingdriver.com/difference-postmates-ubereats/.

[28] Synder, C. (2020, March 19). *Grubhub vs. DoorDash – Which Is Better?* Retrieved August 26, 2022, from https://www.online-tech-tips.com: https://www.online-tech-tips.com/cool-websites/grubhub-vs-doordash-which-is-better/.

[29] Stempel, J. (2022). *Grubhub, Uber Eats, and Postmates must face diners' lawsuits over U.S. restaurant prices*. Reuters. Retrieved August 26, 2022, from https://www.reuters.com/legal/government/grubhub-uber-eats-postmates-must-face-diners-lawsuit-over-us-restaurant-prices-2022-03-30/.

[30] Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80-83. https://doi.org/10.1109/mic.2010.145.

Copyright © 2023 The Author(s). Published by Scientific & Academic Publishing
This work is licensed under the Creative Commons Attribution International License (CC BY). http://creativecommons.org/licenses/by/4.0/