



Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: QRDER

Final Report

Mustafa Oğuz Güngör

Taylan Bartu Yoran

Mehmet Akif Kılıç

Supervisor: Varol Akman

Jury Members: Uğur Güdükbay, Fazlı Can

Innovation Expert: Ahmet Eren Başak

Final Report
Dec 17, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492.

Contents

Introduction	4
Requirements Details	5
Functional Requirements	5
2.1.1 System Functionality	5
2.1.2 User Functionality	5
2.2 Nonfunctional Requirements	6
2.2.1 Extensibility	6
2.2.2 Reliability	6
2.2.3 Usability	7
2.2.4 Accessibility	7
2.2.6 Efficiency	7
Final Architecture and Design Details	7
Overview	8
Restaurant Side	9
Mobile Side	10
Application Architecture & Design Choices	10
Server Architecture & Design Choices	11
3.3.1 REST API	11
3.3.2 Authentication API	11
3.3.3 User API	11
3.3.4 Restaurant API	11
3.3.5 Order API	12
Development and Implementation Details	12
Server	12
Web	13
Mobile	14
Testing Details	15
Version Checking	15
Server Logs	15
API functionality	16
Code functionality	16
Maintenance Plan and Details	16
Limitations of the server	16
Version Checking	17
Change Requests	17
Expanding API	17
Machine Learning Concept	18
3D Food Models	18
Other Project Elements	19
7.1 Consideration of Various Factors in Engineering Design	19
7.2.Ethics and Professional Responsibilities	20
7.3.Judgements and Impacts to Various Contexts	21
7.4 Teamwork Details	22
7.5 Project Plan Observed and Objectives Met	22
7.5.1 System Functionality	22

7.5.2 User Functionality	23
7.5.3 Non-functional Objectives	24
7.6 New Knowledge Acquired and Learning Strategies Used	25
8 Conclusion and Future Work	26
8.1 Conclusion	26
8.2 Future Work	27
User Manual	28
Glossary	49
References	49

1 Introduction

Compared to the last decades, more people prefer eating outside rather than at home. Since there may be many reasons [1] , it can be said making restaurants serve meals fast and good has become harder. Since the number of people who go to well known restaurants increases, restaurant owners try to find new ways to maintain the quality of dishes and restaurant. Since there are some known standards [2] , it may be impossible to meet those when the restaurant is too crowded. Also, these standards are dependent on waiters / waitresses.

Finding a good waiter or making a menu updated and well designed may be costly. So , any owner that wants good service quality and make this sustainable must pay big amounts of money. However, “Qrder” aims to make ordering and payment online so that restaurants may focus on food quality rather than getting order and checking the bill.

The key idea behind the “Qrder” is make ordering, choosing the meal , checking the bill and displaying the menu digital. So that, waiters will have specific missions like delivering the meal instead of being responsible for any mission. To sum up. Qrder automates the order and payment process so that it will reduce the workload of restaurants.

For example, imagine yourself going to a crowded restaurant with your friends. You would waste a lot of time waiting for water to make an order or making payment. When that is the case, Qrder helps you in that case. If you would make an order and payment by using Qrder, you would not have to waste time for ordering and payment. Furthermore, if restaurants let Qrder to be used, their responsibility will be decreased.

This application can be used for different purpose. For example, a restaurant wants to change their menu design anytime they want. For example, when they want to update the price of the meal, they can change it instantly. Another example is changing order. When clients want to change/ cancel the order, they can do it via Qrder. So, to sum up, Qrder let clients and restaurants to changes their decisions instantly.

2 Requirements Details

2.1 Functional Requirements

2.1.1 System Functionality

The system should:

- make the register available for clients/restaurants.
- make a login system for clients/restaurants.
- gather info such as address, phone number and menu for restaurants.
- allow restaurants to create and alter their menu.
- show restaurants that are close to users.
- allow users to look for any restaurant that is close to them.
- show restaurants that meet the needs of the search.
- ask clients to scan the qr code.
- find a corresponding restaurant that matches the qr code.
- display the corresponding menu.
- receive the order.
- inform the restaurant about the order.
- calculates the check.
- inform the user about the check.
- provide online payment.
- work interactively with the diet application.
- provide any interface about advertisements.
- display any special offer.

2.1.2 User Functionality

The user should:

- register the system.

- provide required information while registering.
- login the system.
- scan the qr code.
- choose the meal that s/he wants among the menu.
- look for any restaurant that uses Qrder by searching.
- make any changes about the order.
- look and use special offers.
- use a diet program interactively.
- learn the nutrients of the meal.
- see the information about whole food that is displayed on the menu.
- change it's profile.
- cancel the order.

2.2 Nonfunctional Requirements

2.2.1 Extensibility

The system should:

- be easy to maintain.
- be available on multiple platforms.

2.2.2 Reliability

The system should:

- not store any credit card information unless the user states otherwise.
- be prepared for any possible cyber attacks.
- ensure that the user's data is safe.
- ensure that there will be no fake orders, restaurants and clients.

2.2.3 Usability

The system should:

- be user friendly
- create options in terms of language and theme.
- provide exact pictures of food and accurate info about meals.
- work well with the diet program.

2.2.4 Accessibility

The system should:

- be downloadable for free.
- be downloadable from the official website for the desktop version.
- be downloadable from the App Store or Google Play Sotre for mobile version.

2.2.5 Portability

The system should:

- run in any OS.
- be able to work cross-platform.

2.2.6 Efficiency

The system should:

- not lag when communicating with the server.
- be light.

3 Final Architecture and Design Details

We have used various technologies in order to implement this project. In the Server Side, we have used NodeJS [4]. In the Restaurant Frontend, we have used React [5] and Electron [7]. React provides a dynamic view with the scalability and MVC design

pattern. Electron provides converting the React web project to a desktop project. Hence, with only one project of React we have two builds -both are web and desktop. In the Mobile App, we have used React Native in order to have two builds -both are Android and iOS- with only one project. The reason for using NodeJS and NodeJS based frameworks, React, React Native [6] and Electron, is that it contains tons of libraries and documentaries on the web. Therefore, with such npm and yarn package managers we are able to reach lots of simple and complex libraries and use them in order projects easily without any dependency problems.

3.1 Overview

Since we have 3 different software applications in our project, we will discuss the overview in the perspective of Server Side, Restaurant Side, Mobile Side.

3.1.1 Server Side

We have decided to use Heroku [8] for our server deployments. Heroku is a platform as a service company that provides deployment management, pipelines, resources, configuration variables, buildpacks, etc. We have set up an automatic deployment system that is connected to our git project through GitHub via hooks activated when a commit is pushed to the git project. Our Virtual Machine provided by Heroku has the following specs: 512MB RAM and 2 dynos. Dynos provides run workers, automated computation programs, simultaneously. Since our server has a stateless server architecture with light-weighted operations, we did not require any dynos. We have used the Docker [10] container system in the development part of the server side. Docker runs multiple Virtual Machines and containers at the same time by not giving high pressure to the running OS. Since Docker is highly customizable and scriptable, in order to have the same development environment, it was necessary to use. We are using a NodeJS, a MySQL and a MqTT Broker as containers. Since Heroku does not fully support Docker specifications yet, we have manually activated these containers in the Heroku from the resources provider. We have used free plans for

MySQL and MQTT Broker that provide adequate resources for us -Free MySQL: 5MB storage and 10 simultaneous connections, Free MQTT Broker: a million messages per mount and 20 concurrent connections. In order to provide an api service, we have followed the REST API concept in the server side, HTTP protocols. For the real time data, we have used MQTT Broker instead of Socket because according to research we have done, comparison data claims that MQTT much more efficient by not having overhead as much as Socket and MQTT much more applicable to the IoT kits for the future work. The server side has a 4 different API concepts that are:

- Authentication API
- User API (Only User authenticated)
- Restaurant API (Only Restaurant authenticated)
- Order API (Authentication is required, both auth type has access)

3.1.2 Restaurant Side

We have decided to use Heroku for our server deployments as in the server side. Since Heroku supports React webpack, it is easy to manage deployments. React is a web framework that works on NodeJS environments and developed by Facebook. It provides developers dynamic web projects that can easily communicate with the server with TCP and UDP. Since it works on NodeJS, lots of libraries provided by other developers are accessible via package managers. According to our research, unlike other frameworks such as Vue and Angular, React is much more preferred by projects that must provide higher User Experience. React provides MVC design pattern, however since Model part is received from the server side with predefined structure, it is much more focused on View and Controller parts. For the View (UI) part of the Restaurant Side, we have used a Material UI pack for more elegant and light-weight UI. For the real time data, we have connected the React project to the MQTT Broker with MQTTJS library. Since React framework has an efficient and unique render technology, only updating the state of the corresponding component was enough to represent real time data easily. Moreover, because Electron is efficiently compatible with React

projects, we have used it in order to convert our web project to a desktop project. With using both React and Electron, we have two builds with one project. Therefore, management of both builds are much easier. The main purpose of the Restaurant Side is providing restaurant employees high user experience with less physical attraction.

3.1.3 Mobile Side

React Native is a mobile application framework that converts NodeJS projects to Native mobile applications with high efficiency. It was developed by Facebook and built on the React framework. Therefore, every pros and cons are also valid for the React Native such as lots of libraries by package managers. Since React Native provides developers to build the project for both Android and iOS devices with only one code segment, the project compatibility is increased by it. Like React, it also provides a MVC design pattern and since the Model part is received from the server side, it is much more focused on View and Controller parts. The main purpose of the Mobile Side is providing customers high user experience with less physical attraction and fast response.

3.2 Application Architecture & Design Choices

Earlier mentioned in the Restaurant Side and Mobile Side, our main purpose is to provide a high user experience (ux). Since such projects can be heavily loaded with lots of libraries, computation requirements and complex life cycles, it is hard to have a high ux. Therefore, while designing both sides we have focused on fixing these problems. Since NodeJS is a JavaScript environment, it provides asynchronous and multi-threaded solutions to us. Moreover, in order to do most API requests without not displaying in the UI, React and React Native have been chosen because of their unique render technology. React renders a component, including Pages, when it's state is changed. It has an heuristic algorithm that detects which part of the display must be rendered. With this algorithm, React provides a non-disturbing View part for the project and high user experience by not-showing delays and waits of the synchronous codes.

3.3 Server Architecture & Design Choices

For the server-side, our main goal was creating a reliable server so that it could handle a high amount of users and not allowing fake orders, users, and restaurants.

3.3.1 REST API

REST API is an architectural style for an application program interface that uses HTTP requests to access and use data [3]. Since our application highly relies on the interaction between users and the database, we needed a good API that allows us to transfer only related information. That's why we implemented the REST API for making good communication between server and user

3.3.2 Authentication API

Since reliability was the main case, authentication was one of the main issues for us. That's why we used the Authentication API, we created a token for each user and restaurant so that no fake user and restaurant will be allowed.

3.3.3 User API

User API allows us to show users to see UI that has been created for them. So, users will only see their UI and they will be allowed only user-specified actions. Since we have used tokens that we created with the Authentication API, we achieved to create uniqueness for each user and not letting any unauthorized user action.

3.3.4 Restaurant API

Restaurant API allows us to show users to see UI that has been created for them. So, restaurants will only see their UI and they will be allowed only restaurant-specified actions. Since we have used tokens that we created with the Authentication API, we achieved to create uniqueness for each restaurant and not let any unauthorized user action.

3.3.5 Order API

Order API allows us to check whether order is valid or not. It used both User API and Restaurant API, since creating an order needs both user and restaurant information. By doing that, we only let valid orders to be created and saved in our database.

4 Development and Implementation Details

Implemented subparts and their components were Server, web and Mobile parts. Connections between these subparts are displayed in Figure 1. These 3 were the planned subparts of the Qorder system. However, along the development period; new ideas were added for future plans and convenience of these parts was tested. These parts are

- Machine learning server for recommendation system
- Embedded buzzer program
- Responsive Design of restaurant App for mobile and desktop app

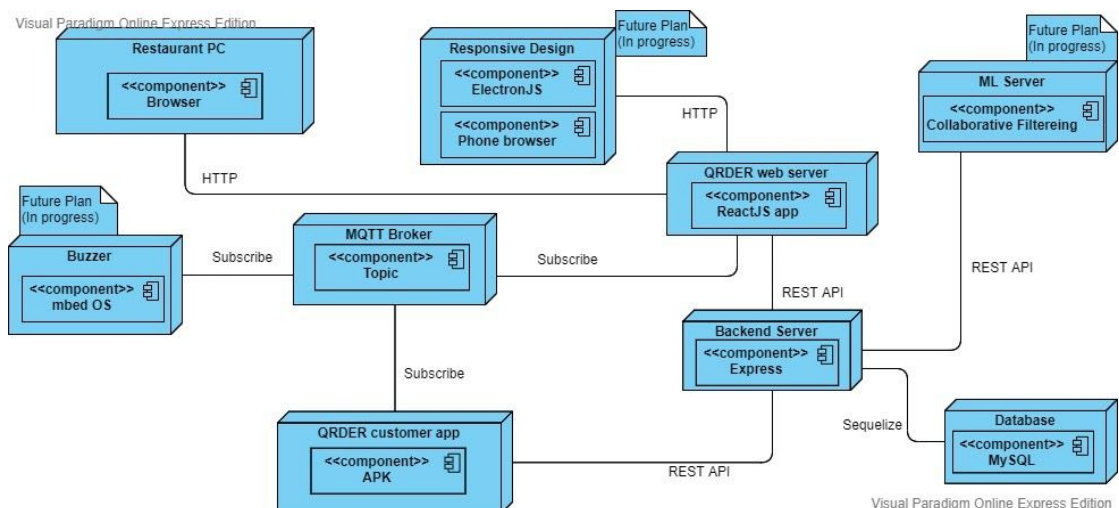


Figure 1 : System Components Diagram

4.1 Server

The server side of Qorder is implemented in JavaScript since it's frameworks provide services that we could use. Furthermore, since we want to make Qorder applicable to each computer, we needed to gain information about Javascript.

We started with writing scripts and config files that initialize Qrder. By doing that, we achieved initializing Qrder for each OS. In addition to that, frameworks and their versions are specified so that we could specify our brand new version and further versions.

After writing config files and scripts, we used node.js for implementation. Since node.js is created for writing server-side applications and it is widely used, we chose it. After choosing that, we gained information with node.js and did some research about it.

The last thing we needed to consider is choosing our database design and implementation. We chose MySQL for the database since we are familiar with it from the courses we took in Bilkent and to initialize it we used the ORM method.

We have created our model, migration, and seeder files. Seeder files included sample objects so that we could test Qrder. After that, we completed and initialized our database design as we stated in our reports.

Lastly, we have implemented a routing system. This allowed us to create our endpoints. Our system had become fully interactive with the user's actions.

Among the implementation, we used GitHub mostly for version checking and we had a meeting weekly so that everyone had an idea about what was going on and everyone was ready to modify the server when needed.

4.2 Web

Web application is developed for restaurant owners to handle requests of the customers at the restaurant. To provide easy access, it is decided to provide desktop application as a web application that resides in a server and can be used via browsers. ReactJS framework is used for implementation due to compatibility with the JSON, better dynamic views with client side rendering , easier component design facilities, various number of compatible CSS design libraries and better dependency handling. In implementation, react-redux module is used to create common states between the components and to make data flow in the application easier and more efficient. Axios module is used to take advantage of RestAPI for communication. Mqtt module is also used to implement notification systems

and to provide access to IOT devices. For the component designs, Material UI module is used with application specific themes to provide uniqueness to the design.

This application consists of two main parts, which are view and controller. View part is designed by creating children of "React.Component" class and rendering JSX structures inside the view. Controller part is the part where redux and component states are updated to set front end data for adjusting view. Also requests are sent and responses are parsed in the controller part. There is no model for MVC structure, because the application data is taken from backend endpoints. Individual data resides in the backend and can be modified via Restaurant API.

Along the implementation, github is used for version checking and source code sharing. With this approach, preserving last versions of the implementation are preserved and modifications on the code can be traced between updates. Heroku is used as a server to deploy the system and github is used to keep server source code repository for the deploy.

4.3 Mobile

Mobile application, QorderApp, is developed for customers to provide high user experience and less physical attractions. As mentioned early, it is developed with React Native mobile application framework on the NodeJS environment. We have used both emulators that are provided by Android Studio and real Android devices in order to run our application. The reason for being able to convert web project code structure to native code is using native development kits. In order to build React Native projects for Android, SDKs provided by Android Studio and Google are necessary. For iOS builds, SDKs provided only by xCode and a MacOS are necessary.

This application consists of two main parts, which are view and controller. View part is designed from the stretch. Except for the QR Scanner library, no other UI library is used. Since React Native is basically a web project that is converted to native projects, CSS like

language is also available to use. For having responsive design, flex-box technology is used. Flex-box provides more efficient percentage allocation and alignment than core CSS. In order to have a responsive design that looks almost the same in every device, emulators with various dimensions are used.

Every component in the React Native has a self-contained state -early mentioned, render is triggered by state changes. Therefore, state management is one of the significant concepts for the lifecycle of the application, Controller part. Instead of using a storelike library such as Redux, built-in components are used in order to provide a light-weight application. QorderApp has 3 main flows -Auth flow, Main flow and Order flow. These all flows have their own self-contained navigation system. Therefore, possible navigation bugs such as mistakenly flow change is prevented with this.

The less mentioned part, Model part of the MVC, it received from the server side by using HTTP request library. Since server side responses have a common structure design, it works like an object oriented Model part in the mobile application.

Hence, development of this like project comes with lots of new components, life cycle, data structure, API connection, utility functions, libraries and etc, a version checking was necessary and lifesaver, therefore, GitHub is used for git versioning.

5 Testing Details

Main area of the Qorder project is web development, therefore testing tools and approaches have to be included in the development cycle. In this manner, appropriate logging and testing tools have to be used to manage a better development phase. Main testing areas and corresponding tools with strategies are given below.

5.1 Version Checking

There are different sides in the project that are dependent on each other. This means that new modifications on the existing system causes crashes and bugs with the new deploys. Therefore, Github is used along the development process to determine the differences from the last versions and turn back to the older versions if it is necessary. For the project, the backend and both of the frontends were kept in different repositories to keep track of a bug or dysfunctionality. So that, when there is a problem in the system after an update; dysfunctionality can be recovered by deploying earlier versions of the corresponding part of the system.

5.2 Server Logs

In Order, there are 2 server machines that run continuously. Therefore, developers should be notified when there is an error on the system and functionalities should be able to be checked after code is deployed to the system. For this requirement, the Heroku control panel was used in the development process. Logs are saved on the heroku system and if a server faces a crash, email is sent to the members of the project. Interfaces to keep track of heroku activities and logs of heroku given in Figure 2. Deployment procedure and steps can be tested via these interfaces.

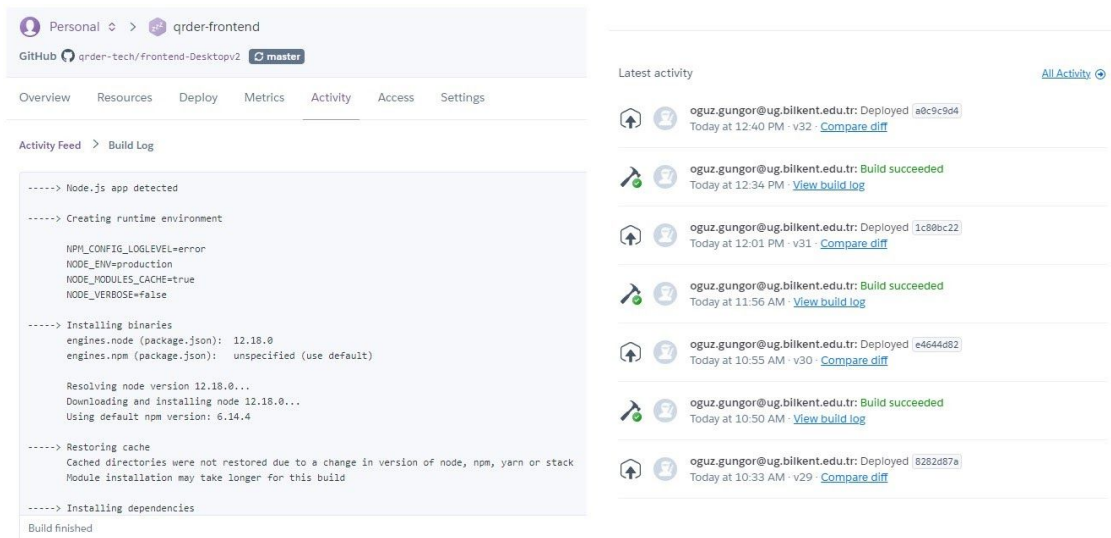


Figure 2: Heroku build log and activity screen

5.3 API functionality

Most of the Qorder system consist of APIs of the subsystems and they communicate with each other with requests. Therefore, endpoints and their responses have to be checked repeatedly before implementing a service related to API. To handle this struggle, the Postman tool is used with a team workspace. Through the postman, modifications in the API can be tested with specified request structures and because a workspace is used in this period, each member in the project was able to observe the changes on the APIs.

To check specified endpoints, followings can be set via postman interface:

- Request type(GET,POST,DELETE, etc.)
- Request address/url
- Request body(url parameters, rest body parameters, etc)
- Authorization parameters(token,session,etc.)

An example request from Qorder system and corresponding response body system can be seen in Figure 3.

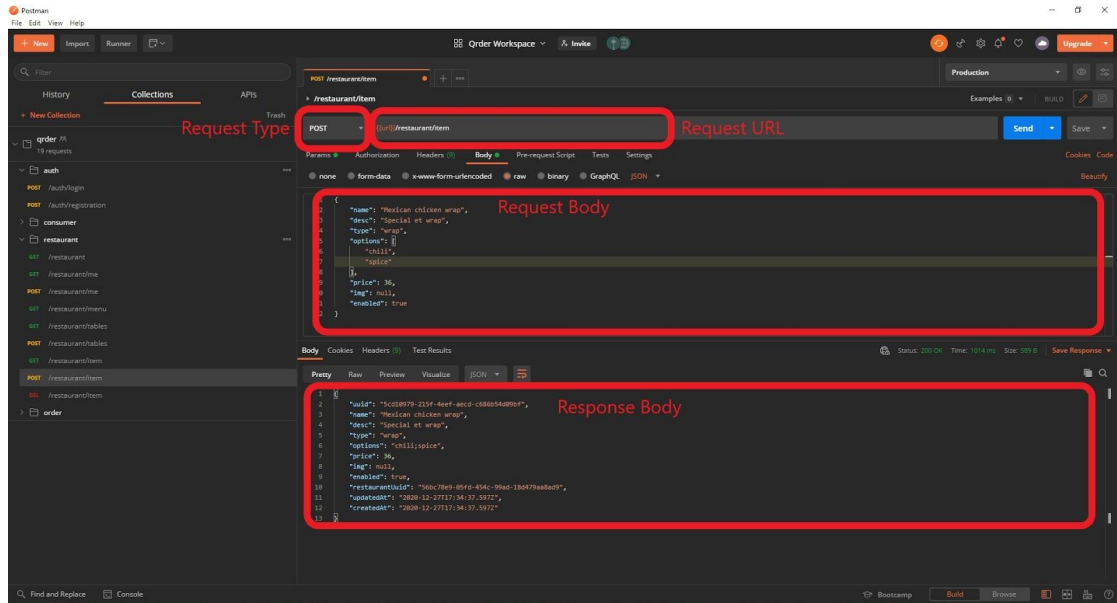


Figure 3: Example Postman Interface

5.4 Code functionality

All of the three sides of the project include modular functional components to manage specified tasks. Therefore, correctness of the results of these components should be checked independently from the system to determine whether the problem is in the components or in the system integration strategy. For the test these distinct tasks, unit testings are done to these components.

6 Maintenance Plan and Details

Maintenance will be shaped with respect to new features that are added to the system. Limitations of the system is another problem that should be addressed in this section.

6.1 Limitations of the server

Back-end server of the Qorder system runs on a server machine and can serve various users at the moment. Because of the cost considerations, this system is deployed on Heroku and is serving from this address. This system has weak support in terms of

memory and bandwidth for free accounts because such accounts were provided for startups. Therefore, if there are a high number of requests that are sent to the server, these requests can not be handled by server and increase of data in the database may cause crash due to running out of memory.

MQTT servers are also free services that are provided for startups, so there is a strong limitation in MQTT server in terms of available number of users that can subscribe to the brokers. Bandwidth limitation is a concern for MQTT servers as well.

Current approach through these limitations is to use these systems because application is not in the market yet and does not have a high number of users. However, Before the Qorder gets on the market with its first release, these limitations should be handled by using new servers that provide more capacity for the program. Therefore, the number of people that uses Qorder will decide the maintenance plan for our project.

6.2 Version Checking

In the development process, Github is used to share source codes and manage version checking for updates. For the future development period, it is planned to maintain this approach and to add new branches to these repositories so that managing correlational work on the project and version checking will be easier.

6.3 Change Requests

After the application takes place in the market, there will be user feedback about bugs, suggestions and critiques. These change requests will be handled with agile methodologies in further iterations of the project. New releases will be planned according to change requests that come from end users and will be developed in agile sprints as new versions of the system.

6.4 Expanding API

Qorder uses MQTT server and HTTP based REST API for communications. Therefore, these protocols can be used to expand the scope of the application by providing usability from other applications and devices. To manage this service, specific APIs will be created for authorized and unauthorized access to the system to use services and data in the system for new projects and specific purpose embedded system devices. For now, planned APIs are the web API to provide a service from Qorder system for new web and mobile applications; and embed API to provide service for IOT devices.

6.5 Machine Learning Concept

Because there will be a lot of data in the system, it is planned to use them in machine learning models to provide strong functionalities for the end user. For now, it is planned to create collaborative filtering model that uses data of Qorder system to train itself. Expected use cases of this algorithm is to provide recommendation for the customers of the restaurants, and to provide marketing suggestions for investors.

6.6 3D Food Models

For marketing purposes, it is planned to use graphic rendering models and APIs for foods of the restaurants to create realistic 3D models of the foods for customers of the restaurant. Javascript is used to implement Qorder system, therefore WebGL library can be integrated to the system for this purpose. However, efficiency issues should be handled to prevent bad end user experience. To create models ;either a specific easy to use service should be provided to the restaurants to design their models, or an API should be implemented to run compatible with model creating programs like AutoCAD.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

There are some factors that may affect Qrder. Below, brief discussion about these factors can be seen.

Public Health

We think that there is no correlation between the usage of Qrder and public health. For the customer side, since Qrder is dedicated and simple, users will use it only when they want to eat out instead of their home. Therefore, people will use Qrder in a small time. So, Qrder does not require a high amount of time. For the restaurant side, we tried to make the user interface simple and understandable. So, our goal was to make everything simpler for restaurants. To sum up, since Qrder is a dedicated application that dedicates to an online meal ordering and paying system and since it will be used only when it is needed, it will not affect people's mental and physical health.

Public Safety

To make Qrder safe for public usage, online payment systems and user privacy are considered. For online payment, we will not store any credit card information. This payment process will be handled by third-party companies and Qrder will be the bridge between them and users. To make it safer, a well known and big company will be chosen. In addition to that, how frequently does the company obey the regulations will be considered. Secondly, Since the user's information will be stored on Qrder's database, it will be hashed and will not be published to any company or person.

Public Welfare

A discussion on public welfare does not apply to our context because Qrder will be free to use and it will have no paid services. So that any customer who has a mobile phone that is connected to the Internet can use Qrder. For restaurants, owners can integrate their restaurants to the Qrders easily since it can be handled easily and no special initialization is

needed. To sum up, Qrder will be free to use for whole restaurants and users and any restaurant will not be considered by their budget or customer count.

Global Factors

To make Qrder fully integrated into today's world, worldwide regulations will be strictly followed. Therefore, as GDPR stated, Qrder will not do any unauthorized operation with user information and will not publish them. In addition to that, security protocols for online payment will be strictly followed. These factors will not only make Qrder globally usable but also make Qrder secure.

Cultural Factors

Since we want to make Qrder a worldwide wide application and each culture has its own dish and design, we want to make Qrder be applicable for whole cultures. So, instead of forcing restaurants to use the standard menu and layout design, Qrder will let restaurants create their menu and layout. So that no one will doubt about losing their local features. In addition to that, images of each meal and drink can be shown on the menu, so that tourists that have no idea about the dishes will have an idea. In addition to that, Qrder will support different language models so that communication problems between users and restaurants will be minimized.

Social Factors

Since Qrder is free to use, requires a little amount of time and dedicated application and any kind of person can use it, we could not find any social factor that is related to Qrder. So, the social factors are not applicable to Qrder.

7.2.Ethics and Professional Responsibilities

About global impact, we want to make Qrder to be used by everyone. To make it possible, we will let restaurants create their own menu and layout design for different cultures. So that no culture will be discriminated against and each culture will be represented. In addition to that, we will have language support so people from different countries can use

Qrder. Since Qrder is free to use, any restaurant and person can use it. Our main goal is making the eating out process easier for restaurants and customers for the whole world.

Regarding economical impact, since there is no application that we can compare with Qrder, we do not know how much Qrder profits. Furthermore, we could not find any value that represents total market value in the digital food industry. Since Qrder is free to use and its services will be free, in the future, we may need to add some advertisement to our application to compensate for the server cost. In addition to server cost, we will need some resources for Google Play Store and App store, however, if Qrder is used widely and if we put an advertisement on Qrder, these three constraints can be afforded easily.

The last impact we will touch on is the social impact. The biggest issue regarding ethics is the issue of user privacy and the safety of online payment. Hence Qrder will not keep any credit card information on its database and this process will be handled by third-party companies that strictly follow the regulations. Furthermore, to make user information private, whole information about users will be hashed and will not be published to third companies and persons.

As explained above, Qrder has nothing to do with public health, public welfare and environmental context so that they will not be discussed.

7.3.Judgements and Impacts to Various Contexts

	Impact Level (out of 10)	Impact
Impact in Global Context	3	make Qrder easier and more accessible
Impact in economic Context	5	make it free
Impact in Social Context	8	caring user privacy and work with third companies for online payment

7.4 Teamwork Details

Taylan Bartu Yoran: Took part in mobile side and back-end. He also prepared the UI interface for the mobile side. In addition to that, he created the models.

Mustafa Oğuz Güngör: Took part in front-end. He prepared the UI interface for the mobile side.

Mehmet Akif Kılıç: Took part in back-end. He also prepared endpoints, models, migrations and seeders that are specified .

Note:

All group members helped with bug-solving, code refactoring and report preparation. Although there was a primary focus for each member, everyone helped each other and took part in each other's work.

7.5 Project Plan Observed and Objectives Met

In this section, we will state the requirements that we have met. System related requirements and user specific requirements will be explained separately.

7.5.1 System Functionality

Now the system is able to:

- make register for clients and restaurants
- make a login system for clients and restaurants.
- gather information such as address, phone number and menu for restaurants.
- allow restaurants to create and alter their menu.
- show restaurants that are close to users.
- allow the user to look for any restaurant that is close to s/he.
- show restaurants that meet the needs of the search.
- ask clients to scan the qr code.
- find a corresponding restaurant that matches the qr code.

- display the corresponding menu.
- receive the order.
- inform the restaurant about the order.
- calculates the check.
- inform the user about the check.
- provide online payment.
- display any special offer (if any)
- create space for advertisement.

7.5.2 User Functionality

The user is able to:

- register the system.
- provide required information while registering.
- login the system.
- scan the qr code.
- choose the meal s/he among the menu.
- look for any restaurant that uses QOrder app by searching.
- make any change about the order.
- look and use for special offers.
- learn the nutritions of the meal.
- see the information about whole food that are displayed on the menu.
- change his/her profile.
- cancel the order.

7.5.3 Non-functional Objectives

In this section, non-functional requirements that are met will be discussed. To make them good, whole members have put a great effort. Below you can find information about Order's non-functional objectives

7.5.3.1 Extensibility

The system is able to:

- easy to maintain.
- available on multiple platforms.

7.5.3.2 Reliability

The system is able to:

- handle online payment processes without storing credit card information.
- hash whole information so that the system has prepared for any possible attack.
- ensure that the user's data is safe.
- ensure that there will be no fake orders, restaurants and clients.

7.5.3.3 Usability

The system is able to:

- being user friendly.
- create options in terms of language and theme.
- provide exact pictures of food and accurate information about meals.
- work well with the diet program.

7.5.3.4 Accessibility

The system is able to:

- be downloadable for free.
- be downloadable from the official website for the desktop version.

- be downloadable from the App Store or Google Play Store for the mobile version.

7.5.3.5 Portability

The system is able to:

- run in any Os.
- work cross-platform.

7.5.3.6 Efficiency

The system is able to:

- provide a fast communication between user and server.
- being light.

7.6 New Knowledge Acquired and Learning Strategies Used

Although we know web and mobile development, we did not have enough experience to implement Order from zero. Therefore, before implementation, we acquired some knowledge. The topics that we acquired knowledge are:

- Mobile Development
- Web Development
- Application Design
- Software Development Planning
- Software Architecture Planning

Since we wanted to implement a good application, we needed in-depth knowledge about web and mobile development. As we knew from Bilkent, good implementation requires a good design. So we learned a lot about software architecture since we did not want to change our system constantly. Lastly, since this project had a time limitation as whole projects in Bilkent, and we had other assignments to do, we needed a good development plan, therefore we gained information about software development planning.

Below, the methods that we used to learn things listed above is listed:

- Literature Review
- Online Learning
- Hands-on Experience

Literature review was essential for us to discover new frameworks, architectures and their working principles and some problems that we face. We used online learning for learning new programming languages and usage of frameworks that we have learned such as JavaScript, node.js, React and electron.js. Lastly, hands on experience was the best method for us because we have learned a lot from our mistakes.

8 Conclusion and Future Work

8.1 Conclusion

To conclude, we created the app that we began last semester. Order runs perfectly. It can add users, orders, and restaurants to its database and allow users to meet the requirements that we listed in section 7. It is free to use and we plan to make it open source so other people can learn a lot from our project about frameworks, programming languages, and design patterns that we used.

There was a positive correlation between the app's progress and our coding, design, testing and development, and friendship between us. We need to get better certainly but we saw that if you put enough effort you will be better at what you are doing. We would appreciate it if we have any feedback.

8.2 Future Work

First, we are going to receive feedback from our professor and colleagues. After that, we want to receive feedback from people in the digital food industry. Then we will change our app concerning feedback. Then we will send our app to the App Store and Google PlayStore.

Since we are experienced now, and we have become a great team, we will create new apps together and we will make sure that Order free to use and safe.

8 User Manual

Restaurant

Login Page

Login Page for restaurant to login system with a registered account. Credentials of the restaurant account should be given to be logged in to the system and to use functionalities. After credentials are entered to the related text boxes, users should click the login button to login. Sign up button redirects users to the sign up page to create a new restaurant account.

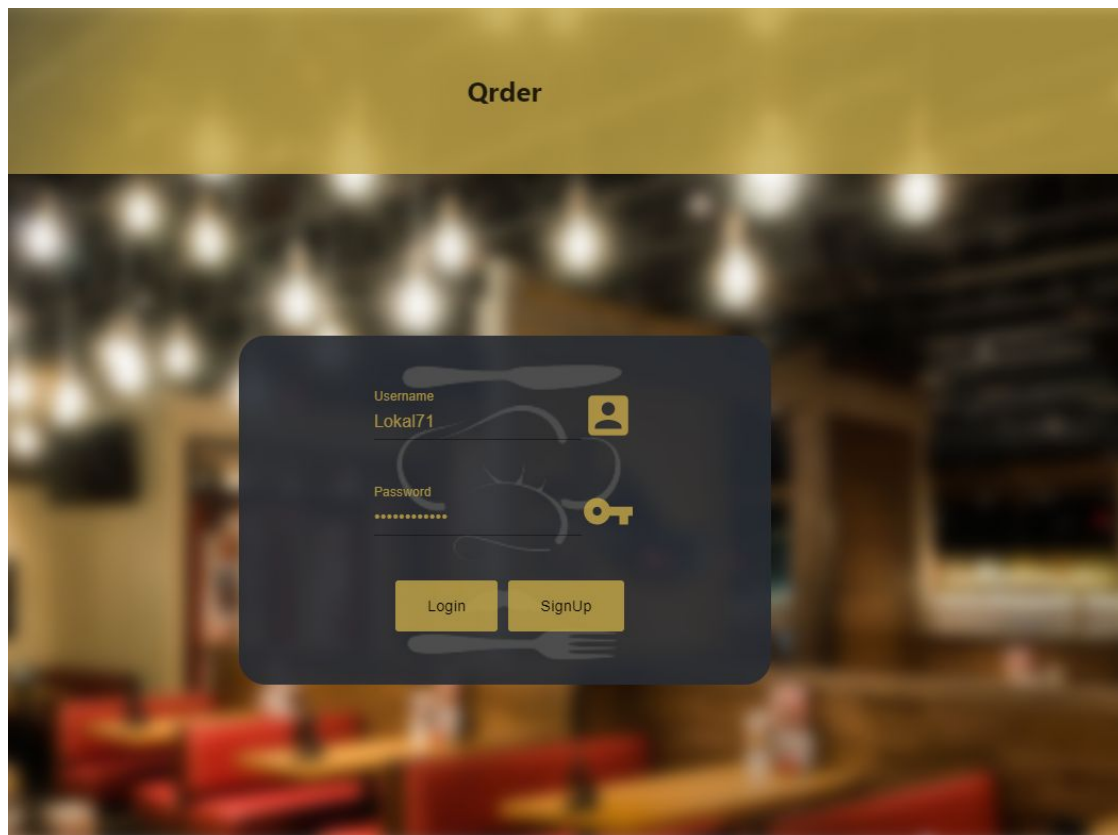


Figure 4: Login Page

Register Page

New restaurants can register the system with required credentials. These credentials are restaurant name, address, phone number, email, username, password and serve type. serve types are self-service and normal. Username is used to be logged in to the system and restaurant name will be seen by the customers. After the user fills required credentials in the text fields, registration can be completed by clicking the register button. Back button redirects users to the login page.

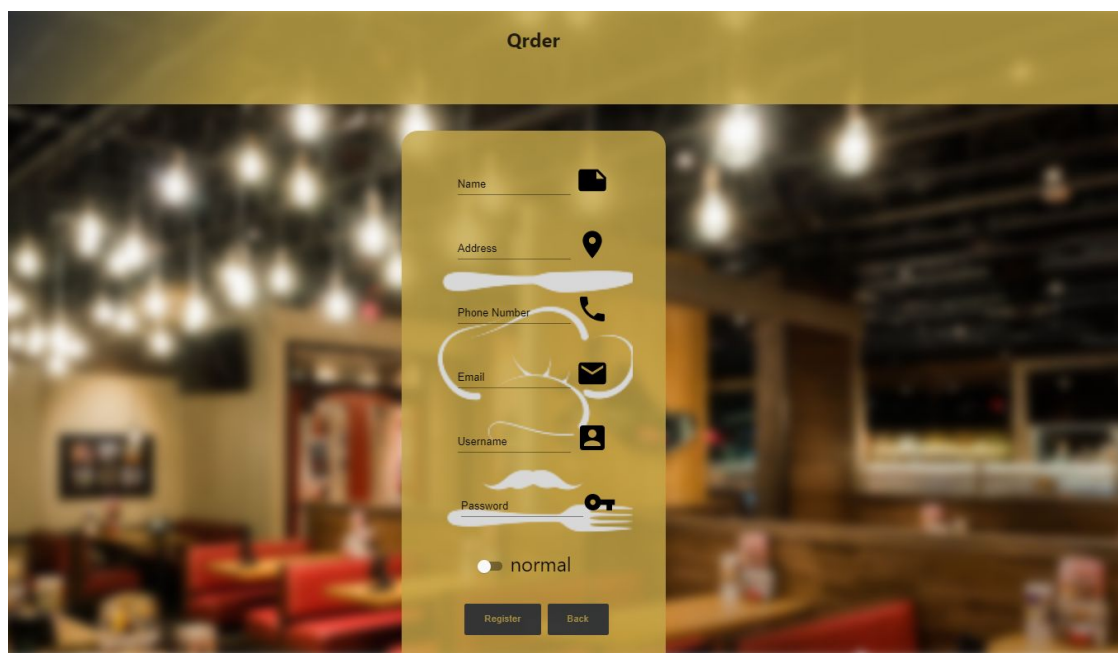


Figure 5: Register Page

Main Frame and Drawers

Table panel and order panel are the main panels for applications. After the user logged in as a normal restaurant, the system redirects the user to the tables panel. These panels can be switched by tab buttons at the header. Drawers are another tool to be redirected inside the application. For Left Drawer, Edit redirects the user to the profile edit panel, Menu redirects the user to the menu panel, AddItem redirects to the item adding panel and AddOrder redirects to the adding order panel. Right Panel is used just for notifications and information.

Table Panel

At the top of the table panel, there is a text field and button to add a new table to the restaurant in the system. Below that, tables are displayed with notification attributes. Each table is colored with respect to time that passed after order was given. Initial color of the table is green and as time passes, it turns yellow and finally red. When 45 minutes past after that order was given, the table becomes fully red; so it is the upper bound that is determined for an order to be served. This time value is the most delayed order time of the table. If the table is not in a wait state, tables are colored with default theme color. There are 4 notifiers for each table, which are hourglass, waiter, check sign and wallet sign. Hourglass shows that the table has a waiting order, waiter shows the waiter was called by the customers of that table, check sign shows order is served but is not paid yet and the wallet sign shows customers of the table want to manage payment.

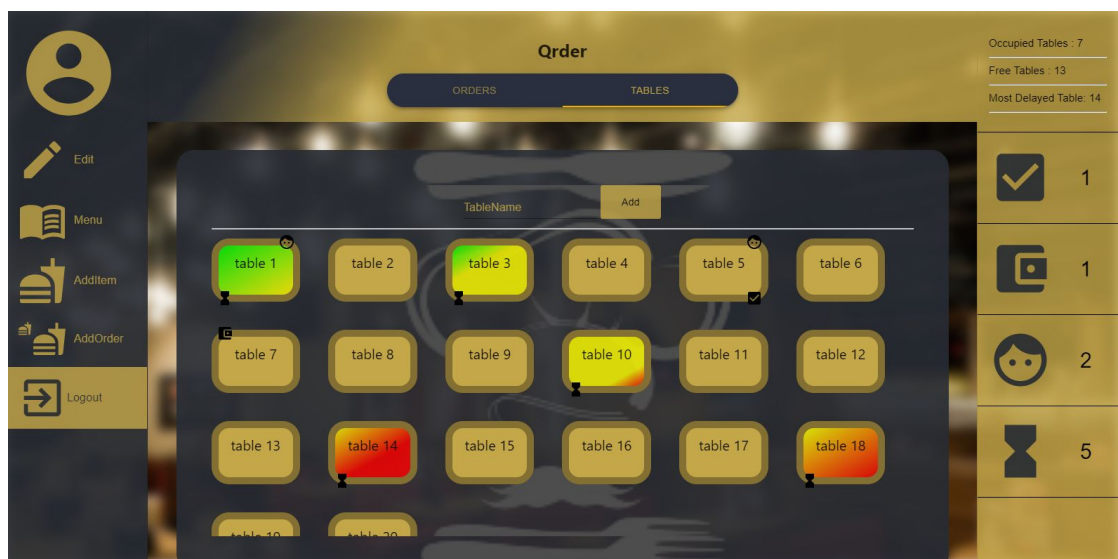


Figure 6: Table Panel

Order Panel

Order panel shows the most recent orders of the restaurant. Active orders are sorted descending respect to time that is passed after the order was given, where the following past orders are sorted in the same manner, but in ascending way. Time is shown for the active orders and status of the orders is also notified with the same icons that are described in the table panel. Specific order numbers are another attribute of the table to distinguish orders.

Order No	Time	Status
7	38	
6	33	
4	25	
3	18	
5	-	
2	-	

Occupied Tables : 6

Free Tables : 14

Most Delayed Table: 14

1

1

2

4

Figure 7: Order Panel

Menu Panel

Menu panel shows the foods of the restaurant. These foods are classified with respect to their types. A new type can be added with the text field and button that resides at the top of the panel. Image, name and price attributes of the foods are shown in the table. There is a cross sign at the right part of the elements to remove the corresponding item from the menu. Also by clicking on the items, users are being redirected to the item details panel.

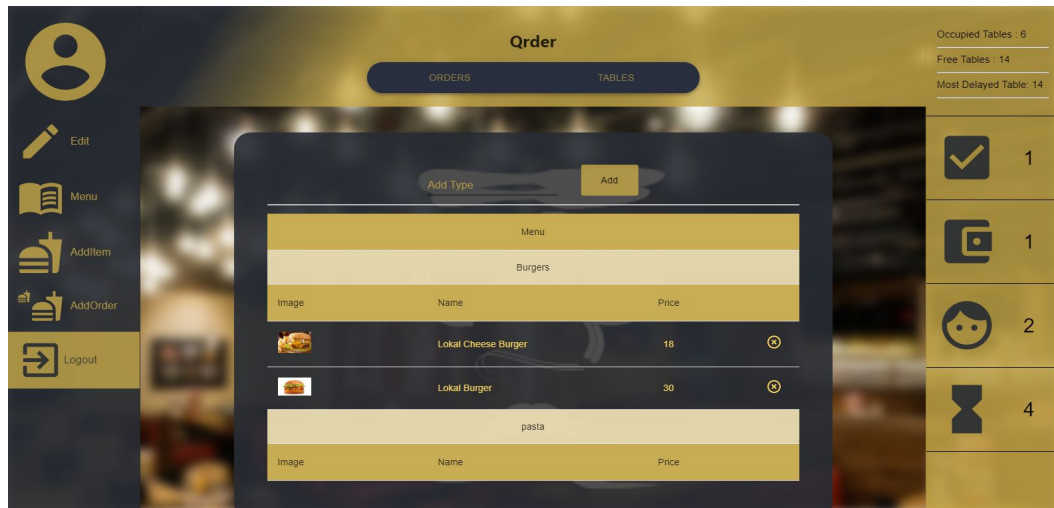


Figure 8: Menu Panel

Item Detail Panel

In the item detail panel, attributes of the food are shown at the right side and image of the item is shown at the left side. This panel only consists of current attributes of the item, and an edit button to be redirected to the item update panel to edit the represented item.

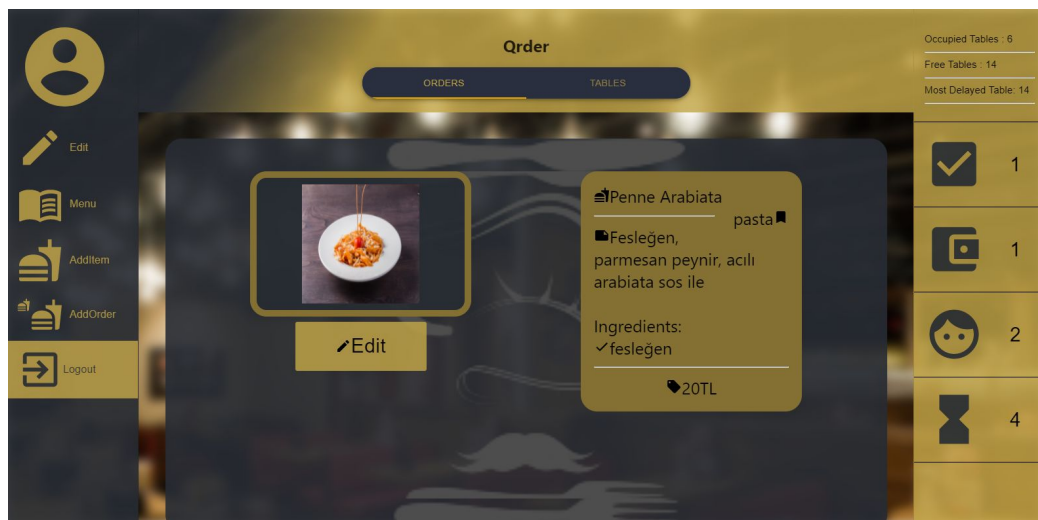


Figure 9: Item Detail Panel

Item Adding/Updating Panel

In this panel, new items are created or existing ones are modified. If a user was redirected to this panel by clicking an item in the menu, then required fields are displayed with the current values. If a user was redirected to this panel by clicking the AddItem Button on the drawer, then these fields are displayed as empty. When a user changes these fields to modify the existing item or to create a new one, a preview of the requested item is displayed before the request is sent. And then, when the user clicks the add button(it is displayed as an update when item is being updated), item request is sent to the system and item with given preview is placed to the menu.

The screenshot shows a web application interface for managing restaurant items. The main header is 'Order' with tabs for 'ORDERS' and 'TABLES'. On the left is a sidebar with icons for 'Edit', 'Menu', 'AddItem', 'AddOrder', and 'Logout'. The central area displays a form for adding or updating an item. The form includes a preview image of a dish, a description, and a list of optional items. The right sidebar shows table status: 'Occupied Tables : 6', 'Free Tables : 14', and 'Most Delayed Table: 14'. Below this is a list of items with checkboxes and counts.

Item	Count
✓	1
☐	1
☐	2
☐	4

Figure 10: Item Adding/Updating Panel

Order Adding/Updating Panel

Main scope of the project requires that orders have to be given with the mobile app. However, System also should be integrated to the current system for satisfying requirements as they are done in the current market. Therefore, There is a panel to add a new order or update an existing one from the restaurant application. In this panel, items can be added from the menu in the left part of the panel, and preview of the order is shown at the right side. Tables have to be chosen in normal restaurants where for the self service panel, there is no selection panel for tables because there is no table. Orders can be removed by clicking the “Remove” button and updated or created with the “Done” button.

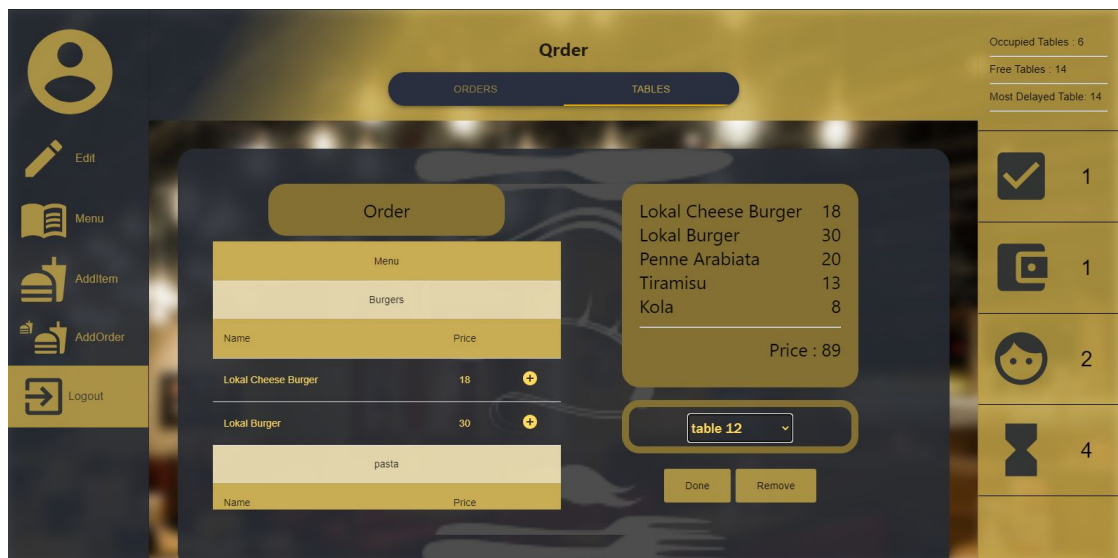


Figure 11: Order Adding/Updating Panel

Table Detail Panel

In the table detail panel, orders and requested services of the specified table are shown. Notification icons are described in the previous panels. Different from notification panels, there are edit buttons in this panel to manage modifications on the services or orders from restaurant account. Also restaurant user can be redirected to the order create panel by clicking the “Add Order” button and specified table can be removed from the system by clicking “Remove Table” button.

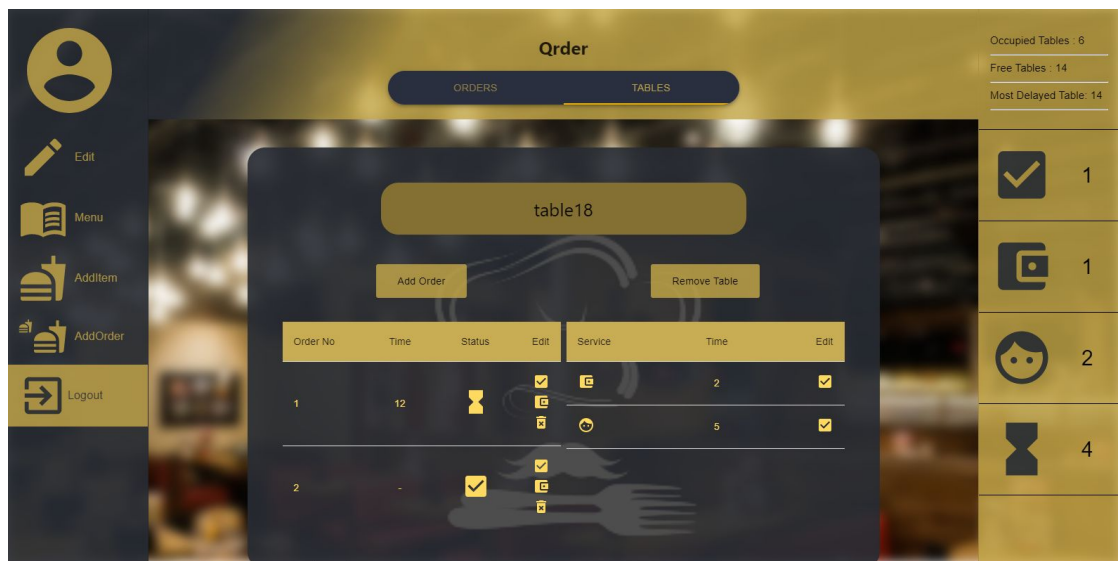


Figure 12: Table Detail Panel

Qrder App (Mobile)

Welcome Page (*Auth Flow*)

Welcome page is the first page that welcomes the new users. It provides users the ability to navigate between Login and Registration pages with the buttons below.

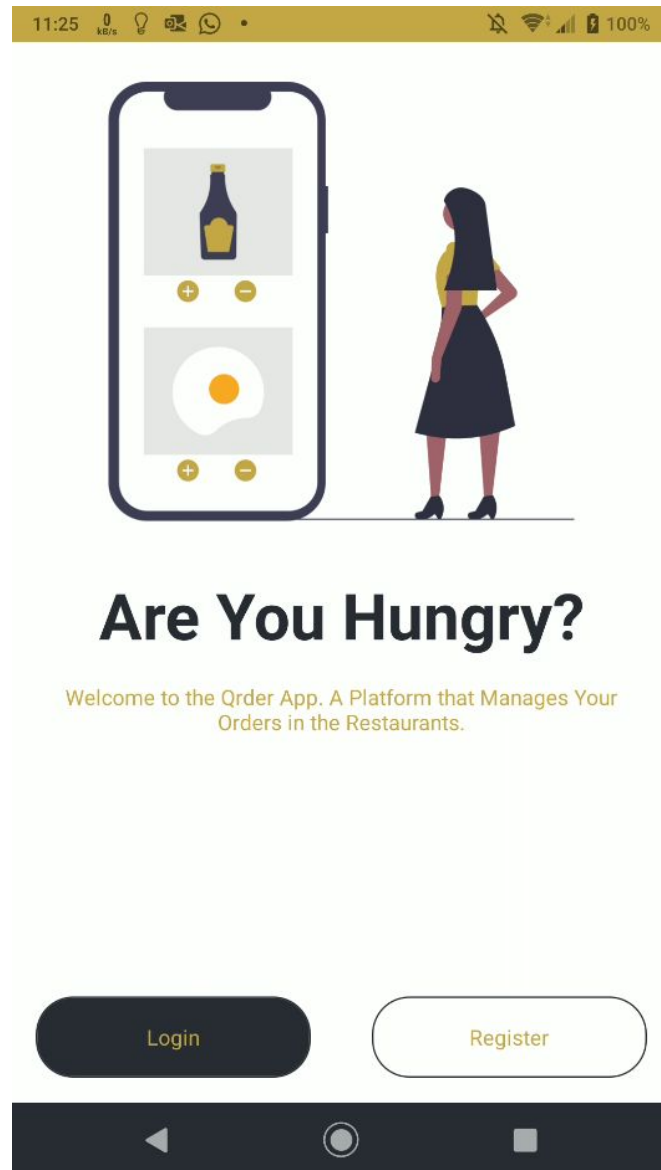


Figure 13: Welcome Page

Login Page (*Auth Flow*)

Login Page provides users to login with their accounts in order to start giving orders. There are two input areas that must be filled by the user with correct identity information to use the program with specified authorization. If the user did not register yet, should register by going back to the Welcome Page and navigate to the Registration Page in order to sign up. If a user forgets password, should click the forgot password button to get his/her new password to his/her specified email address. If a user has already registered, can login to use the program after filling required input areas. After the login is done, users will be redirected to Main flow.

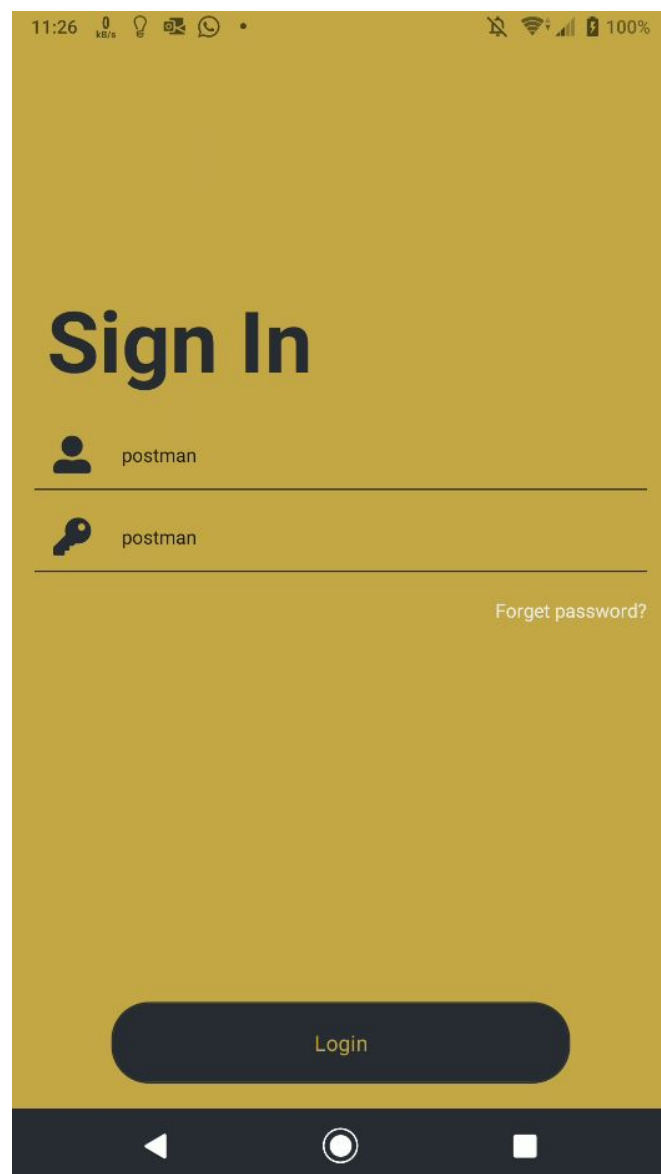
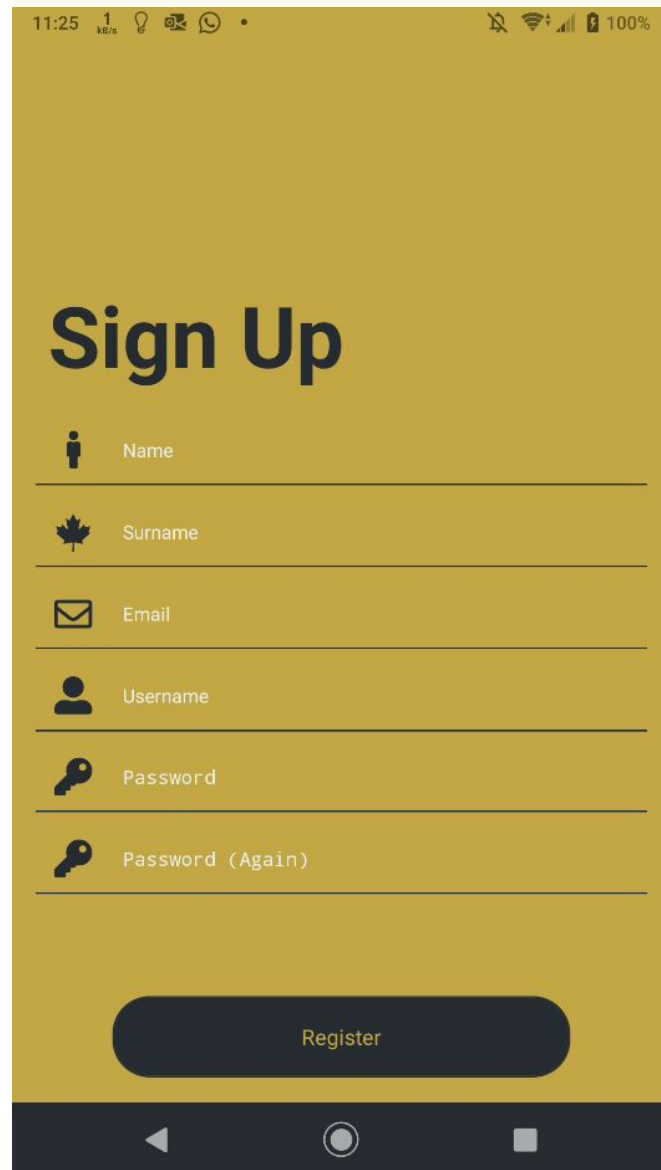


Figure 14: Login Page


Registration Page (*Auth Flow*)


Registration Page provides new users to register to the system in order to use the app. Name, surname, email, username and passwords are the required areas to register. Once a user completes to required areas, can login with username and password that are used in registration.





11:25 1 KB/s 100%


Sign Up


 Name

 Surname

 Email

 Username

 Password

 Password (Again)

Register

Figure 15: Registration Page

Home Page (Main Flow)

The Home Page is the first page of the Main flow that is seen after the login. At the top, there is a banner as a slider box that includes ads of registered restaurants. Under it, there is a container that shows the last 2 orders' information -rank, restaurant title, order relative time. After that, there is a container that shows 3 of the favourite restaurants' information -rank, restaurant title. At the bottom, there is a tab navigation bar that provides users to navigate different pages.

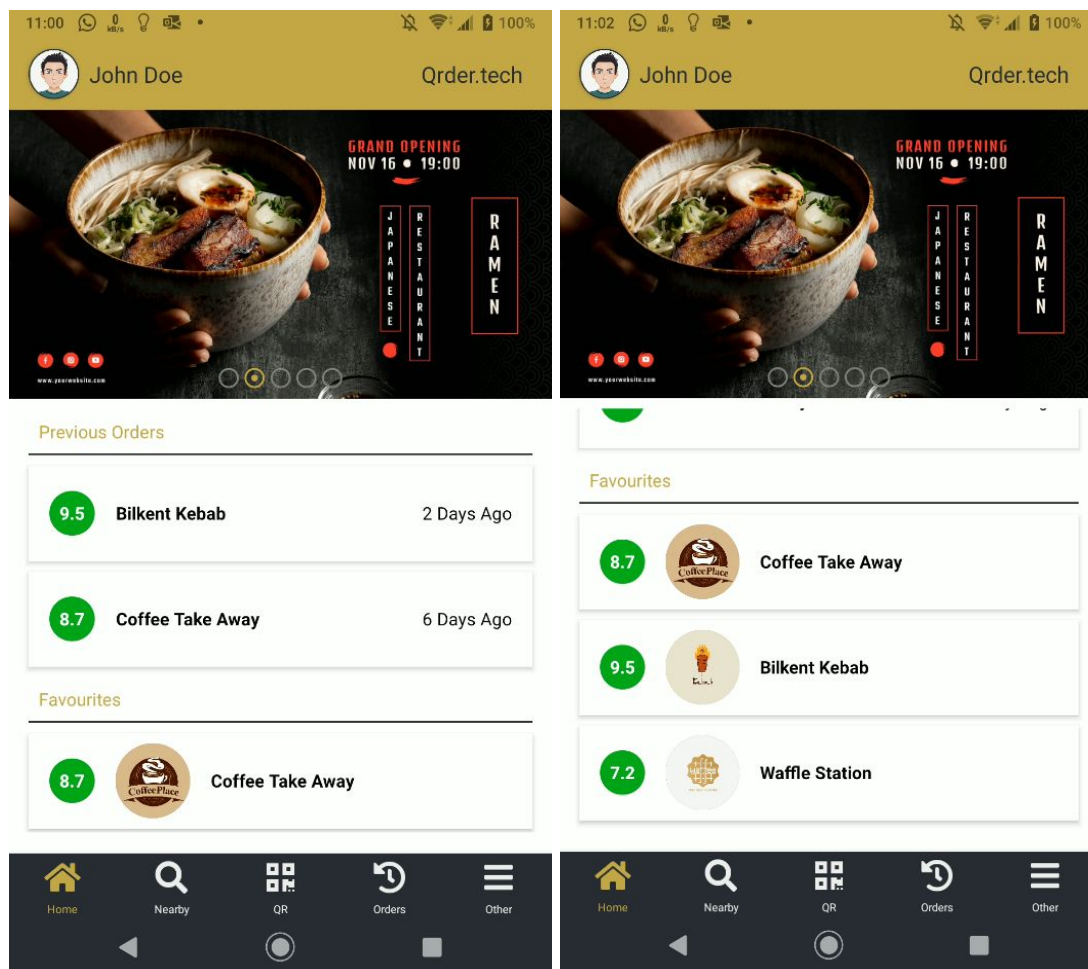


Figure 16: Home Page

Drawer (*Main Flow*)

Drawer is the navigation menu page that provides users to navigate between screens. Moreover, users can manage their logout operations. Logout operation will redirect users to Auth flow.

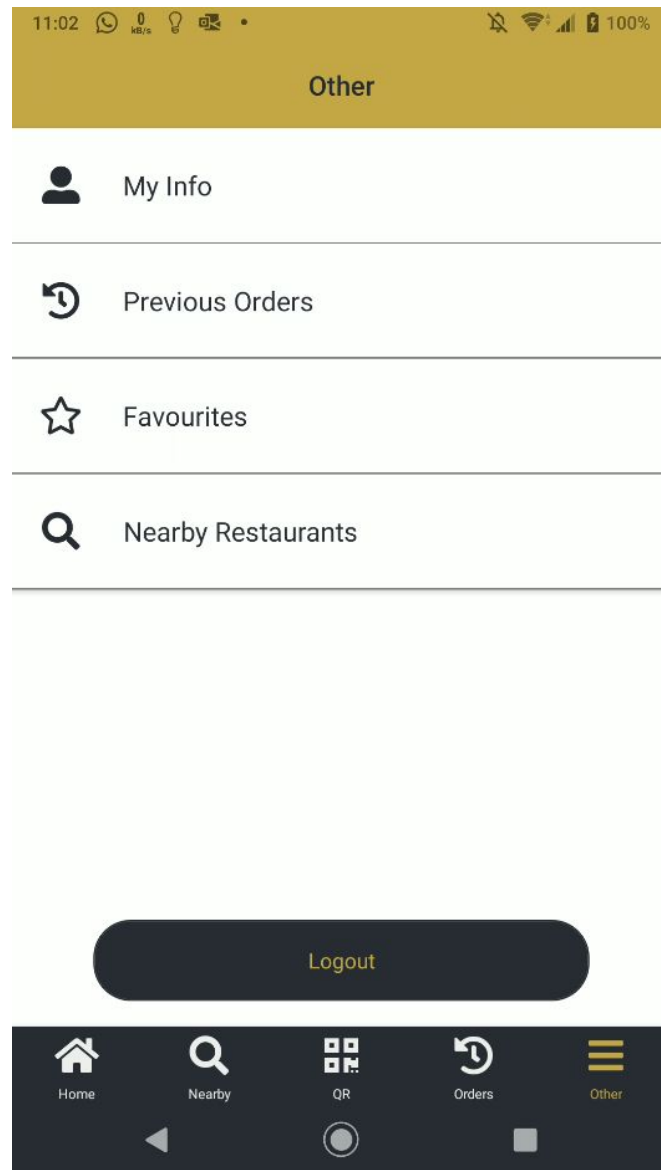


Figure 17: Drawer

My Info Page (*Main Flow*)

My Info Page provides users to see and update their account information easily. Only changed areas will be updated by the system. Therefore, if a user does not want to change password, it is not necessary to re-type the password.

11:03 0 KB/s 100%

← My Info

John

Doe

john_doe@postman.com

postman

.....

Save

Home Nearby QR Orders Other

Figure 18: My Info Page

Previous Orders Page (*Main Flow*)

Previous Orders Page provides users to reach their order history. For each inactive order -paid order- necessary information -rank, restaurant title, relative order date- is given. Users can reach detailed information by tapping the corresponding order's container.

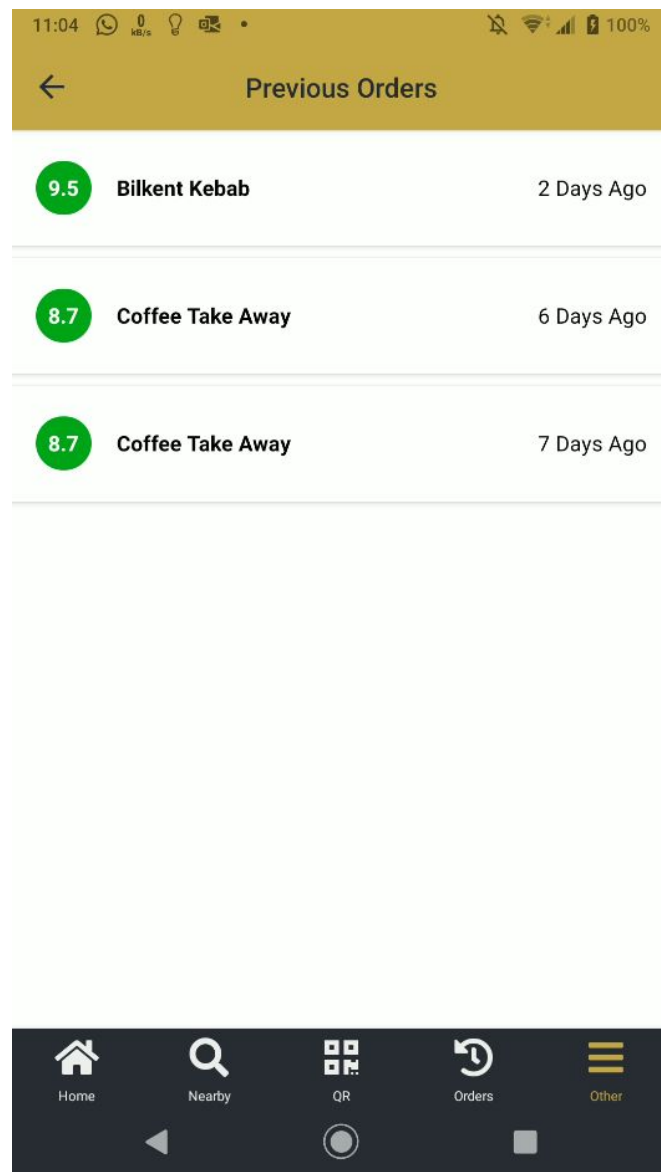


Figure 19: Previous Orders Page

Favourites Page (Main Flow)

Favourites Page provides users to see their favourite restaurants and information -rank, restaurant title- of each restaurant. Users can tap the corresponding restaurant's container in order to see detailed information about it.

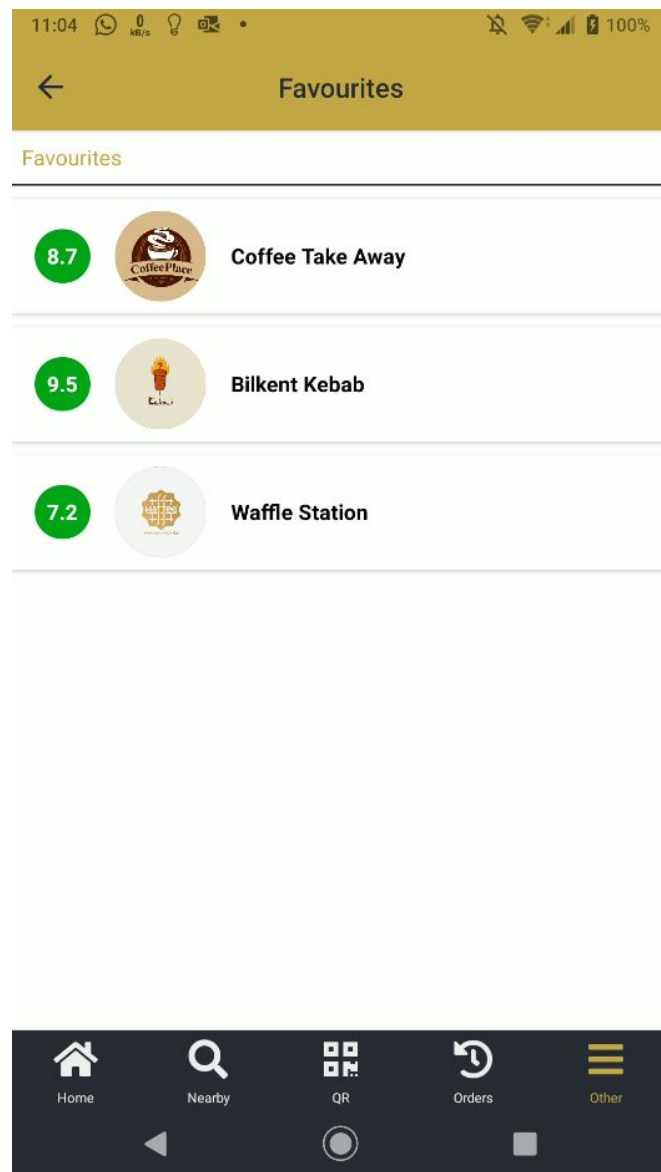


Figure 20: Favourite Restaurants Page

QR Page (*Main Flow*)

QR Page provides users to read QR codes provided by the restaurants physically. QR code includes necessary information for connecting the corresponding restaurant. After the camera read the QR code, the user will be redirected to Order flow automatically.

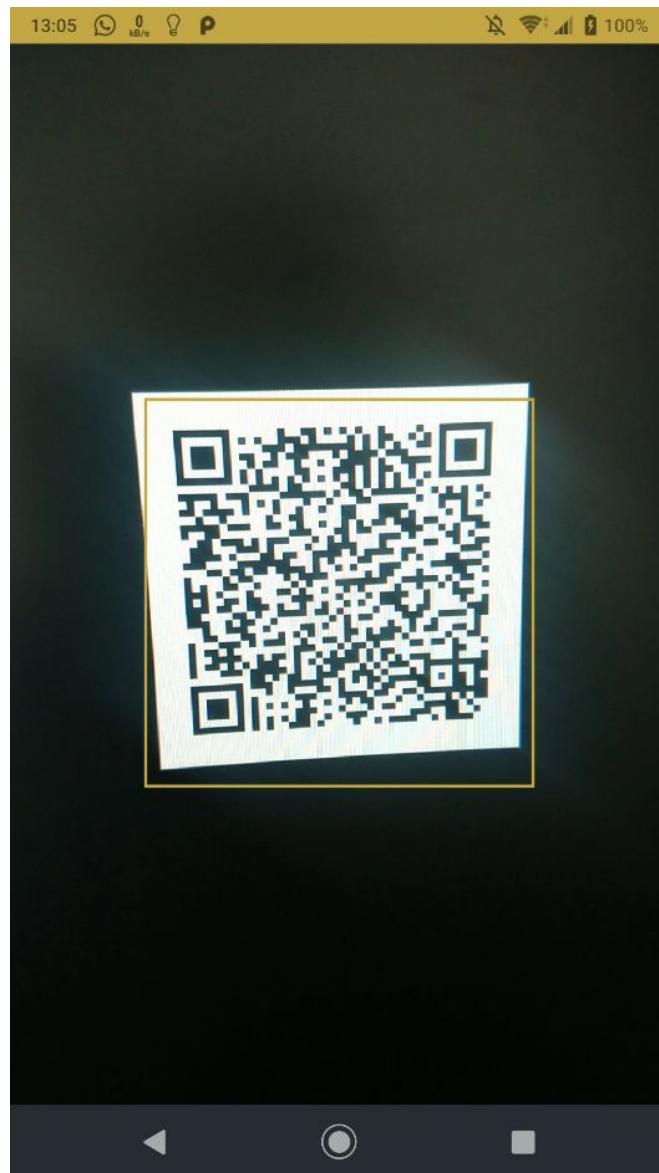


Figure 21: QR Camera

Restaurant Menu Page (Order Flow)

Menu Page provides users to see items in the menu of the corresponding restaurant that is provided by the QR code previous flow. Users can add the items to the basket by the popup menu that is displayed after the corresponding item container is tapped. Tapping out of the popup will close it. Quantity and ingredients can be manipulated by the button and check box. Button in the below will save the modified item to the basket with the settings provided by users choices. Tab bar in the below provides navigation between Menu, Basket and Request Pages.

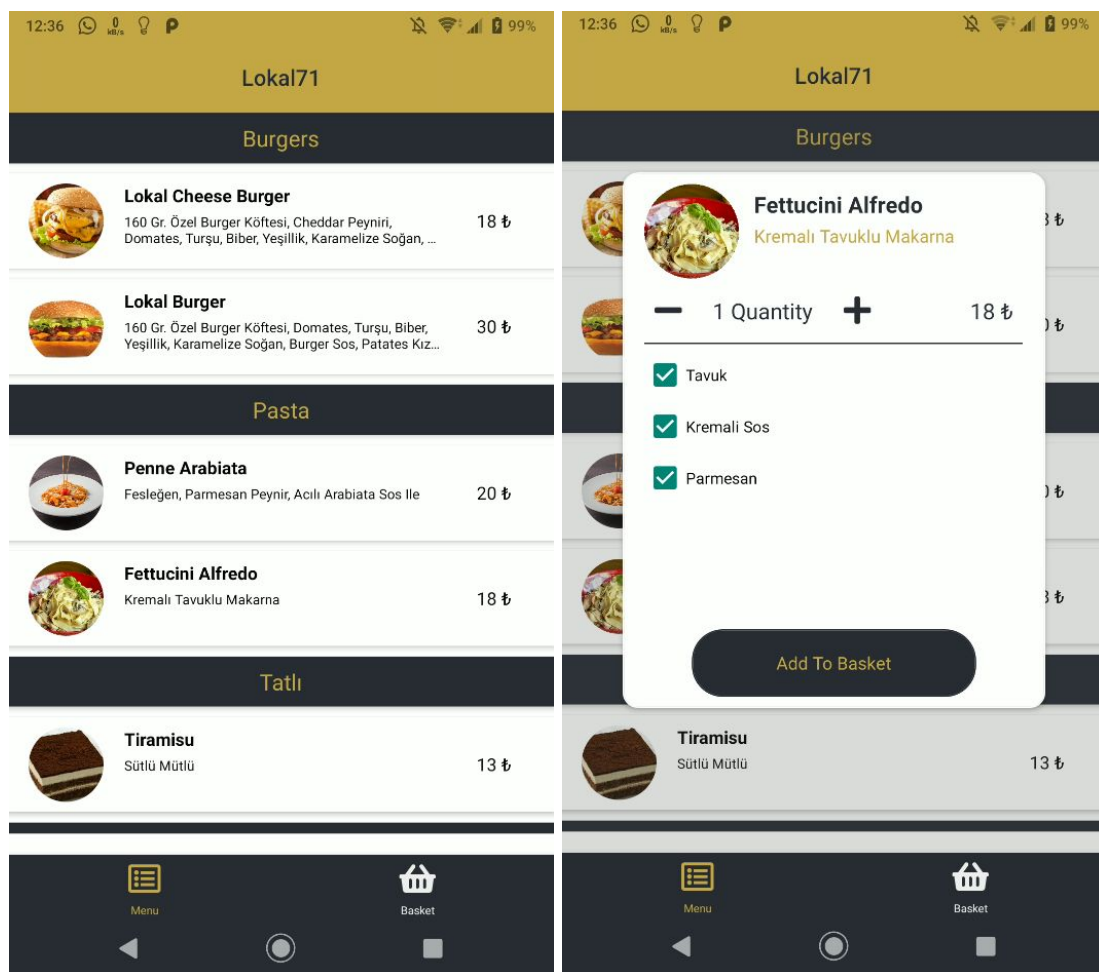


Figure 22: Menu Page

Basket Page (*Order Flow*)

Basket Page provides users to see and manage their current basket and moreover, displays current active order, if there is one, in order to provide full order view. If there is an active order, it will be shown in the collapsible container with the title of “Ordered Items”. Necessary information -item details, total price- will be given in the collapsible part of the container. If there is no active order, the Requests tab will be hidden.

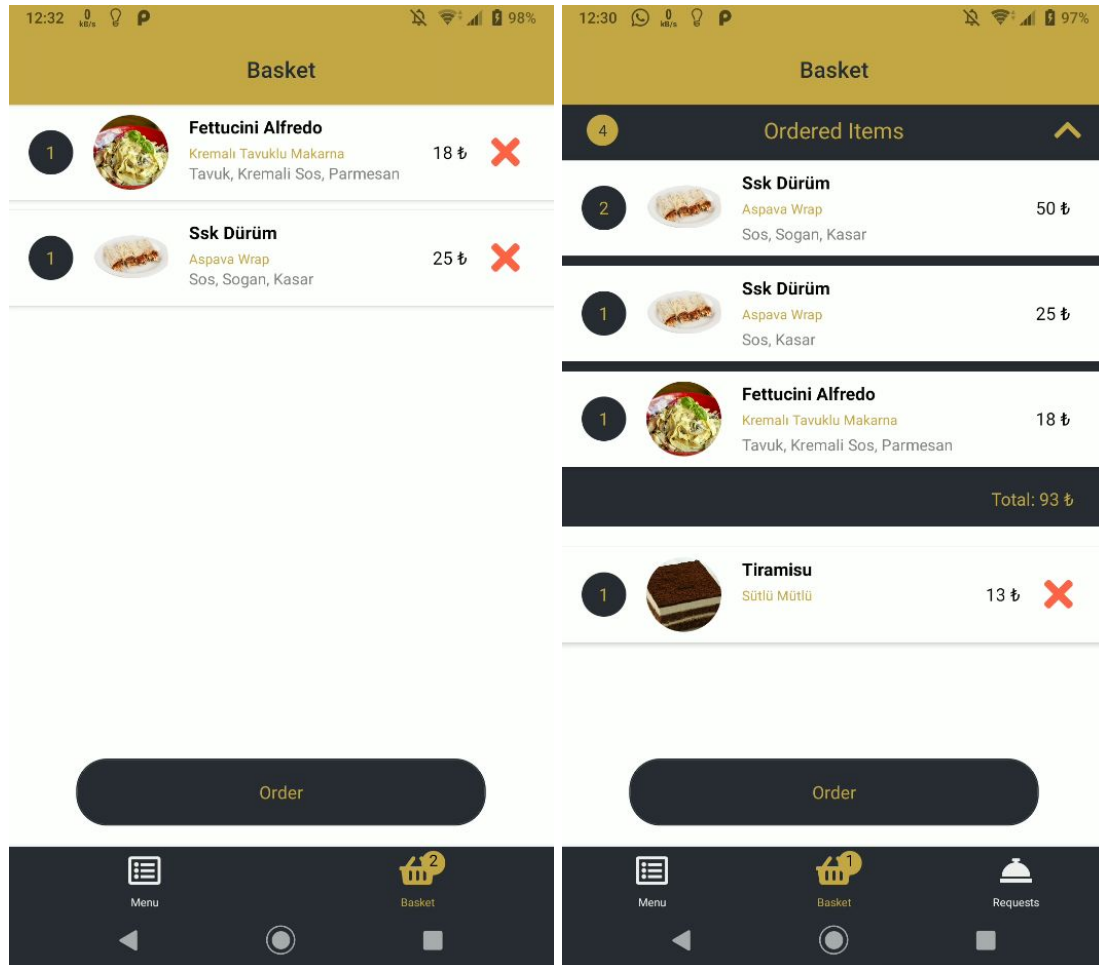


Figure 23: Order Basket

Requests Page (*Order Flow*)

Requests Page provides users to do operations related with the active order. At the top of the page, there is an active order information -item details, total price- container. At the bottom of the page, there is a button group that provides operations related with the active order. Request Payment Button sends a message that equals “check, please!” in the physical world to the corresponding restaurant. Online Payment Button navigates users to the Payment Screen in order to make them complete their orders online. Waiter / Waitress Button sends a message that calls for any available waiter / waitress to the corresponding restaurant.

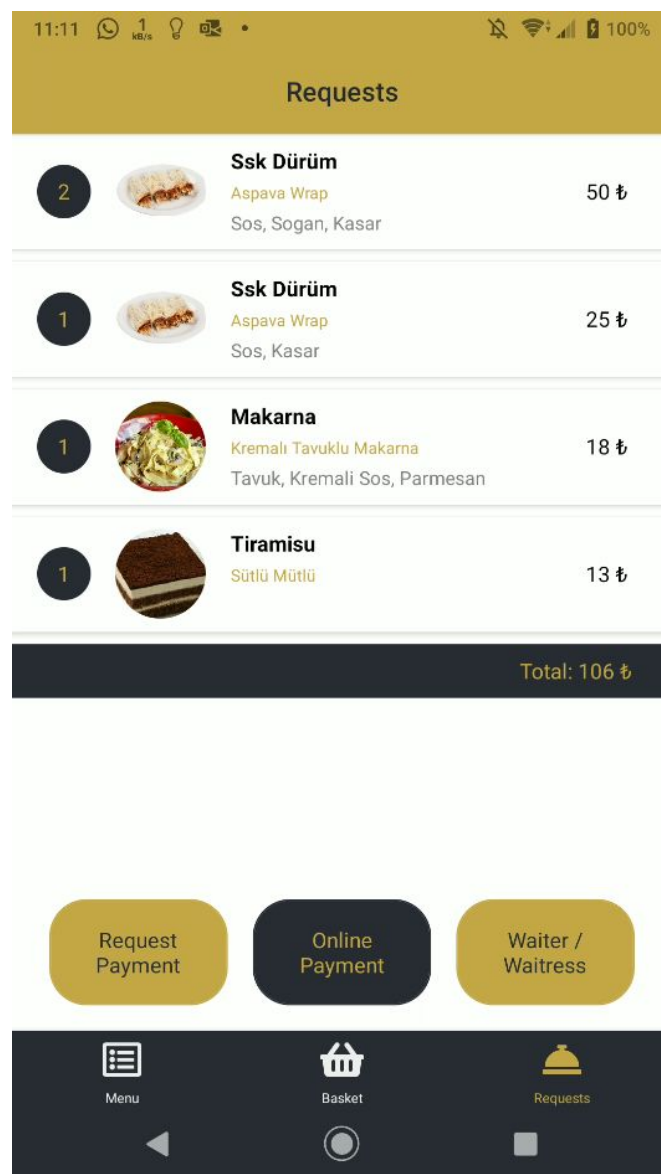


Figure 24: Request Page

Payment Page (*Order Flow*)

Payment Page provides users to pay their final of the active orders with their credit card online. At the top of the page, the total price of the active order is shown. Users must fill the all input fields -Name on Card, Card Number, Expiration Date, CCV- in order to send payment requests. After payment successfully done, users will be navigated to the Main flow.

11:12 0 kB/s 100%

← Payment

Total: 106 ₺

John Doe

1111 1111 1111 1111

12 / 20

123

Pay

Menu Basket Requests

Figure 25: Payment Page

9 Glossary

NodeJS: JavaScript environment, asynchronous event-driven JavaScript runtime [4].

React: JavaScript web framework [5].

React Native: JavaScript mobile framework. Builds are native [6].

Electron: JavaScript cross-platform desktop framework [7].

Heroku: Cloud application platform. Platform as a service company [8].

MQTT: A lightweight messaging protocol for IoT devices oriented [9].

Docker: Container technology [10].

References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] Baeldung. “The DAO Pattern in Java.” Baeldung, 21 Mar. 2020, www.baeldung.com/java-dao-pattern.

[3] Contributors, TechTarget. “What Is REST API (RESTful API)?” *SearchAppArchitecture*, TechTarget, 22 Sept. 2020, www.searchapparchitecture.techtarget.com/definition/RESTful-API.

[4] “About NodeJS”, OpenJS Foundation. <https://nodejs.org/en/about/>.

[5] “React A JavaScript library for building user interfaces”, Facebook Inc.. <https://reactjs.org>

[6] “React Native A framework for building native apps using React”, Facebook Inc.. <https://reactnative.dev>

[7] “Electron Building cross-platform desktop apps with JavaScript, HTML, and CSS”, OpenJS Foundation. <https://www.electronjs.org>

[8] “What is Heroku”, Heroku Inc.. <https://www.heroku.com/what>

[9] “MQTT The Standard for IoT Messaging”, MQTT. <https://mqtt.org>

[10] “What is a Container? App Containerization Docker”, Docker Inc.. <https://www.docker.com/resources/what-container>