

MINISTERUL EDUCAȚIEI



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

---

# MOBILE RESTAURANT ORDERING PLATFORM

License Thesis

Autor: **Barbu Mihai-Adrian**

Conducător științific: **S.L. Dr. Ing. Radu Dan**

**2021**



**UNIVERSITATEA TEHNICĂ**

DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Vizat,

DECAN  
Prof.dr.ing. Liviu MICLEA

DIRECTOR DEPARTAMENT AUTOMATICĂ  
Prof.dr.ing. Honoriu VĂLEAN

Autor: **Barbu Mihai-Adrian**

## **Mobile Restaurant Ordering Platform**

1. **Project proposal:** *Mobile application aimed to create a new way of ordering items at restaurant with help of QR Technology.*
2. **Project content:** *Introduction, Bibliographic Research, Analysis and Design, Implementation & Development, Testing and Validation, Conclusions, Bibliography.*
3. **Place of documentation:** *Technical University of Cluj-Napoca*
4. **Consultants:** -
5. **Date of issue of the proposal:** November 1, 2020
6. **Delivery date:** July 7, 2021

Author            Barbu Mihai-Adrian

Supervisor      S.L. Dr. Ing. Radu Dan

**UNIVERSITATEA TEHNICĂ**

DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

**Declarație pe proprie răspundere privind  
autenticitatea proiectului de diplomă**

Subsemnatul(a) **Barbu Mihai-Adrian**, legitimat(ă) cu CI seria SB nr. 733727, CNP 1971106324804,

autorul lucrării: Mobile Restaurant Ordering Platform.

elaborată în vederea susținerii examenului de finalizare a studiilor de licență la **Facultatea de Automatică și Calculatoare**, specializarea **Automatică și Informatică Aplicată (în limba engleză)**, din cadrul Universității Tehnice din Cluj-Napoca, sesiunea Iulie 2021 a anului universitar 2020-2021, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

07.07.2021

Prenume NUME

Barbu Mihai-Adrian

(semnătura)



## SINTEZA

proiectului de diplomă cu titlul:

### **Mobile Restaurant Ordering Platform**

Autor: **Barbu Mihai-Adrian**

Conducător științific: **S.L. Dr. Ing. Radu Dan**

1. Cerințele temei: Making a platform that will replace the physical menu from restaurants and will introduce a new way of ordering with the help of smartphones and QR technology.
2. Soluții alese: For the development I used a Django to develop the back-end of the application and Flutter to develop the mobile application.
3. Rezultate obținute: Developed a back-end service along with a mobile application that also benefits from a presentation site developed with HTML, CSS, JavaScript.
4. Testări și verificări: The application benefitted of a suite of manual tests before every deployment, therefore it works as expected.
5. Contribuții personale: Developed a platform that offers dynamically created user interfaces along with database support that fetches data from a service that is hosted on a personal server for customers to benefit from a much faster and safer ordering experience.
6. Surse de documentare: Personal knowledge, Internet.

Semnătura autorului

Semnătura conducătorului științific

# Cuprins

<b>1</b>	<b>INTRODUCTION.....</b>	<b>3</b>
1.1	GENERAL CONTEXT .....	3
1.2	OBJECTIVES .....	4
<b>2</b>	<b>BIBLIOGRAPHIC RESEARCH .....</b>	<b>5</b>
2.1	TRENDING IN MOBILE INDUSTRY.....	5
2.1.1	<i>Android development</i> .....	5
2.1.2	<i>iOS development</i> .....	5
2.1.3	<i>Cross Platform Development</i> .....	5
2.2	FLUTTER FRAMEWORK.....	6
2.2.1	<i>Flutter widgets</i> .....	6
2.2.1.1	Stateless Widgets .....	6
2.2.1.2	Stateful Widgets .....	6
2.2.1.3	Dart Programming language .....	6
2.3	TRENDING IN BACKEND FRAMEWORKS .....	7
2.4	DJANGO FRAMEWORK.....	8
2.4.1	<i>Python Programming Language</i> .....	8
2.4.2	<i>Python Virtual Environment</i> .....	9
2.4.3	<i>Django Database Support &amp; Models</i> .....	9
2.4.4	<i>Django Views and Routing</i> .....	10
2.4.5	<i>Administrator interface</i> .....	10
2.4.6	<i>Cross Site Request Forgery protection (CSRF)</i> .....	11
2.4.7	<i>Server Gateway Interface</i> .....	11
2.4.7.1	WSGI.....	11
2.4.7.2	ASGI.....	12
2.4.8	<i>Django Channels</i> .....	12
2.5	SERVER-CLIENT ARCHITECTURE .....	12
2.6	HTTP REQUESTS & RESPONSES .....	12
2.6.1	<i>Response Status code</i> .....	13
2.6.2	<i>Response Headers</i> .....	13
2.6.3	<i>Response Body</i> .....	13
2.7	WEB SOCKETS .....	14
2.8	JSON (JAVASCRIPT OBJECT NOTATION) .....	15
2.9	WEB DEVELOPMENT TECHNOLOGIES.....	16
2.9.1	<i>HTML</i> .....	16
2.9.2	<i>CSS (Cascading Style Sheets)</i> .....	16
2.9.3	<i>JavaScript</i> .....	16
2.9.4	<i>AJAX</i> .....	17
2.9.5	<i>jQuery</i> .....	17
2.10	WEB SERVERS.....	17
2.10.1	<i>Apache HTTP Server</i> .....	17
2.10.2	<i>NGINX</i> .....	17
2.11	LINUX.....	18
2.11.1	<i>Security-focused</i> .....	18
2.11.2	<i>User-Focused</i> .....	18
2.12	RASPBERRY PI.....	19
2.12.1	<i>CPU Architectures</i> .....	19

2.12.1.1	CISC vs RISC .....	19
2.12.1.2	x86.....	19
2.12.1.3	ARM.....	19
2.12.1.4	Raspberry Pi 4 specifications.....	19
2.12.2	<i>Performance</i> .....	20
2.12.3	<i>Power Consumption</i> .....	20
2.12.4	<i>Raspbian</i> .....	21
2.13	FIREWALL.....	21
2.14	NETWORK ADDRESS TRANSLATION (NAT) .....	21
2.14.1	IPv4 vs IPv6.....	22
2.14.2	Public vs Private IP address.....	22
2.14.3	Port Forwarding .....	22
2.15	DNS (DOMAIN NAME SYSTEMS).....	23
2.15.1	Static vs Dynamic IP .....	23
2.15.2	DDNS (Dynamic DNS).....	23
2.16	SSL CERTIFICATE .....	24
2.17	QR TECHNOLOGY .....	24
<b>3</b>	<b>ANALYSIS AND DESIGN .....</b>	<b>26</b>
3.1	BACK END.....	26
3.2	FRONT END.....	27
3.3	DATABASE.....	28
3.4	MOBILE DEVELOPMENT .....	28
3.5	PROGRAMS USED .....	29
3.6	USE CASE DIAGRAM & SEQUENCE DIAGRAM.....	30
<b>4</b>	<b>IMPLEMENTATION &amp; DEVELOPMENT.....</b>	<b>31</b>
4.1	FRONT END DEVELOPMENT .....	31
4.2	BACK-END DEVELOPMENT .....	32
4.3	MOBILE DEVELOPMENT .....	36
4.4	DATABASE IMPLEMENTATION .....	41
<b>5</b>	<b>TESTING AND VALIDATION.....</b>	<b>43</b>
<b>6</b>	<b>CONCLUSIONS.....</b>	<b>45</b>
6.1	CONTRIBUTION SUMMARY .....	45
6.2	OBTAINED RESULTS .....	45
6.3	OPTIMIZATIONS .....	45
<b>7</b>	<b>BIBLIOGRAPHY .....</b>	<b>48</b>

# 1 Introduction

## 1.1 General Context

We must face it; smartphones have become a part of our daily routine and without them we feel like there is a hollow part in us. That is because they literally became an extension of ourselves. With the use of those powerful devices, we manage our emails, make appointments, set reminders, and stay in contact with our family and friends. And finally, we can search for information in every moment of every day, in every corner of the internet.

We can already see that over the years, more and more tools supported by smartphones have reached us, the consumers, replacing along the way, conventional tools. Examples of obsolete utilities are agendas, video cameras, GPS, bank accounts, watches, calculators, compasses, alarm clocks, mail, lanterns, photo albums, MP3 players, magazines and the list might continue for a good amount of time still.

This “21<sup>st</sup> century adaptation” has brought us a lot of flexibility and adaptability. Even though we are facing the risk of forgetting our smartphone at home for instance, making us virtually disconnected from the outside world, it will still make it an overall smaller risk, than constantly forgetting one of the tools our smartphone provides.

Being dependent on so many tools and utilities, a person is virtually bonded to the smartphone. This will ensure that a smartphone user will have his/her device in his possession at any given time. This ordering platform, Qord, comes to make use of the smartphone’s UI adaptability and combine it with traditional restaurant menus, that we all used for years now, to create a user experience that is both more efficient yet more relaxing than traditional restaurant ordering.

This platform, which comes by the name “Qord”, has the purpose, not only to replace the standard restaurant ordering process, but also to provide a powerful interface for restaurant personnel. This tool is also meant to deliver an easy-to-use tool for the restaurant administrator to dynamically update menu’s contents, make personnel administration and monitor sales and other data with the purpose of optimizing their enterprise and perhaps discover in time the weak links of the business.

For the development of this application, there was a Server-Client architecture used. The Server Side or Backend was developed in Django, a framework based on Python, that manages routing, databases, HTTP requests, security and so on.

All the backend related applications were hosted on a Raspberry Pi, including the database and presentation site. A DDNS was provided free by the personal ISP, making real world deployment, testing, performance analysis and logging, possible.

The Framework chosen to build the Mobile applications, was Flutter. This framework makes it easy and accessible to make cross platform deployment. Flutter, just like React Native, makes code reusable and saves a lot of time by making the code usable for both Android and iOS.

The communication between Server and Client is obtained via HTTP requests and via Web Sockets for real-time data transfer between Restaurant Personnel application and the server application that serves the Orders handling.

## 1.2 Objectives

In the last couple years, in restaurants, there has been a tendency to replace the conventional menus with online menus that you can visualize on your phone. The client will scan a QR code from a label that is provided on the table, containing a URL that takes the user to a web page, which displays the menu. The idea behind this scanning method is that it offers flexibility for the restaurant administrator, making it easy to make changes to the menu, like delete, add, or change foods or drinks, or make items unavailable.

This Platform comes to take this idea one step further, by not just providing a static method of displaying the menu in the form of a HTML or PDF, but to deliver a system that makes ordering routine both interactive and more efficient.

In this project, the bricks for the foundation of this app will be placed, making it an expandable system, on top of which many features and new functions will be easy to develop and deploy. This way of organizing the features of the app is the way to go to make it appeal to potential customers by initially replacing the ordering mechanism and further on, finding a solution to gradually replace all the traditional routines necessary for managing a restaurant: billing, personnel payments, inventory management and so on.

The main objectives of this thesis are:

- study what are the needs of the restaurant owners and what are the areas that the traditional restaurant ordering can be improved to make a better user experience for the restaurant's customers.
- create a website for the presentation of the platform along with all the necessary styling to make an appealing look of the whole project.
- create all the database models required for the administration and data handling for a wide variety of restaurants.
- create a reliable, performant and secure Back End application that handles all the database storing, presentation website hosting along with a restaurant subscription application.
- create a mobile application that offers an intuitive user interface that is created dynamically based on the menu items each restaurant offers.
- integrate a QR scanning system into the mobile application that executes server requests and retrieves data.
- establish a reliable connection simultaneously between multiple clients and the server via HTTP requests and Web Sockets.
- Handle Orders with an app designed for the restaurant's personnel that will keep the order status updated and offers real time data transfer between client and server.



## 2 Bibliographic Research

### 2.1 Trending in Mobile Industry

From 2007, since the release date of the first iPhone, and 2008, the first release of the Android OS, the market has been divided between Android users, and iPhone users. Summed up, they hold 98% of the total global market for mobile operating systems. [1].

Because virtually the whole mobile device industry is hosted on iOS and Android, for a product like Qord to survive on the competitive market, it is necessary to be supported by both platforms.

#### 2.1.1 Android development

Without doubt, the Android Operating System is the leader on mobile device market and worldwide is the most used OS. It is forecasted that by 2022, from the total mobile device market, 87% will be hosted on Android. [1]

For the development of apps for Android, the most used programming languages are Java, and Kotlin (which is a cleaner and more compact than Java). Kotlin remains one of the most widespread technologies used to develop Android applications which “enhances productivity and enhances developer happiness”. [2]

#### 2.1.2 iOS development

Falling right behind Android in terms of market share, with a total percent of 22.1% in 2019, is Apple’s mobile operating system, iOS. [1]

Most common options available for developing software for iOS are Objective-C and most common choice nowadays for iOS development is Swift.

#### 2.1.3 Cross Platform Development

In the last years, there was a tendency for mobile development to slowly make a transition from native development that focuses on the target platform, to cross platform apps that focuses more on re-usable code. This trend comes to keep the development cost low and make portability an easy task.

Examples of cross platform frameworks are React Native (a platform released by Facebook in 2015) and Flutter (platform developed by Google and released in 2018).

Despite being newer on the market than React Native, Flutter has shown signs of rapid gain in popularity, and it will perhaps, overcome React Native, as the user community rises.

## 2.2 Flutter Framework



Flutter is a cross-platform mobile development framework developed by Google that was first released in 2018.

Being a cross-platform framework just like React Native, Flutter is supported on both Android and iOS, making deployment an easier task and helps by making code more reusable and pleasant to develop and work with.

### 2.2.1 Flutter widgets

“Flutter widgets are built using a modern framework that takes inspiration from React.” [3]

Flutter widgets are basic components that can be used to build the user interface of your application or can be used to build more complex widgets, in order to make code more reusable. There are two types of widgets in Flutter framework: Stateless and Stateful widgets.

#### 2.2.1.1 Stateless Widgets

Stateless Widgets are widgets that, as the name suggests, have no state. A widget is stateless when, over time, no changes apply to the view constructed by the widget tree. Otherwise, Stateless widgets can be considered, static. Examples of these widgets are Text, Icon, IconButton, Center.

#### 2.2.1.2 Stateful Widgets

Unlike Stateless Widgets which are considered static, stateful widgets can change the appearance over time. The state change can be triggered by a user interaction or data transmission from a Web Socket, for example a counter that is incremented on a press of a button, in this case, at every press of the button the widget is rendered again to keep data updated. This data change is made by calling the `setState()` function, that will update the user interface.

Examples of these interactive widgets are Checkbox, Radio, Slider, Form, Textfield.

#### 2.2.1.3 Dart Programming language

Dart is a programming language developed by Google and first came to the market in 2011. It is designed for development of apps and web-apps destined for client-side use. The syntax is C++ based. Dart goal is to deliver a productive programming language that supports multi-platform development and flexible execution all along.



Dart supports native platform as well as web platforms. For web applications, Dart offers compilers that translate Dart to JavaScript for Web use. [4]

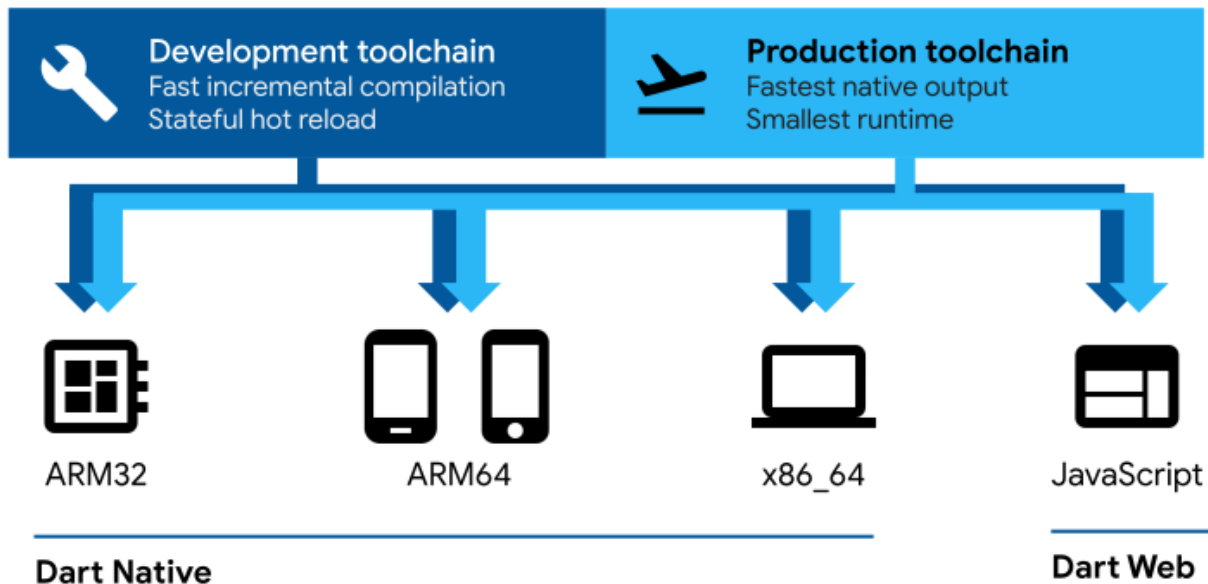


Figure 2.1- Dart Native & Dart Web [4]

## 2.3 Trending in Backend Frameworks

Backend represents the part of the web application that is related to the Server side. A Backend framework represent a multitude of libraries that are offered to develop a web application or mobile application.

A framework, typically, offers necessary functionality that would be required to develop typical Backend tasks like handling HTTP requests, routing, sorting out security standards, database interfacing and so on.

This functionality is standard for every framework available on the market, so it makes sense to stick to a framework that has already been in development for years and do not try to “reinvent the wheel” [5] as the Django Backend Framework documentation suggests.

In terms of popularity in 2021, Laravel is the most used Backend Framework for web development, even though is written in PHP and it is claimed that PHP is steadily losing its popularity. Then, Django and Flask. Two frameworks that use Python for development, making it very easy to comprehend because of its simple syntax.

There are a lot of choices available on the market for Backend development on top of which you can build the structure of your application. Here are some examples of available frameworks and the programming languages used with them:

- Django (Python)
- Flask (Python)
- Laravel (PHP)
- Ruby on Rails (Ruby)
- Spring (Kotlin)
- Express (JavaScript)

## 2.4 Django Framework

Django is a Web Framework on Python programming language, created to ease up database driven web sites. [6] For the development of the applications, routing, database models and settings, Python is the programming language used. It is a fast, reliable, scalable, and versatile, encouraging the “rapid and clean” development of Web applications without the need to “reinvent the wheel”. [5]



Since 2005, the year of the first release, Django has gradually gained popularity in the area of web development and now, there are some major companies that are using this python-based framework, like Instagram, Pinterest, Mozilla, National Geographic Disqus and so on.

### 2.4.1 Python Programming Language

Python is a high-level programming language that has been on the market for well over 30 years now. Python does not require any compilation, making it an interpreted language, adding to its flexibility, just like JavaScript, PHP, or Ruby. Python supports both functional programming, as well as object-oriented programming.

The idea behind Python is to write code that is both readable and logical for small projects up to large-scale ones used in production environments.

Python syntax is easy to comprehend, clean and makes use of indents instead of brackets to mark the start and the finish of a function or statement. Also, in Python the semicolon use at every end of line in the code is absent.

In terms of popularity, Python is among the top 3 used programming languages, along with Java and JavaScript and it is regarded as “the easiest programming language to learn, because of its simple syntax” [7].

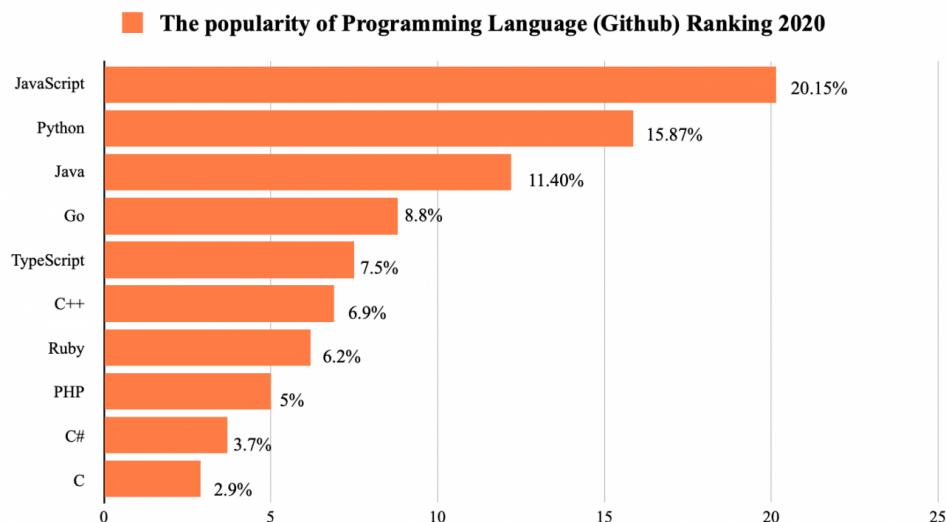
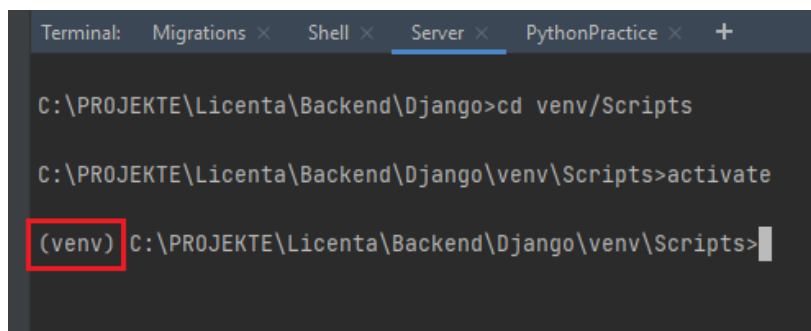


Figure 2.2 – The popularity of Programming language [7]

### 2.4.2 Python Virtual Environment

When working on different projects simultaneously, there might be a case when one Python instance cannot meet the requirements of every project. This is because throughout the projects, different versions of packages may be used. In order to avoid conflicts, when a project cannot run because of package incompatibility, Virtual Environments are the solutions to this problem.

Virtual environments can support different versions of Python and different instances of the containing packages and libraries. When creating a virtual environment, a copy of the Python interpreter will be created with all the base libraries it contains and various additional files. After creating the environment “*activate*” command must be executed to start using the interpreter.



```
Terminal: Migrations x Shell x Server x PythonPractice x +
C:\PROJEKTE\Licenta\Backend\Django>cd venv/Scripts
C:\PROJEKTE\Licenta\Backend\Django\venv\Scripts>activate
(venv) C:\PROJEKTE\Licenta\Backend\Django\venv\Scripts>
```

Figure 2.3 – Virtual Environment Activation

### 2.4.3 Django Database Support & Models

Django supported databases are PostgreSQL, MariaDB, MySQL, SQLite, and several third-party providers.

As standard, Django comes preinstalled with SQLite database management system, with a pre-installed environment to start the database development.

The way table fields are created in Django, are through Models. Models are created by defining a class which inherits `django.db.models.Model`. Each attribute of the class will represent a field in the table.

Whenever a change in the models of the database happens, there are two steps that need to be done in order to perform all the changes in the database structure:

- `makemigrations` – is the command that generates SQL commands for preinstalled applications.
- `migrate` – the model created with the `makemigrations` command are then executed and added to the database.

One attribute of the Django model is the `__str__` method which represents the identifier under which it is found in the admin interface under *localhost:8000/admin*.

```

20
21 class School(models.Model):
22     name = models.CharField(max_length=200)
23     years = models.IntegerField(default=4)
24
25     def __str__(self):
26         return self.name
27

```

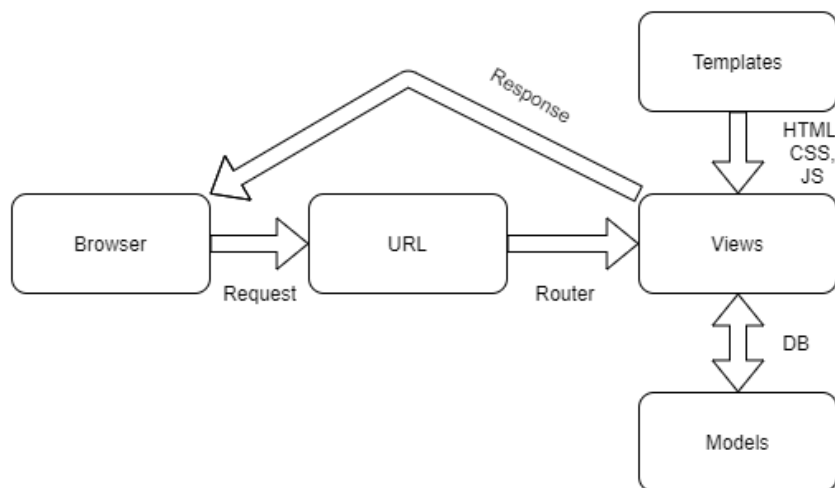
**Figure 2.4 – Representation of Model and its fields**

#### 2.4.4 Django Views and Routing

Django tree application is built based on a dispatcher that routes the traffic to the service it delivers. The dispatcher rules are stored in a `urls.py` file which directs the requests to the requested service that is typically located in the `views.py` file that can be found in the app's specific folder.

When working with Web Sockets in the application or Django Channels, the structure changes from `urls.py` and `views.py` to `routing.py` and `consumers.py`.

The way the URL dispatcher works is by linking a URL name from the `urls.py` file to a method that can be found in the `views.py` file. In order to preserve a sense of structure throughout the Django application, both files will be located in the same folder which is regarded as a module of the application.



**Figure 2.5 – Data Exchange Within Django**

#### 2.4.5 Administrator interface

Django provides an administrator page that can be accessed at `"/admin"` route while the server is active. It is a powerful interface that gives the developer the ability to

validate the views that interacts and manipulates the database. Django documentation informs that the recommended use of the administration page to be limited to an internal management tool and not to be used as a front-end interface in production.

Each Django module will have a file, internally, named *admin.py* that will contain the models that will be displayed in the administrator interface. In this way, only key Models can be chosen to be viewed within the page so that a clean and compact view will be preserved.

Django offers the ability to create custom functions that can be executed within the administrator page, offering this tool a customizable approach, increasing its usability properties.

#### 2.4.6 Cross Site Request Forgery protection (CSRF)

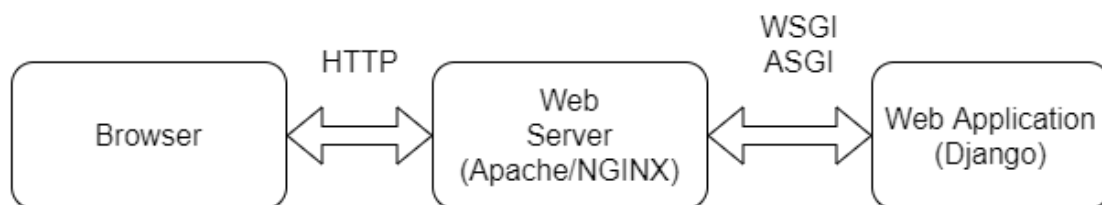
CSRF represents a web vulnerability which user is persuaded to perform unintended actions that may lead to data steal (passwords, accounts, bank accounts).

These vulnerabilities can be avoided with the use of CSRF Tokens, which Django has support for. CSRF Tokens are sent from the Django server via a cookie when a request is made by the client. Afterwards, when a request happens, the data will pass through a validation procedure based on the CSRF Token and rejects the requests if the token is invalid or missing.

As mentioned earlier, Django offers protection against CSRF attacks, by providing CSRF Token support. For example, when building a Form in Django, by including the line: `{*csrf_token*}` at the beginning of a form in Django, a CSRF Token will be sent along and when the form is submitted, a validation will take place on the Server-Side.

#### 2.4.7 Server Gateway Interface

Server Gateway Interface is a calling convention that supplies a description on how a Web Server like Apache or NGINX should communicate with the Web Application typically written in Python.



**Figure 2.6 – Web Server & Web Application**

##### 2.4.7.1 WSGI

Web Server Gateway Interface is a convention that allows server-client data to flow synchronously. When the client makes a request, the server will handle the response. When, for example, we want to implement a system that checks if the server has updates to send, we will need to repeatedly perform a request to check if any

updates took place. This system will be not suitable for a real-time application when live updates are a must.

#### 2.4.7.2 ASGI

Unlike its predecessor, WSGI, ASGI (Asynchronous Server Gateway Interface) offers asynchronous communication for async-capable Django web applications while maintaining the capabilities of synchronous data transfer of WSGI, offering a backwards-compatibility implementation. Asynchronous communication is necessary for the implementation of Web Sockets communication.

#### 2.4.8 Django Channels

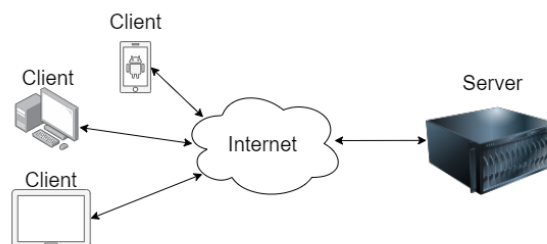
- Channels is the library that implements Web Sockets technology into the Django Framework offering beyond HTTP requests capabilities.
- Gives the option to choose between synchronous and asynchronous connections.
- Because it is implementing an asynchronous communication method, it is built upon ASGI.

### 2.5 Server-Client Architecture

In a Server-Client architecture, the client (a remote computer or any type of device that has processing power) will make requests and receive services that are hosted by a server (centralized computer or powerful device, that is able to offer some services).

Unlike peer-to-peer architecture, where data is equally distributed throughout the system and all the entities in the system are equal in terms of responsibility, in a client-server architecture, in the center of system stands a server that handles the data, every time a client makes a request (and has permission to execute it).

The client-server architecture is most useful for applications that require a layered approach, where data abstraction between the server-side and the client-side is needed. Separation of functionality improves security and offers the service provider a better control on the way the application runs.



**Figure 2.7 – Client-Server Architecture**

### 2.6 HTTP Requests & Responses

HTTP (Hypertext Transfer Protocol) is a protocol designed to establish a communication between clients and servers. This protocol works based on a request-



response protocol, where clients make a request to either obtain or send data to the server, while servers listen and process requests.

Requests are placed, based on the actions the clients need to perform. There are 7 HTTP methods: GET, POST, PUT, HEAD, DELETE, PATCH, OPTIONS.

The most used HTTP methods are: GET (used to retrieve data from the server), POST (used to send data to the server).

Naturally, because this is a request-response protocol, after a request is sent to the server, a response is waited in exchange. A response represents the server's result after the client's request has been processed.

A response is made from three main components: *response status code*, *response body* and *response headers*.

### 2.6.1 Response Status code

The response status code is a 3-digit number that represents the outcome of the HTTP request. The first digit of the code represents the class of the response:

- 1xx – Informational Code (request received and preceding)
- 2xx – Success Code (received successful)
- 3xx – Redirection Code (further action must be taken to complete request)
- 4xx – Client Error (request could not be completed ex. 404)
- 5xx – Server Error (server could not complete a request)

However, the last two digits of the code do not have a categorical value and are specific for the code class.

### 2.6.2 Response Headers

Headers represent additional information associated with the request. A request may contain 0 or more headers.

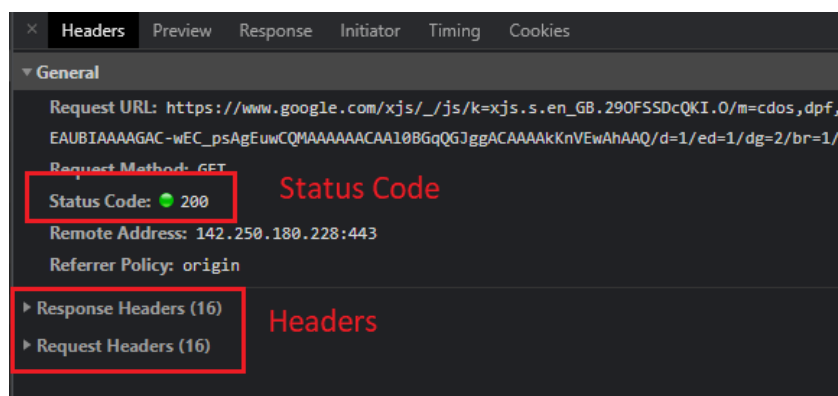
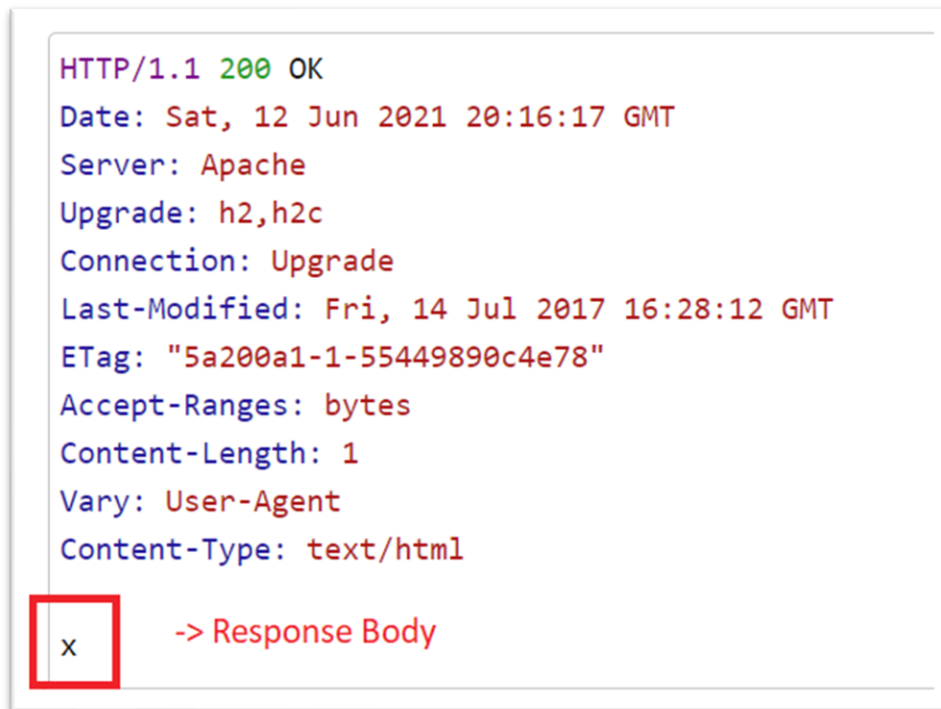


Figure 2.8 – Response status code and headers

### 2.6.3 Response Body

The Response Body represents the message associated with the response, after the header's messages end. In order to visualize a response body along with headers and status code you can use a site called [www.reqbin.com](http://www.reqbin.com) in order to place requests. This is

the response of the site [www.x.com](http://www.x.com) which returns only the letter “x”, along with the headers:



**Figure 2.9 – HTTP Response body**

Typically, a HTTP response body will be a HTML, JSON or plain text.

## 2.7 Web Sockets

Web Sockets is a communication protocol that makes real-time bidirectional communication, between two end points, possible. Unlike HTTP protocol, that is based on request response methodology, web sockets maintain permanent communication and the data transmission is full duplex (data can be sent in both directions).

Usually, when accessing a website, the *http://* or *https://* protocol will pop up at the beginning of the search bar indicating that in order to access a website or any web address for that matter, the (*secure*) *hypertext transfer protocol* will be used in order to accomplish the task. However, in order to access the Web socket protocol *ws://* or *wss://* protocol will be used.

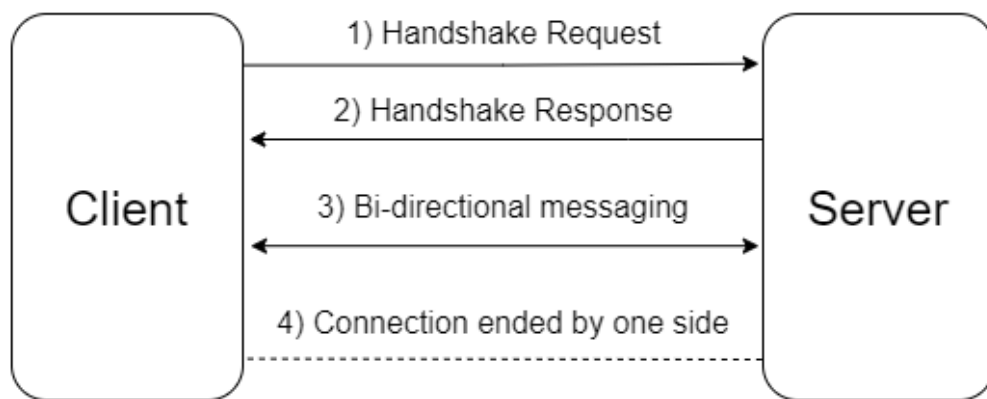
This advanced technology is very useful when creating real-time applications, because messages can be sent to the server and obtain responses without needing to poll the server for a response.

There are four functions that need to be implemented by both endpoints in order to be able to accomplish a web socket connection:

- Handshake – The connection between the devices (“onopen” function)

- Data Transfer – Both devices must be able to receive and send data to each other (send function – will send data; onmessage – will receive incoming data from other web socket users)
- Handshake close – The connection end between the devices

The WebSocket communication protocol is actively used by several real time data applications. For example, a real time chat application, intensely used by millions of people every day, is WhatsApp or Facebook Messenger. Web Sockets are used virtually by every application that need instant data transfer (Social Feeds, Financial tickers, multiplayer games, location-based apps, online education, video streaming and so on).



**Figure 2.10 – Web Sockets Life Cycle**

## **2.8 JSON (JavaScript Object Notation)**

JavaScript Object Notation represent an easy to read and write data interchange format that consists of a collection of names or value pairs.

The first value of the pair is usually represented as a string, and this will be the key (or identifier) of the object. The second element is data stored under that identifier. This data can be of any format: bool, int, double, array, sequence or other objects.

This data format nowadays is universal, and any programming language supports it under a form or another and is called differently:

- Python - Dictionary
- JavaScript - Object
- Dart – Map
- Java – HashTable
- C# - Dictionary/HashTable
- C/C++ - \*\*\*Own Implementation\*\*\*

## **2.9 Web development technologies**

### **2.9.1 HTML**

Hypertext Markup language represents the World's Wide Web core markup language on top of which every website has been created since 1991. Nowadays, the world wide web consists of millions of web pages and more and more pages are coming along every day.

At the foundation of HTML stand tags and attributes. These components act like containers that are meant to build entire web pages. The HTML tags are keywords that are wrapped in angle brackets that describe how the content will be displayed on the screen. The end of the tag will be also displayed in angle brackets but with a "/" (slash) that indicate the end of the tag. Examples of tags:

- `<html>` - The tag that will wrap the whole HTML document.
- `<title>` - This tag will modify the document's title that is shown in the browser's tab bar.
- `<h1>...<h6>` - These tags are called headings and are categorized by their importance. `<h1>` represent the most important tag, therefore it will be displayed in the biggest font and bold font-weight (32 pixels) and up to `<h6>` which is the least significant one (10.72 pixels).
- `<p>` - Will return a new paragraph in the HTML document.

According to [www.w3schools.com](http://www.w3schools.com), there are a total of 119 HTML tags. A list of all the available tags can be found at this link: [www.w3schools.com/TAGS/default.ASP](http://www.w3schools.com/TAGS/default.ASP).

### **2.9.2 CSS (Cascading Style Sheets)**

Cascading Style Sheets is a style sheet language which has the purpose of describing a document that is written in a markup language (in our case HTML).

The purpose of HTML is, as the name suggests, to add style to a web page for better illustration of data or just to offer to the users, a better, more modern user experience. CSS can manipulate fonts, colors, and layout of certain elements of a web page, making web pages more illustrative and pleasant to watch.

Other standardized Style Sheets languages among CSS are: DSSSL (Document Style Semantics and Specification Language) and XSL (Extensible Stylesheet language), but CSS remains the most used standard since it was introduced in 1996.

### **2.9.3 JavaScript**

JavaScript is referred as the scripting language used in the development of interactive web pages, but it can also be used in other environments, outside web browsers. JavaScript is an interpreted language just like Python or Ruby and with the help of a runtime environment like Node.js, JavaScript code can be executed in an environment that is not related to a web browser like Chrome's V8 JavaScript Engine.

The most uses JavaScript has, is in web-based applications development. But the area it is used in does not stop in the web development area. JavaScript can also be used

in embedded hardware control, game development, image processing and the uses are far to be over. In terms of syntax, JS's syntax is related to Java.

#### 2.9.4 AJAX

Ajax (Asynchronous JavaScript and XML) represent a series of techniques meant to update a web page content without needing to execute a page refresh. This ability is crucial for having live updates from a server to have real-time applications like chat rooms or live data visualizer.

#### 2.9.5 jQuery

jQuery is a JavaScript library that has the purpose to standardize JavaScript differences across browsers. jQuery uses AJAX to fulfill its various functions.

jQuery has the purpose of simplifying complicated functions from JS like AJAX calls and HTML/CSS/DOM (Document Object Model) manipulation.

### 2.10 Web Servers

A web server represents a machine that runs software and can store and distribute web sites and web applications for people and users that are privileged to access them from a web browser.

The server must be able to accept HTTP requests, or the secure variant HTTPS, (Secure HTTP), process the requests and give responses accordingly to the requests.

#### 2.10.1 Apache HTTP Server

Apache is an open-source server software that handles almost 70% of all websites available on the world wide web. It is the most common choice on the market.

The most common operating system for Apache software to run on a server, is Linux, but later versions are also able to run on Windows and other UNIX operation system variations. For example, Apache version 2.4 can only support versions of Windows beyond *Windows 2000*.

#### 2.10.2 NGINX

Nginx, just like Apache is an open-source web server software, that also supports reverse proxy, HTTP cache and load balancer capabilities. Because it has its roots in performance and optimization, Nginx will often outperform other web servers like Apache, Microsoft IIS or LiteSpeed, in benchmark tests. Nginx is optimized to offer low memory usage and high concurrency where requests are handled in a single thread.

This is the best choice when serving static files like CSS, HTML, JavaScript, or images, because it is well-equipped in handling the load and client's requests.

It has been proven that it will perform more than 2 times faster than the competition when handling over 1000 simultaneous connections, therefore since its release in 2004 it has gained popularity over the market leader, Apache, and it is estimated that in the following years it will be the most chosen product in hosting websites and web applications.

## 2.11 Linux

Linux is an operating system that was released in 1991, designed for applications that are security-focused, unlike other operating systems like Windows and macOS that are more about the user experience. It is based on UNIX architecture and standards which provides a programming interface along with a user interface, even though Linux's functionalities are mainly accessed through its command line interface.

Statistically speaking, Linux consists of under 1% of the total computer users, Windows holding the crown with over 90% of the total user base. Linux based applications are mainly in the server related area, where companies choose Linux for their servers.

The market offers many variations and categories of Linux distributions. These are called Linux flavors and they are focused on a particular use and a particular application. In total there are roughly 300 Linux variations on the market.

### 2.11.1 Security-focused

- **Kali-Linux:** It is based on another flavor of Linux named Debian. The aim of this version is Penetration Testing and Security Auditing. This distribution comes pre-installed with several tools that are aimed towards various security tasks like Security Research, Penetration Testing and Reverse Engineering. There are a total of over 600 tools that are oriented for Penetration testing, and everything is offered for free.
- **Qubes OS:** The main advantage of this Linux flavor is that it enhances security against malware attacks by isolating personal files and Virtual Machines. In case of a malware attack data will not be damaged. Qubes OS require high experienced users because it comes with an advanced level of difficulty regarding system management, and it is not intended for beginners or even for intermediate skilled users.

### 2.11.2 User-Focused

- **Debian:** The first release of Debian is in 1993 making it the one of the first flavor of the original Linux from 1991. Unlike Ubuntu, Debian is lightweight because it does not have pre-installed packages, making it slightly faster than Ubuntu, even though Ubuntu is #1 choice among Linux distros.
- **Ubuntu:** This distro remains the most popular among Linux versions, making it very accessible for users that want to start using Linux and its powerful command line interface. There is also a low resource and low power version, Lubuntu, that is less demanding for the operating machine, and it is suitable for devices that are not powerful and are resource restrained. It is more suitable for battery powered applications.
- **Linux Mint:** For users that want to switch from other OSes (macOS or Windows), Linux Mint is a great idea because it comes preinstalled with the equivalent software found on the other operating systems. Mint, along with its other flavors: Cinnamon and MATE, are Ubuntu derived.

## 2.12 Raspberry Pi

Raspberry Pi is a single board computer developed by a UK-based foundation with the same name, in collaboration with the semiconductor manufacturing company, Broadcom. The idea behind this project is to offer a relatively powerful device that encourages students, and any other passionate man for that matter, to explore computing. Raspberry Pi boards offer a bridge to the electronics field via a set of GPIOs (General Purpose Input Output) pins that can, for example, make data acquisition from a sensor.

Some products from the Raspberry Pi's Foundation boards collection include capabilities like Bluetooth and Wi-Fi, that enables users to create IoT applications and projects that were not possible otherwise.

### 2.12.1 CPU Architectures

#### 2.12.1.1 CISC vs RISC

- **CISC** (Complex Instruction Set Computer) is a type of processors that has the purpose of completing a task by executing as many operations as possible in a single clock cycle.
- **RISC** architecture on the other hand has the purpose of simplifying hardware by making use of a simpler set of instructions that consist of a few simple steps.

#### 2.12.1.2 x86

This acronym has its origin from the first CPU model from Intel's processor series, 8086. After the release of 8086, come 286, 386, 486, 586, Pentium and up to today's series i3, up to i9, all having x86 as a suffix.

This standard was introduced in 1978 and it is among the first ISAs (Instruction sets architectures) on the market, and it is an implementation of CISC architecture.

#### 2.12.1.3 ARM

ARM (Advanced RISC machine) is an implementation for the RISC architecture in a micro-processor. Because it combines high-performance processing with low power consumption, it makes the best suitable processor for battery- or power-concerned applications. All mobile devices and tablets make use of ARM architecture to enhance efficiency and time between battery recharges.

#### 2.12.1.4 Raspberry Pi 4 specifications

In order to deliver high performance, as well as low power consumption, because some Raspberry Pi applications may be battery oriented, it may seem suitable that the Raspberry Pi series would be fitted with an ARM based processor. Therefore, all Raspberry Pi boards are suited with Broadcom ARM chips.

The 4<sup>th</sup> series of Raspberry Pi provides a Quad Core Cortex-A72 (ARM v8) processor that runs on 64 bit and has a clock speed of 1.5 GHz.

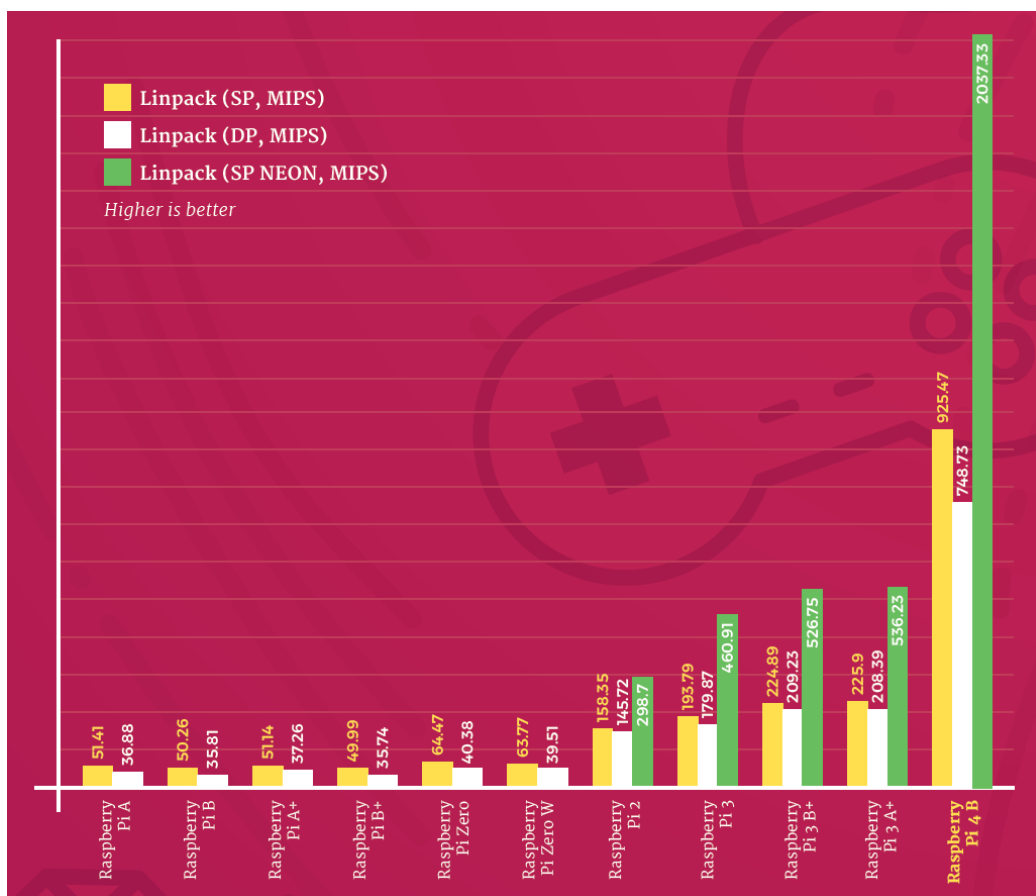
In combination to the ARM chip, comes a LPDDR4-3200 SDRAM in Raspberry Pi series 4 which are a huge improvement over the last series which featured only a DDR2

RAM. The options for the 4<sup>th</sup> series are ranging from an entry level 1GB up to 8GB RAM that offer a huge advantage at processing requests simultaneously and threaded applications.

### 2.12.2 Performance

The performance evolution subject was addressed by the Raspberry Pi official magazine in 2019 after the release of Raspberry Pi series 4. In the performed test, several performance factors were taken into consideration like CPU speed, GPU performance, image editing performance, file compression and so on. [8]

The big performance gap between the 4<sup>th</sup> Raspberry Pi series and the previous models is because the latest model is fitted with a DDR4 RAM memory with a clock speed of 3200 MHz and a Quad Core, 1.5GHz processor compared to the previous model that only featured an 800MHz DDR2 RAM memory.

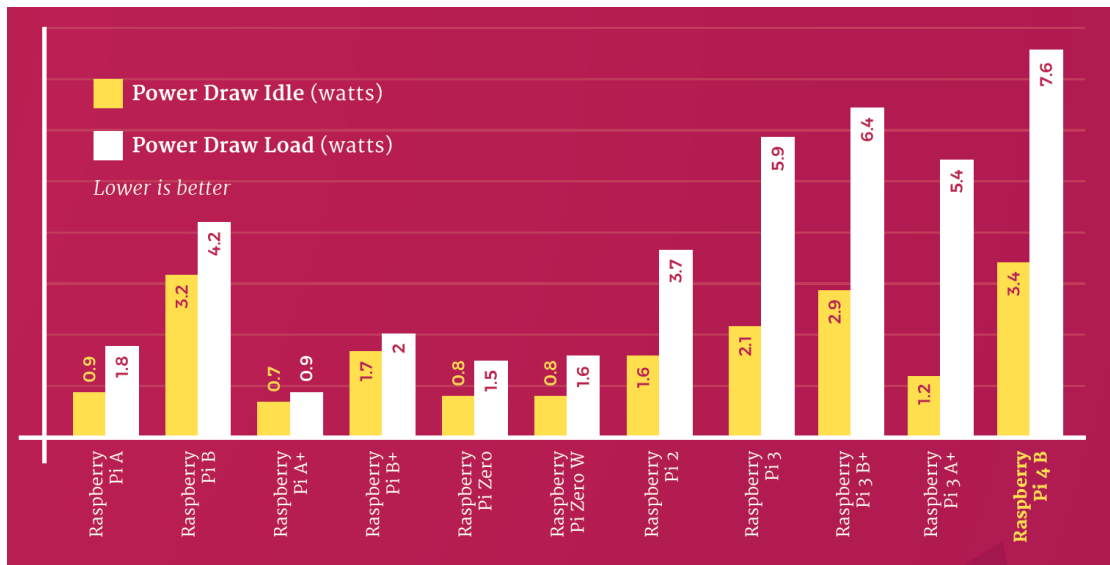


**Figure 2.11-Raspberry Pi Family Performance Comparison [8]**

### 2.12.3 Power Consumption

All Raspberry Pi products, overall have a very low power draw. As the generations passed, the power consumptions from the wall slowly rise, so that the most powerful Raspberry Pi, on peaks, consumes between 7 and 8 watts, consuming the most power among all Raspberry Pi products. But when compared to the performance it offers, it makes it the most efficient Raspberry Pi ever made. The Raspberry Pi's performance/watt is the best of all the products the Foundation has to offer.





**Figure 2.12 – Raspberry Pi Family Power Consumption [8]**

#### 2.12.4 Raspbian

Except Raspberry Pi Pico, all Raspberry Pi product run Raspbian, which is a Debian deviation that is optimized for Raspberry Pi products. Unlike Debian, which is a pure OS, Raspbian comes with pre-installed packages like GPIO drivers and libraries. Just like Debian and any other Linux distro, Raspbian offers a graphical user interface on which you can connect via a HDMI port or via Remote Desktop Connection with the use of SSH (Secure Shell) on Ethernet or Wi-Fi. Therefore, a Raspberry Pi needs to be addressed like any other Linux machine.

### 2.13 Firewall

The Firewall is a security system that was introduced more than 25 years ago and has the purpose of monitoring and impose restrictions for untrusted networks on the internet. A Firewall works based on a set of rules, in which are specified the ports on which a request may be placed and the ones that are restricted.

On the market are both paid and free Firewall package providers:

- Paid: Bitdefender, CISCO, Avast, Norton
- Free: ZoneAlarm, GlassWire, Windows Defender

### 2.14 Network Address Translation (NAT)

NAT is a system that is meant for more efficient use of IP addresses. By assigning a public IP address to the router, that will act as an internet agent, instead of having a separate public IP for every device in a network, we can conserve public IP address that come in a limited number. With this configuration, the only device that has direct access to the WAN (Wide Area Network) is the router. The rest of the devices (LAN) will access the internet through the router and not directly.

### 2.14.1 IPv4 vs IPv6

IPv4 (Internet Protocol Version 4) is a 32-bit internet addressing system that was introduced in 1981. By using a 32-bit system, this system can host a total of  $2^{32} = 4294967296$  addresses. By 1981's standards this number was enough to support all the devices on the planet, each having its own IPv4 address. But the engineers did not anticipate the exponential growth of devices over the years. Now almost every IPv4 addresses are in use, and engineers had to come with a new solution to overcome the IP exhaustion problem.

IPv6 is a 128-bit addressing system introduced in 1995 and can host a total of  $3.4 \times 10^{38}$  addresses which equivalates to 340 trillion trillion IP addresses. More than enough!

### 2.14.2 Public vs Private IP address

The public IP address is the address that is accessible over the internet (Wide Area Network) and is typically assigned to the router by the ISP (internet service provider). All the devices will connect to the router, therefore this will be the Local Area Network (LAN). The IPs on the local area network are not unique and can be found in every private network.

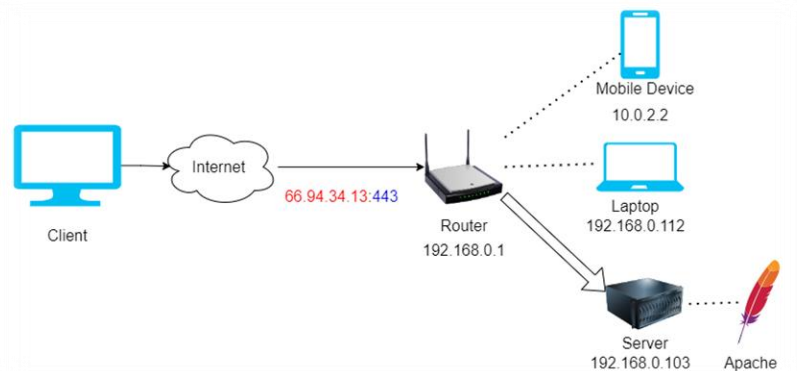
On the private network there are reserved IP addresses that have a certain role in the network. For example, in most cases, 192.168.0.1 is the IP destined for the router. By accessing this IP in the router, a HTML page will appear, through which all functionalities and configurations of the router can be accessed. Some functions like port forwarding, parental controls, DDNS configuration and so on.

There are 3 private IP classes:

- Class A: 10.0.0.0 – 10.255.255.255
- Class B: 172.16.0.0 – 172.31.255.255
- Class C: 192.168.0.0 – 192.168.255.255

### 2.14.3 Port Forwarding

Port forwarding is a way of allowing external devices to access a private network's devices and services. The way port forwarding works is by translating an external IP address and port to an IP on the private network. By using this method, devices become accessible on the internet for applications like server hosting or Remote Desktop Connection.



**Figure 2.13 – Port Forwarding Illustration**

## **2.15 DNS (Domain Name Systems)**

A Domain Name System is a Server that maps public IP addresses to domain names, to access servers and devices over the internet by a name that is readable and easier to remember, and not by public IP addresses. For example, when accessing a web page like [www.google.com](http://www.google.com) , a DNS will take over the request and will redirect that request to the public IP, mapped to the server (or load balancer) corresponding to Google's domain name.

A DNS has the purpose of narrowing the gap between humans and computers by making IPs more readable to humans. Without a DNS, instead of writing a domain name in the browser, a user would have to remember the IP address for every page he wants to access and write an IP address instead of a domain name.

### **2.15.1 Static vs Dynamic IP**

Static IP, as the name suggests, is an IP that is assigned to a network and will not change until the internet service provider decommissions the IP from the user. This approach is seen in server applications. Typically, static IPs require an extra charge by the ISP, and it is not so used anymore nowadays, because there are other approaches for the same network applications.

Dynamic IP is an addressing system that will regularly change the IP of the router based on DHCP (Dynamic Host Configuration Protocol) Server. The Server will allocate IP addresses to all the users commissioned to the ISP and will refresh, in a circular manner, all the allocated IP addresses. The IP change happens based on the ISP internal policy. The changes may appear once every 12 hours or once a month. Because of having a Dynamic IP allocation, port forwarding with an IP and port is not possible, because every time the IP change happens, the forwarding rules will become obsolete.

### **2.15.2 DDNS (Dynamic DNS)**

To be able to allocate a domain name to a dynamic IP address, a DDNS will be used to accomplish this. A DDNS allows to access a router that has been given a dynamic IP even if the IP has been changed by updating the change of IP in real time. Overall, this system is used in small home applications like CCTV surveillance cameras, Remote Desktop Connection, Home automation, Small Servers etc.

In order to make use of a more reliable system, companies will rather pay a monthly charge to be given a static IP from the ISP.

There are DDNS providers on the market that offer domain names for free: Dynu, FreeDNS, DynDNS, NO-IP and many others. Although some ISPs are offering DDNS services for free for its users.

## **2.16 SSL Certificate**

An SSL certificate allows sites to make the transition from HTTP to secure HTTP (HTTPS) by SSL encryption. When accessing an SSL encrypted website, first thing done is obtaining the public key of the server, which is a file obtained from the server which contains information about the identity of the server itself. By using SSL certification and encryption, traffic will not be visible to potential attackers and website identity is guaranteed.

An SSL certificate is obtained via a thrusted third-party CA (certificate authority) that has the purpose of generating and digitally sign SSL certificates that will be typically released in a matter of minutes. CAs will charge a fee for emitting the certificate, but there are also CAs that will provide free certificates for websites.

Examples of Certificate Authorities on the market:

- CloudFlare (Free and Paid)
- SSL.com (Paid)
- Let's Encrypt (Free)

## **2.17 QR Technology**

QR means “Quick Response” and it is the successor of barcode technology. Just like barcode technology, QR codes represent data in a visual manner so that scanner tools can quickly read and interpret contents of the code. This technology was first introduced in 1994 by a Japanese automotive company, a Toyota subsidiary. The technology's purpose was to track parts during manufacturing process and supply chain so that a new level of automation will be implemented.

Unlike barcodes, which have a one directional representation, QR codes will represent data in a 2D arrangement, a matrix format. Therefore, QR codes can be denser in terms of data packaging. Barcodes can have a maximum capacity of 85 characters, whereas QR codes can pack hundreds and up to thousands of characters while maintaining a small size.

The data typically stored in QR codes are website URLs, phone numbers, but can also store text up to 4000 characters or an equivalent of 4.3kb of alphanumeric characters or a total of roughly 7000 numeric characters.

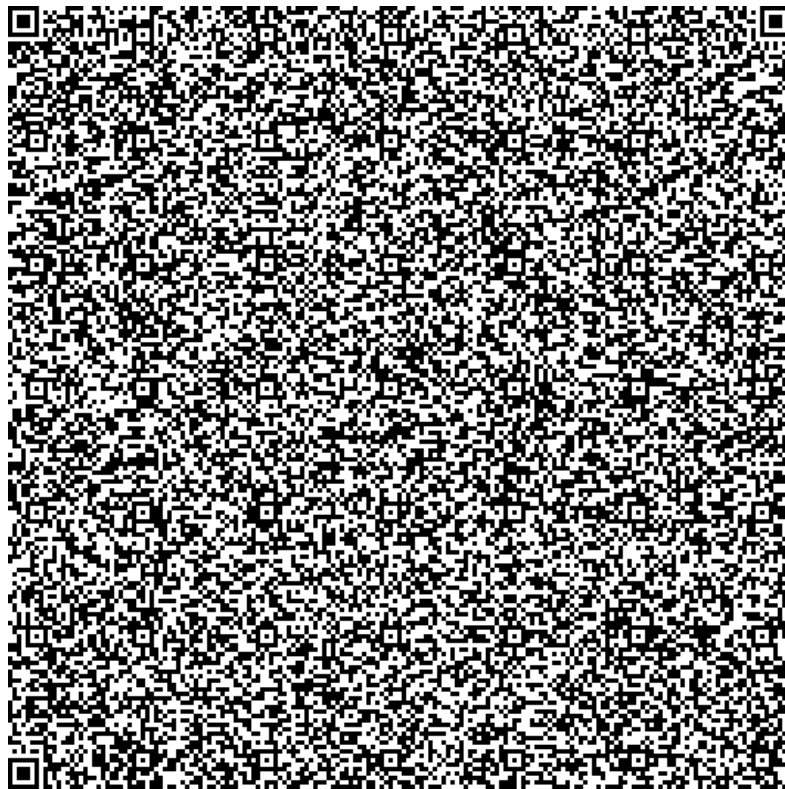
QR codes come in a total of 40 versions depending on the information capacity. Version 1 is composed of 21 x 21 modules matrix that can pack a total of 25 alphanumeric characters. Each version above will pack 4 more modules on every row so that Version 2 will be a 25 x 25 matrix and so on. The biggest version, Version 40, will be a 177 x 177 matrix that can support a total of 4296 alphanumeric characters.

This technology also features Error Correction Capability (ECC) that improves robustness of the scanning system, in case the code will get contaminated by external factors like water or dirt. There are 4 data restoration levels featured in QR code system

that can restore a total of 30% of the damaged code. A downside of the ECC feature is that it increases the amount of data stored in the code.



**Figure 2.14a – Version 1 QR code**



**Figure 2.14b – Version 40 QR code of General Context**

## 3 Analysis and Design

For the implementation of such an application, a client-server architecture and database support are mandatory. A platform like this will need the following components:

- Database: A system that will keep track of Restaurants, Orders, Employees etc.
- Server: An application that will handle and keep track of the platform's clients
- Mobile support: Application that will replace the traditional menu.
- Subscription Application: A web page that will allow new Restaurants to Subscribe to the program.
- Employees Application: A web page or native application that can allow Restaurant Staff to handle orders from clients in real time.

### 3.1 Back End

In order to develop an application that can sustain all the applications mentioned above, and appropriate server application must be developed with all the necessary functionalities and configurations.

The backend must support the following functionalities:

- Route the incoming traffic to the appropriate application
- Be able to communicate with the mobile application via JSON.
- Handle all the errors that may occur during operation.
- Handle all incoming HTTP requests from clients.
- Offer an interactive Front End for future customers to register.
- Hold a reliable Database system for the whole platform.
- Be able to accept data from consumers (clients) that would be processed and store it in the database.

We can implement all the functionalities mentioned above by splitting the whole program into smaller parts, that will make an easier job to maintain it and an elegant way to develop it, in a modular manner.

The Django Back End application will be divided in the folder structure from Figure 3.1 that will follow the functionality in a modular way.

As there can be seen, all the traffic will be routed to the corresponding application with the use of a configuration file that will handle all the routing and the global configuration of the program.

Finally, the program will arrive at the destination (the API that will serve any consumers that requested the service). There will be 5 applications available:

- Enroll App (handle all the requests to enroll new clients)
- Restaurant App (will distribute all the data regarding restaurant details)
- Orders App (store new orders or modify existing ones)
- Presentation App (will handle all the data regarding Front End)

- Personnel App (application that is offered to the staff of a restaurant)

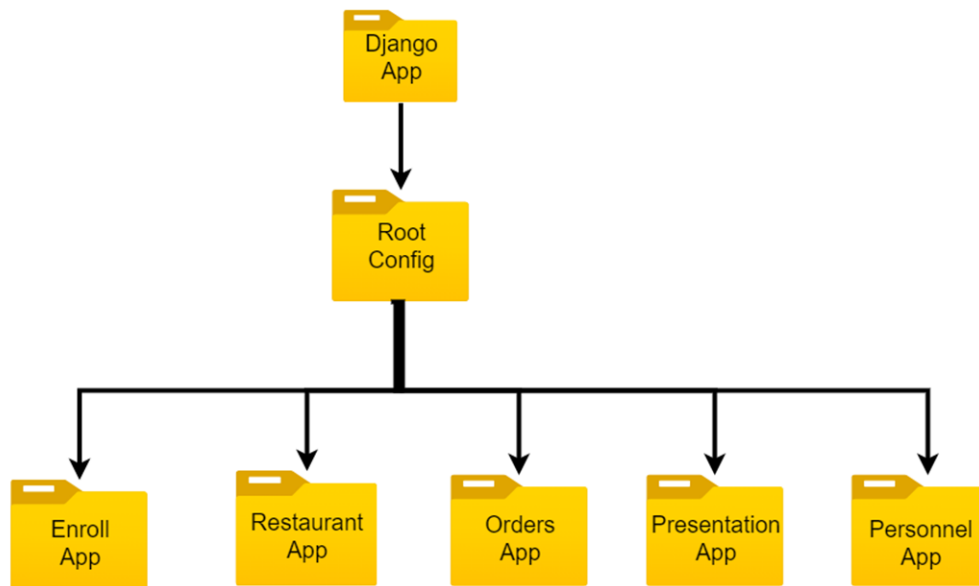


Figure 3.1 – Program Structure

## 3.2 Front End

For the Front-end part there will be three main pages. The users can firstly interact with the Qord through:

- Presentation page – this page will contain a general description of the program and the goals Qord is designed to accomplish (or tries to lay the groundwork for a platform that can sustain many features).
- Download page – a page that will offer the users a page to download the mobile application.
- Enroll page – a page (or can be referred as a portal) for restaurant owners that want to take part of the program. This page will contain a form that will request details about the restaurant and a general description, so that a join request will be created. In a production environment, this request will be taken over by a Qord employee that will continue the process of subscription.

The Front-end application will be developed using HTML, CSS, and JavaScript. In order to make the application mobile compatible, Bootstrap was used to ensure a layout based on columns.

### **3.3 Database**

Because the application needs to keep track of many details from multiple customers, restaurants, orders and as well as restaurants employees, a database infrastructure is mandatory.

Just like the Back-end functionality is split in small, modular parts, the database structure needs to obey a similar approach. Therefore, the database will be designed in a similar way:

- 1) The RestaurantApp will be responsible for handing all the details regarding a particular restaurant, along with all the data that is associated with the menu along with other informational aspects. A restaurant menu should be structured as following:
  - A restaurant should contain many food categories.
  - A food category should contain items (Pizza Calzone, Coca-Cola Zero etc.)
  - A menu item should have fields like Price, Calories and Weight
  - A ingredient list should be associated with a menu item.
  - A allergens list should be associated with a menu item.
  - A restaurant should have tables with IDs and associated indexes.
  - Other information like Smoking permission, Description etc.
- 2) OrdersApp should contain a database that handles all the order details placed by a client from the restaurant app. The details an order should contain are:
  - A list of all the items ordered.
  - A payment method
  - A payment request that is initialized from the app (like a check)
- 3) Enroll App is an application that will be available under a form from the Front-end. All the fields of the form will be stored in the database when the form will be submitted.
- 4) The database associated with Personnel App will give the ability to the administrator to keep track of the Restaurant Staff. The database for this app will contain various information about the workers and will, perhaps in the future, feature a time management tool for the employees. The stored data will mainly be to offer information and statistics about the people.

### **3.4 Mobile Development**

The mobile development for the platform was developed in Flutter. Flutter obeys a development methodology oriented on widgets, which are basically small components that can act individually or contribute to create more complex components, adding to the reusability of the code.



There are some key functionalities that the mobile application must be able to accomplish for it to deliver the quality and usability expected by the customers and restaurant administrators.

These are the main features that must be included to form the core of an application that can be used at a large scale:

- Make HTTP requests to the internet so it can retrieve data and be constantly updated. It should be able to access a server when a certain task is asked for.
- It should be able to create an interactive user interface based on the structure of the menu that is fetched from a server.
- The application must be able to have QR scanning capabilities, be able to read a code and based on it to fetch data from the server.
- Have a modern design that will appeal to the customers to make them faster embrace the idea that mobile ordering is a better way.
- Offer an intuitive workflow that will make an easy job even for new clients to find their way through the application.

The first page of the application should greet the users with a hybrid between a brief description and a slogan about the application and a starting point for the users that will consist of a button that will route the users to the scanning page.

The users will scan a code that is displayed on the restaurant table with a sticker or with a device that will perform a QR code refresh after every order is closed enhancing the overall security of the application.

After the QR code is detected, a HTTP request will be performed to retrieve data about the menu. Based on a JSON data structure retrieved from the server, an interactive menu page will be created that will enable users to see a detailed description about foods and to add items in an order list interactively.

When the order list is complete an order will be placed, and the user will be routed to a so-called “waiting lobby” where real time details about the order will be displayed, and a status will provide info for the users so that they will stay updated.

From now on, the user has the option to pay for the order or to add new items to the existing order. The user must be able to choose a payment method and make a payment request at any given time during ordering.

### **3.5 Programs Used**

- PyCharm (Django development)
- Android Studio (Flutter development)
- Visual Studio (HTML, CSS, JS development)
- Photoshop (User experience)
- Postman (HTTP requests testing)
- PuTTY (secure shell between Raspberry Pi)
- WinSCP (File Transfer with Raspberry Pi)

- Remote Desktop Connection (User interface with Raspberry Pi)
- Draw.io (program for creating diagrams)
- Microsoft To-Do (tasks organization)
- Git (Versioning Control System)
- Beyond Compare 4 (data comparison utility)

### 3.6 Use Case Diagram & Sequence Diagram

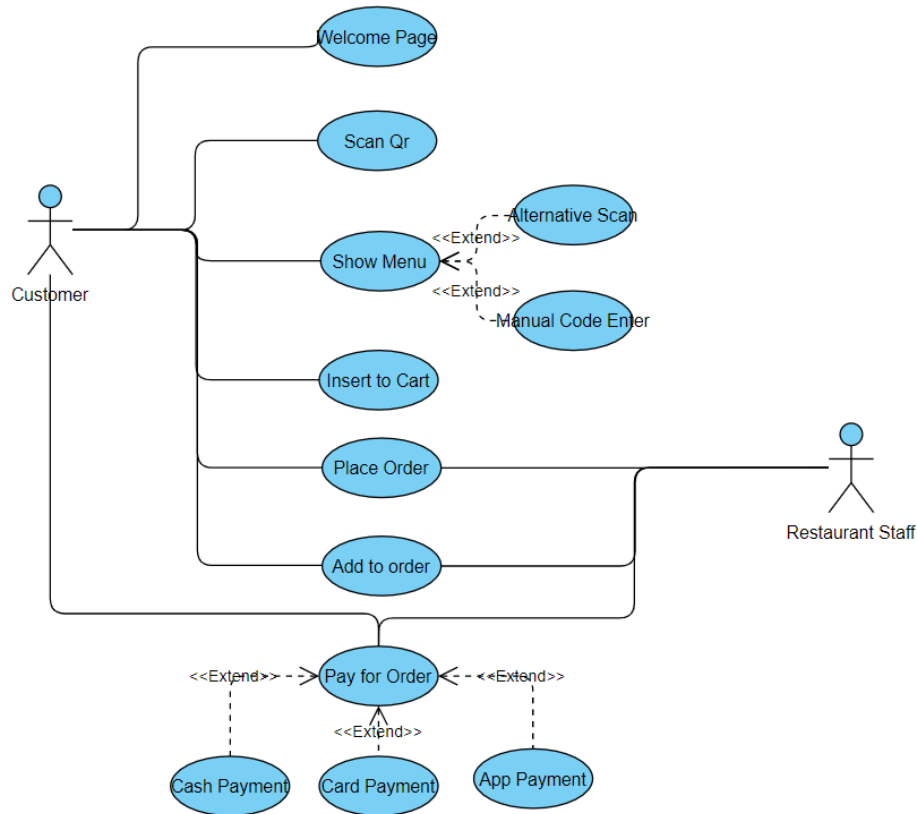


Figure 3.2 – Use Case Diagram

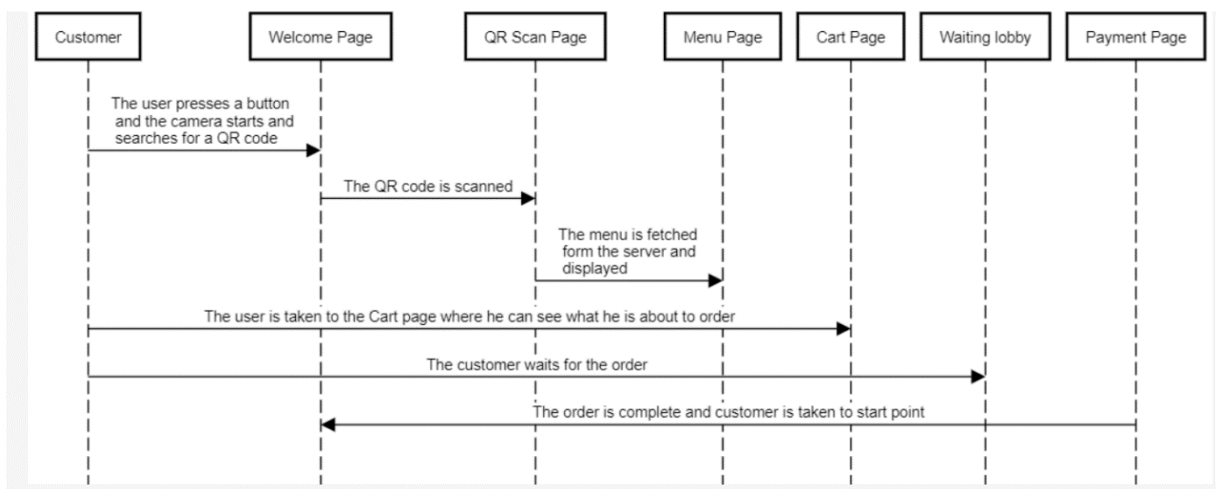


Figure 3.3 – Sequence Diagram

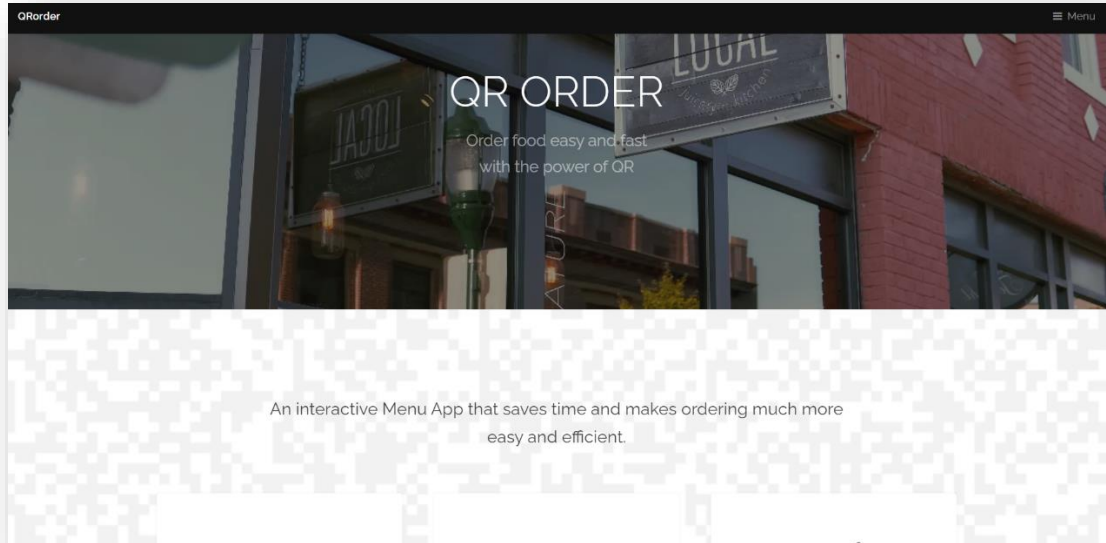
## 4 Implementation & Development

### 4.1 Front End Development

For the front-end implementation HTML, CSS and JavaScript were used. The application's presentation site is consisted of 3 pages that are located on 3 separate roots:

- /qrpresentation – page that holds the “welcome” page.
- /download page – a page that mainly consists of a download link for the mobile application that is available for android as well for iOS.

- /register – a page that will allow new restaurants to register.



**Figure 4.1 – Qord Presentation Page**

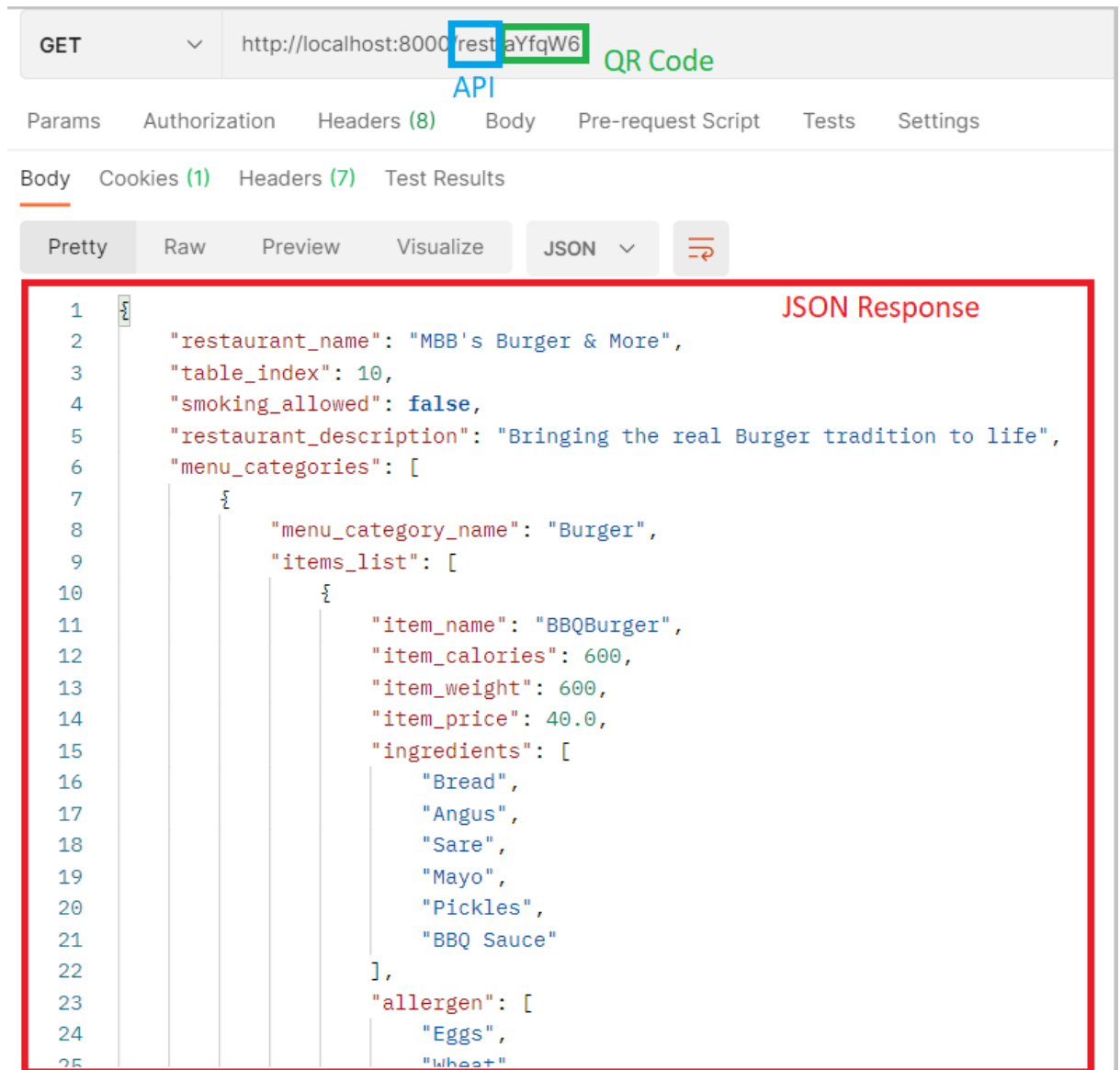
A screenshot of a web form titled 'Register Restaurant to QrOrder'. The form is centered on a light orange background. It contains three input fields: 'Restaurant Name', 'Owner Email', and 'Owner Phone Number'. Below these fields are two buttons: 'Enroll Restaurant' and 'Reset'.

**Figure 4.2 – Register Restaurant Page**

## 4.2 Back-end Development

As mentioned in the chapter above, the backend application must be able to accept HTTP requests from clients and handle them accordingly and be able to store data in a database. All of this is managed with help from Django Framework that is able to both handle HTTP requests and has SQLite database support.

The server-client communication is done via JSON objects. For example, when a QR code is scanned, a JSON response will be sent. In case of RestaurantApp, the response JSON will contain all the details about the restaurant and the menu. By using Postman, we can visualize the response:

**Figure 4.3 – Restaurant App JSON Response**

Every time a request is made from a client, Django will log the outcome and display information about the request like date and time, route of the request, status code and type of the request (GET, POST etc.):

```
System check identified no issues (0 silenced).
June 28, 2021 - 00:04:21
Django version 3.1.7, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[28/Jun/2021 00:04:23] "GET /qrpresentation HTTP/1.1" 301 0
[28/Jun/2021 00:04:23] "GET /qrpresentation/ HTTP/1.1" 200 3716
```

**Figure 4.4 - Requests Console**

For example, in Orders App (the part of the project that handles the order part), POST request is used to send the items and quantity so that an order is built. When an order is sent, the system will put the order in the database. An order will have the following template:

```
{
  "payment_method": "Card",
  "payment_request": false,
  "orders":[
    {
      "item_name" : "Barbeque Burger",
      "quantity" : 2
    },
    {
      "item_name" : "Quattro Stagioni",
      "quantity" : 1
    },
    {
      "item_name" : "Coca-Cola",
      "quantity" : 4
    }
  ]
}
```

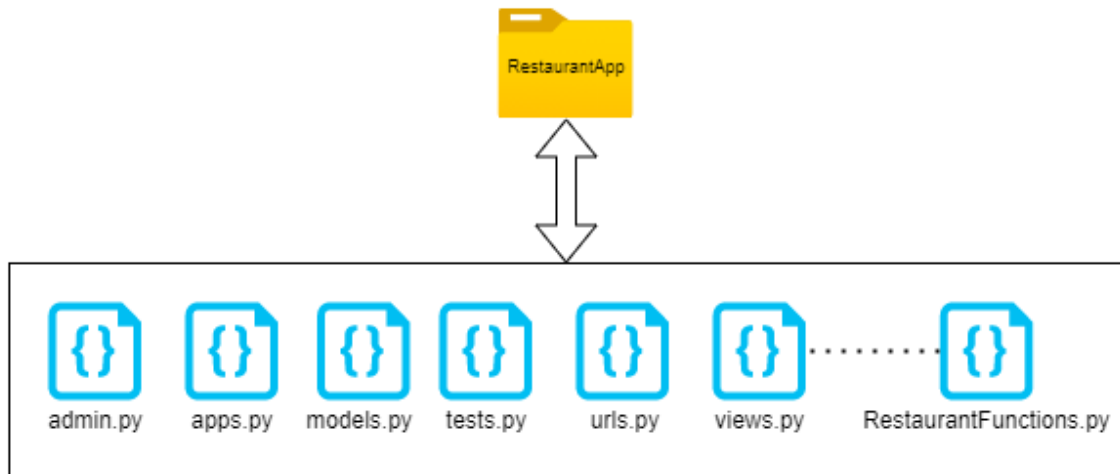
**Figure 4.5 – Order Model JSON**

An order will contain the following information:

- Payment Method (an option for payment that can be Cash, Card, App Payment)
- Payment Request (A flag that indicates if a client requested the check)
- A list of all the items that have been ordered by the customer (an order object will contain the item's name or ID and the quantity of that order that has been ordered).

To keep a clean and disciplined code writing, some rules had to be obeyed, therefore the views.py file will be kept only for delivering the responses and returning

the HTML pages, while the processing will be kept in a separate file. For example, in RestaurantApp, the folder structure is the following:



**Figure 4.6 – RestaurantApp Folder Structure**

In a Django App *views.py* is the file responsible for the API methods, but in this case, a new file called *RestaurantApp.py* is created, to handle all the processing that needs to be returned so that a clean and comprehensive project structure will be maintained. For example, when a QR code will be scanned, a JSON file will be returned that is built based on data fetched from a database. The function that builds the JSON has well over 50 lines of code and can be seen in Figure 3.9. Having all this in mind, it would make sense to have a separate class that will expose its methods to make the main API's file (*views.py*), easier to understand.

Having this done, a response function will look something like this and would not need to include all the code from Figure 3.9:

```

19  @csrf_exempt
20  def return_restaurant_data(request, code):
21      try:
22          if request.method == 'GET':
23              # time.sleep(2)
24              return JsonResponse(data=RestaurantInterface.create_Restaurant_JSON_data(code))
25          if request.method == 'POST':
26              # get_token(request)
27              return HttpResponseNotAllowed('Not allowed')
28      except TableId.DoesNotExist:
29          return HttpResponseNotFound()

```

**Figure 4.7 – Response method example**

```

65     @staticmethod
66     def create_Restaurant_JSON_data(table_code):
67         # Acquire needed objects
68         table_object = TableId.objects.get(table_unique_code=table_code)
69         restaurant_object = table_object.restaurant
70
71         restaurant_name = restaurant_object.restaurant_name # Restaurant_name
72         table_index = table_object.table_number # Table Number
73         smoking_permission = table_object.smoking_allowed
74         restaurant_description = restaurant_object.restaurant_description
75
76         # gets all food categories of the restaurant (by id) and stores in querySet
77         food_categories_querySet = FoodCategory.objects.filter(restaurant=restaurant_object.id)
78
79         # gets all food items of the restaurant (by id) and stores in querySet
80         all_food_objects_in_menu = MenuItem.objects.filter(restaurant=restaurant_object.id)
81
82         # Create JSON restaurant data return Object
83         ret_dict = {
84             'restaurant_name': restaurant_name,
85             'table_index': table_index,
86             'smoking_allowed': smoking_permission,
87             'restaurant_description': restaurant_description,
88             'menu_categories': [],
89         }
90
91         for fc in food_categories_querySet:
92             food_per_category = all_food_objects_in_menu.filter(food_category=fc)
93             food_list = []
94             for mi in food_per_category:
95                 food = {'item_name': mi.food_name, 'item_calories': mi.food_calories, 'item_weight': mi.food_weight,
96                        'item_price': mi.food_price, 'ingredients': [], 'allergen': []}
97
98                 ingredient_list_per_food = FoodIngredient.objects.filter(food_item=mi)
99                 for ing in ingredient_list_per_food:
100                     food['ingredients'].append(ing.ingredient_name)
101
102                 allergen_list_per_food = FoodAllergen.objects.filter(food_item=mi)
103                 for alg in allergen_list_per_food:
104                     food['allergen'].append(alg.allergen_name)
105
106                 food_list.append(food)
107
108             food_categ = {
109                 'menu_category_name': fc.food_category_name,
110                 'items_list': food_list
111             }
112             ret_dict['menu_categories'].append(food_categ)
113         return ret_dict
114

```

Figure 4.8 – JSON processing function

### 4.3 Mobile Development

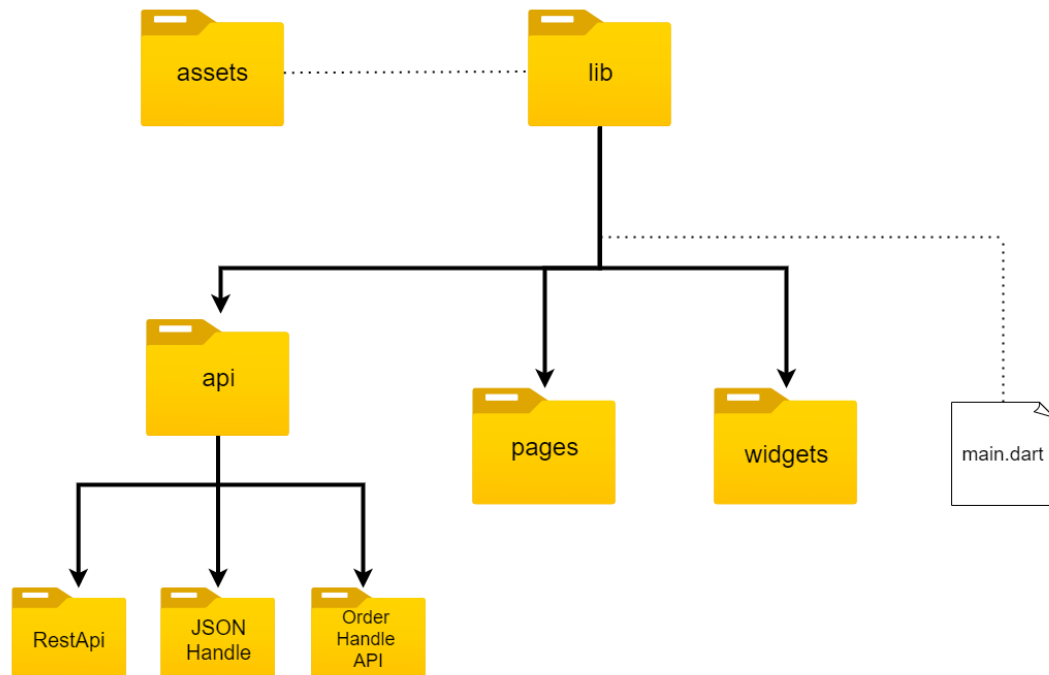
The development for the Mobile application was done in Android Studio IDE with the help of a built-in Virtual Machine that Android Studio included.

Just like the Django application, the Flutter application will maintain a structure that is based on layers. A layer will represent a set of methods grouped in a class that will handle a job like:

- HTTP requests
- JSON manipulation
- User interface rendering
- Widget build-up
- Page routing



Each layer will have its own class and will be placed in separate file so that we will obtain a folder structure like this:

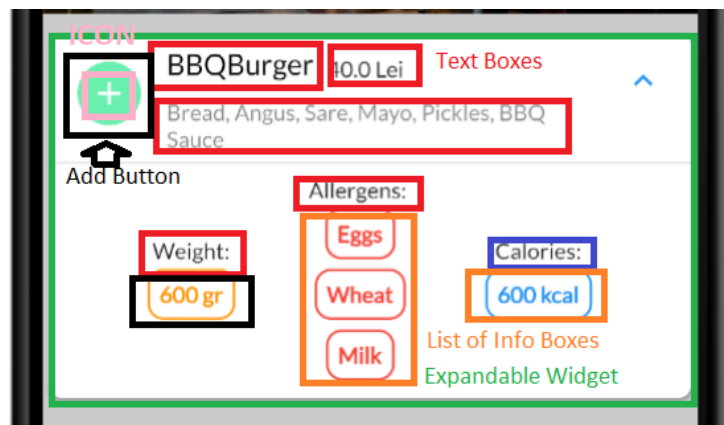


**Figure 4.9 – Mobile Application Folder Structure**

The *lib* folder is basically the root of the application. There will be the code for the whole application. There are three folders that will group three functionalities:

- *api* folder – will be responsible for all the functionalities of the application (http requests, JSON manipulation etc.)
- *widgets* folder – is the folder that will hold all components (widgets) that are meant to build the application pages.
- *pages* folder – folder that will have all the final pages that will build the application.

As mentioned in Chapter 3, the user interface must be dynamically rendered based on data fetched from the server. To achieve this, a reusable code format will be needed. For example, the following component that builds the user interface is reused. This is a Food item widget that is built from many components like text items and buttons.

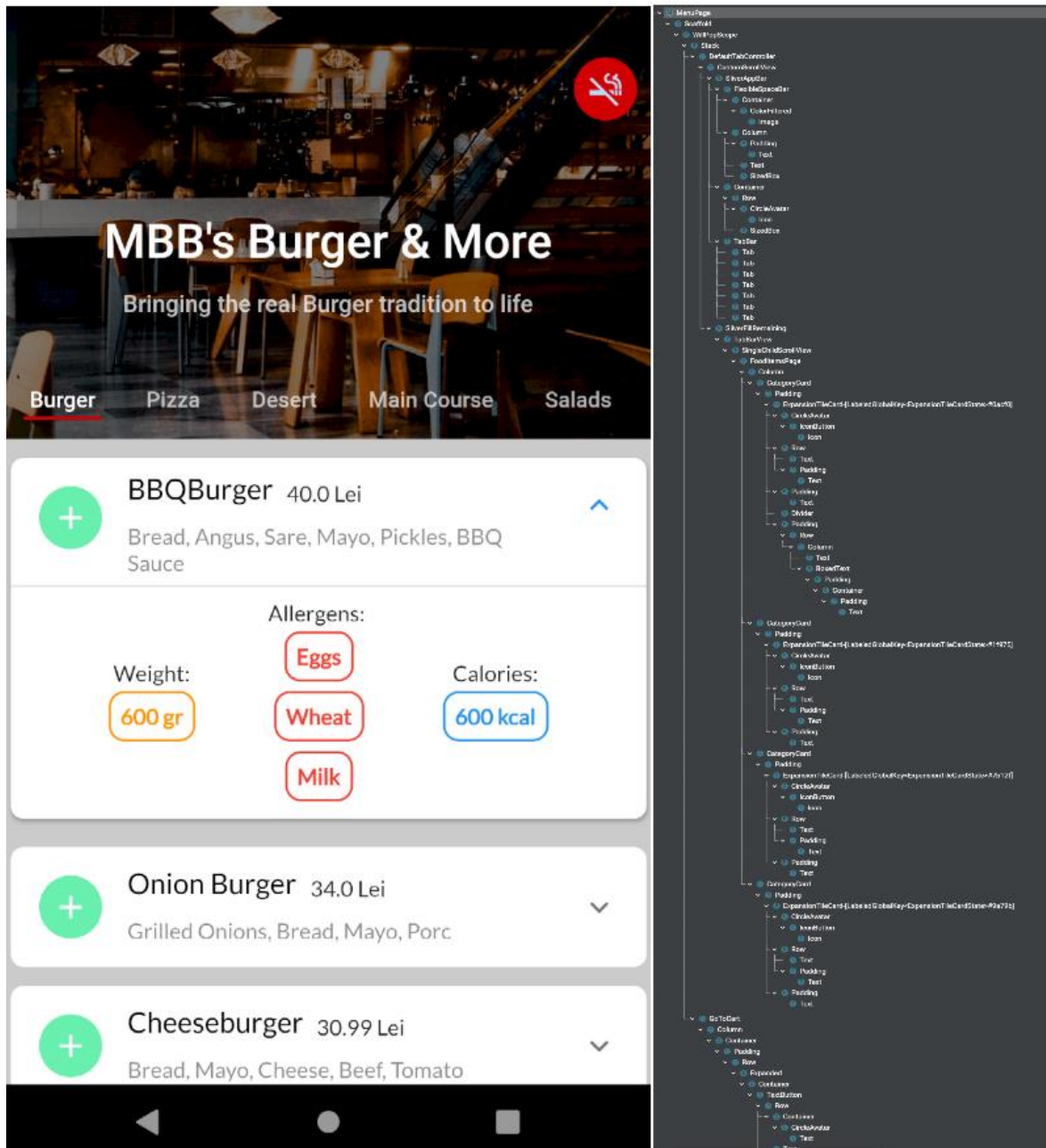


**Figure 4.10 – Illustrated Widgets**

The widget tree that builds a page will consist of many widgets, which can be built themselves from other widgets. For example, the page that will display the menu of the restaurant will have the following widget tree that is displayed in the picture below.

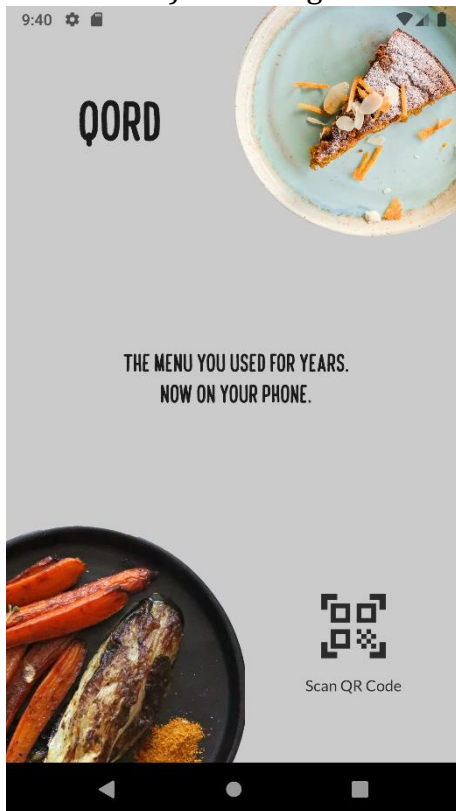
There are tens and sometimes hundreds of widgets that build a Flutter application and are nested to form a widget tree.

### Figure 4.11 – Menu page and its widget tree



The mobile application has a total of 10 pages. The start script of the application is *main.dart* file represents a router for the rest of the pages which in Flutter comes under the name of *Navigator*. The pages are labeled below:

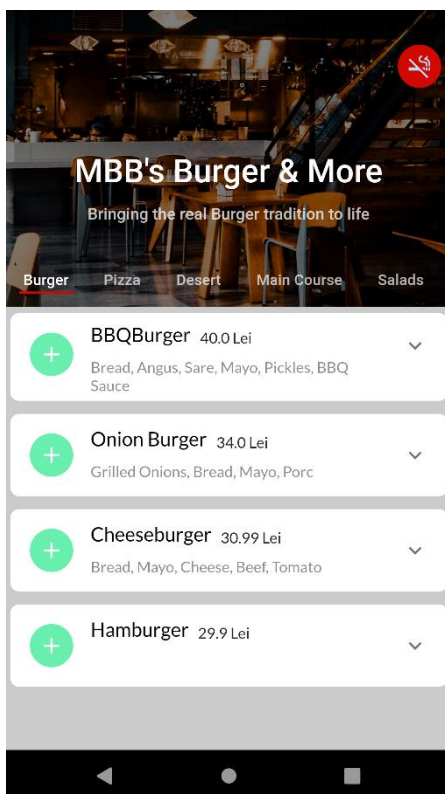
1) Start Page



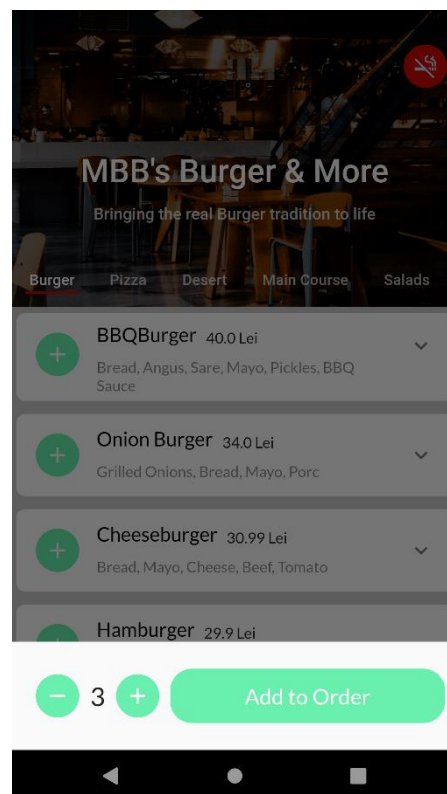
2) QR code scan page



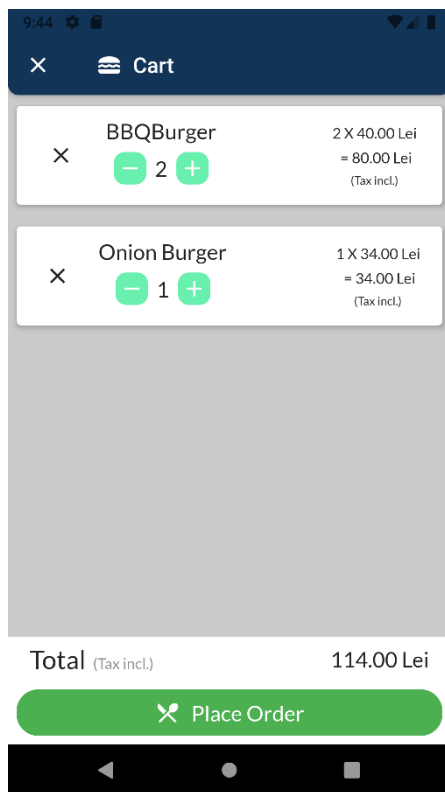
3) Menu Page



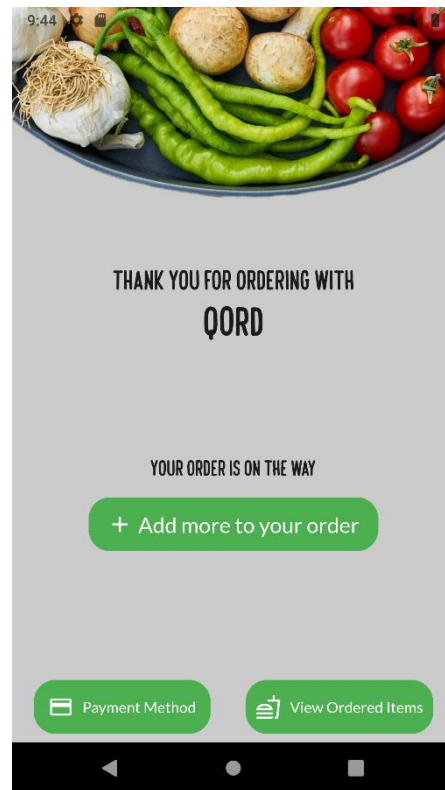
4) Select item



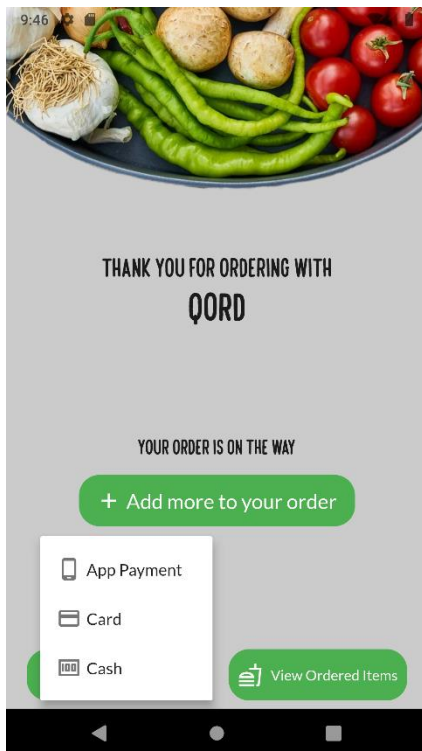
### 5) Cart Page



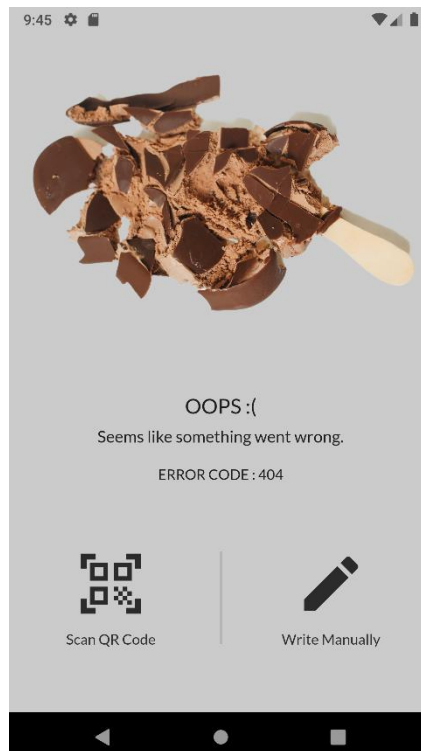
### 6) Waiting Lobby



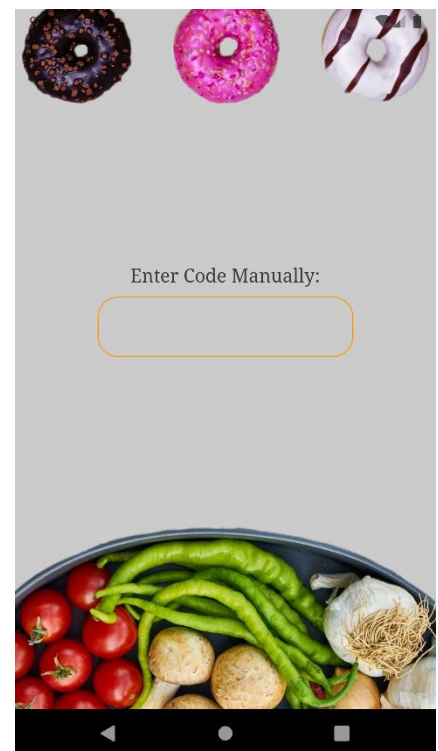
### 7) Select Payment



### 1.1) Alternative Scan



### 1.2) Manual Code Entry

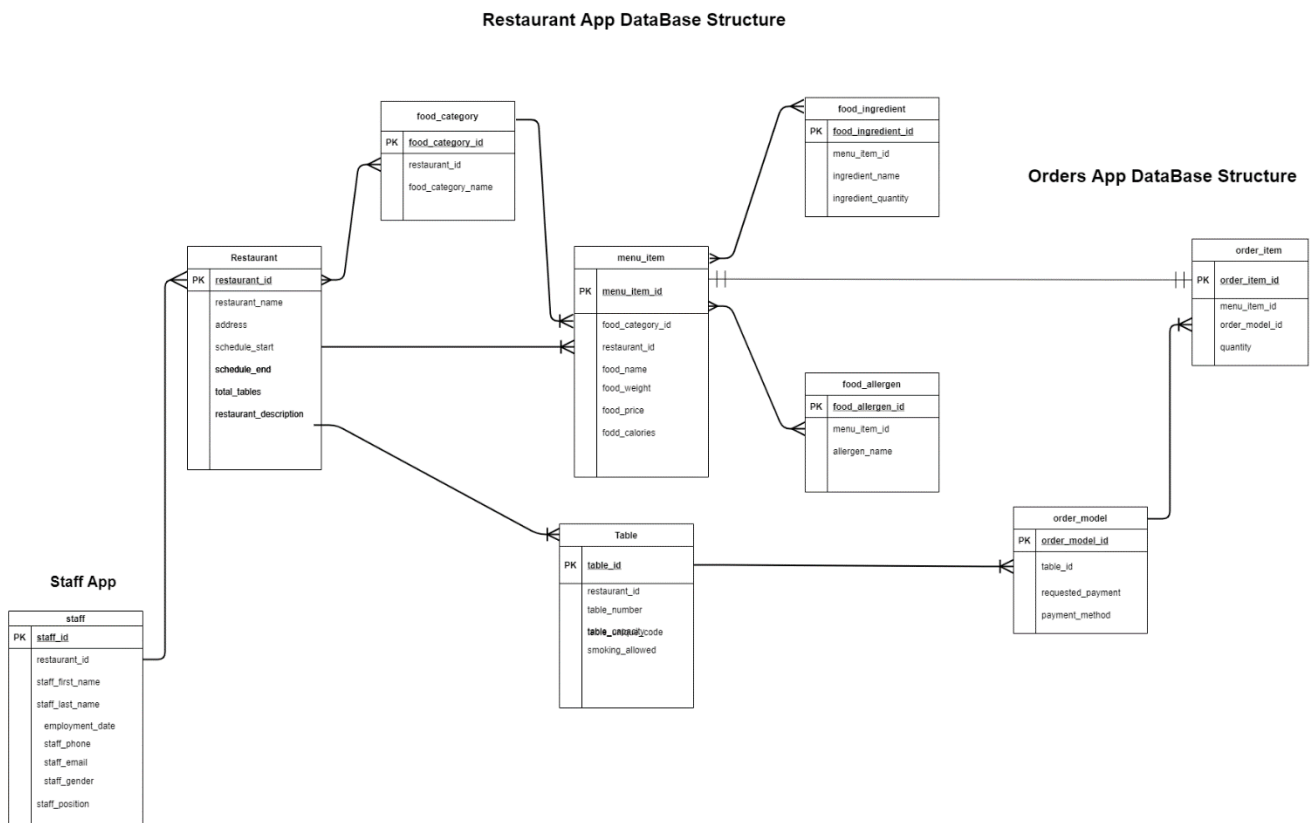


## 4.4 Database Implementation

In Django, databases are organized based on models. When we start a new module in our application, a *models.py* file will be created along, which will contain all the models that are associated with that application. Models can be seen as fields of a database table and are written as classes in *models.py* file.

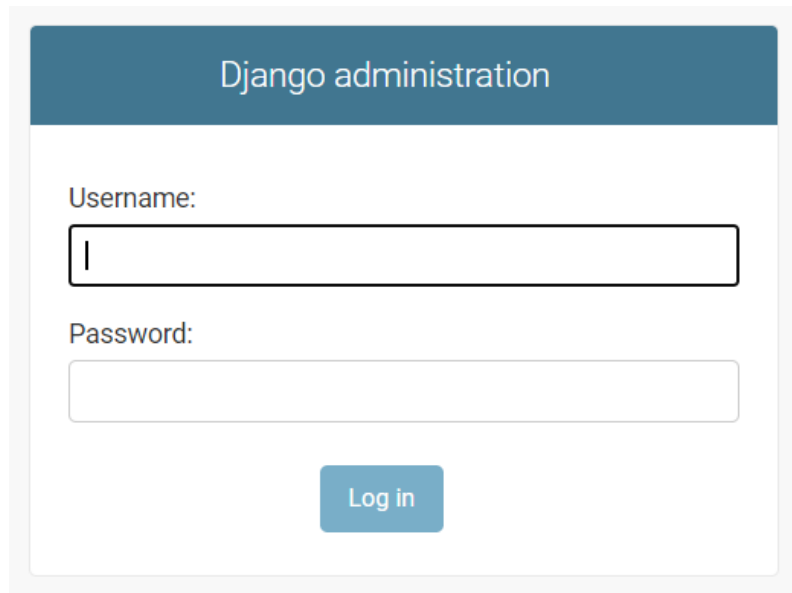
The Entity Relationship Diagram (ERD) of the whole application can be seen in Figure 3.13 and consists of 4 main applications:

- Restaurant App database (RestaurantApp/models.py)
- Orders App database (OrdersApp/models.py)
- Staff App (PersonnelApp/models.py)
- Enroll App (EnrollApp/models.py)



**Figure 4.12 – Qord Database Representation**

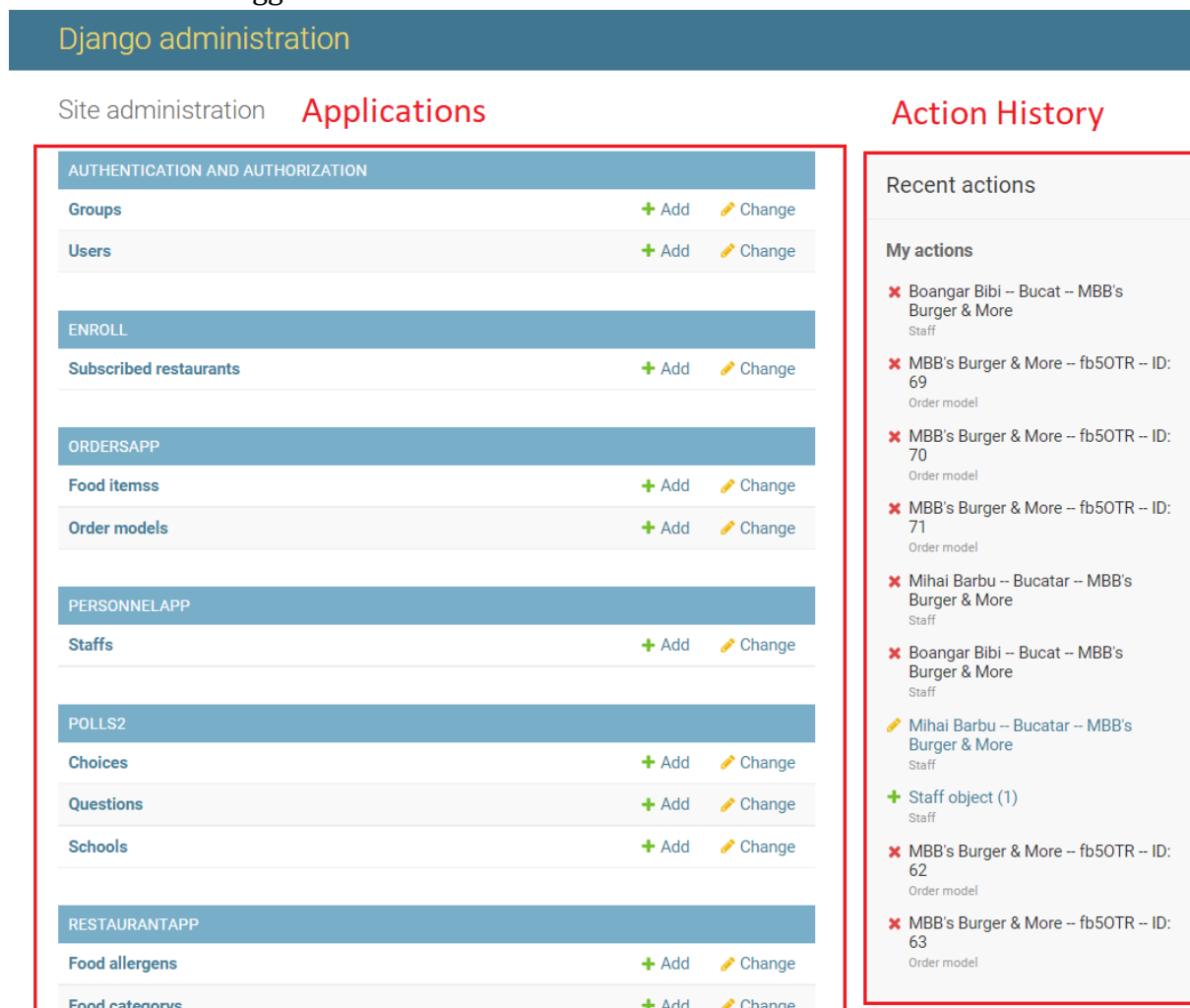
Django Framework also offers a built-in interface for debugging. Administrators can create an account with the command *python manage.py createsuperuser*, afterwards you can log into the administrator interface on this page that you can access through the browser:



The image shows the Django administration login page. It has a dark blue header with the text 'Django administration'. Below the header, there are two input fields: 'Username:' and 'Password:'. The 'Username:' field contains a single character 'l'. Below the password field is a blue 'Log in' button.

**Figure 4.13 – Administrator login page**

If login process was successful, the following page will appear and will provide a user interface for data visualization and must be disabled in production as Django documentation suggests:



The image shows the Django administration dashboard. It has a dark blue header with the text 'Django administration'. Below the header, there are three main sections: 'Site administration', 'Applications', and 'Action History'.

**Site administration**

- AUTHENTICATION AND AUTHORIZATION**
  - Groups: + Add, Change
  - Users: + Add, Change
- ENROLL**
  - Subscribed restaurants: + Add, Change
- ORDERSAPP**
  - Food itemss: + Add, Change
  - Order models: + Add, Change
- PERSONNELAPP**
  - Staffs: + Add, Change
- POLLS2**
  - Choices: + Add, Change
  - Questions: + Add, Change
  - Schools: + Add, Change
- RESTAURANTAPP**
  - Food allergens: + Add, Change
  - Food categories: + Add, Change

**Applications**

**Action History**

**Recent actions**

**My actions**

- ✗ Boangar Bibi -- Bucat -- MBB's Burger & More Staff
- ✗ MBB's Burger & More -- fb5OTR -- ID: 69 Order model
- ✗ MBB's Burger & More -- fb5OTR -- ID: 70 Order model
- ✗ MBB's Burger & More -- fb5OTR -- ID: 71 Order model
- ✗ Mihai Barbu -- Bucatar -- MBB's Burger & More Staff
- ✗ Boangar Bibi -- Bucat -- MBB's Burger & More Staff
- ✗ Mihai Barbu -- Bucatar -- MBB's Burger & More Staff
- + Staff object (1) Staff
- ✗ MBB's Burger & More -- fb5OTR -- ID: 62 Order model
- ✗ MBB's Burger & More -- fb5OTR -- ID: 63 Order model

**Figure 4.14 – Administrator Page**

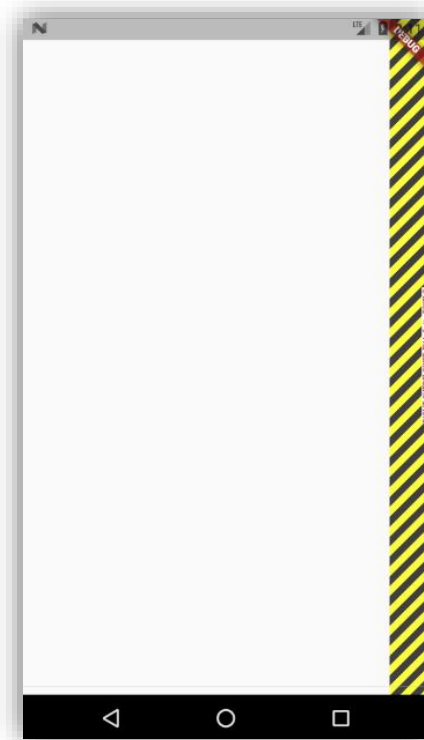


## 5 Testing and Validation

The development was mainly done on the personal computer and when a certain checkpoint was reached (a certain functionality was complete or reached an advanced state), a deployment takes place which represents uploading the new code to the server (Raspberry Pi) and making an end-to-end testing that mainly consists of manual testing.

In a deployment, the mobile application was uploaded on multiple devices so that rendering bugs would be analyzed so that the problem could be tracked and fixed.

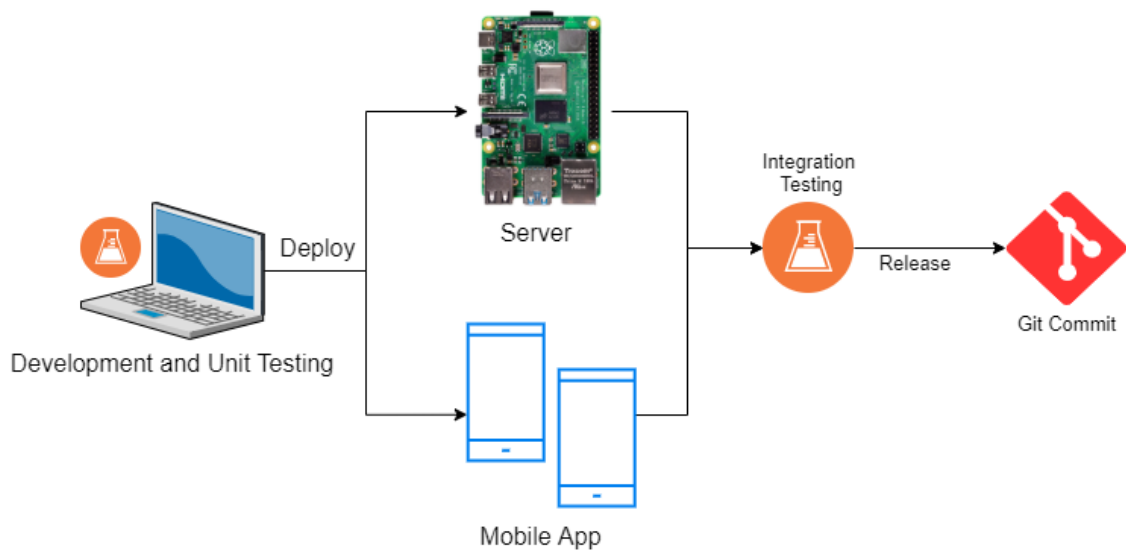
A common issue that was encountered in the development was the RenderFlex overflow which means that the widgets of the application does not fit on the screen. When this happens, a yellow and black stripe indicator that shows the number of pixels that could not be displayed on the page. The reason of this problem is that, depending on the resolution of the mobile phone, the widgets can be rendered slightly different, therefore bugs may appear.



**Figure 5.1 – RenderFlex Overflow example**

After all the tests are done, the code will be committed to GitHub and in case of blocking points, an older commit can be downloaded from GitHub and a new path could be evaluated.

The workflow for this project can be seen in the following Figure:



**Figure 5.2 – Workflow Diagram**



## 6 Conclusions

### 6.1 Contribution summary

This project is aimed to implement the structure of a platform that can be used on a large scale by a number of restaurants and provide a reliable alternative to traditional ordering. For this project to be in production, the security part must be reevaluated and benefit from a rigorous test analysis for both bugs and security vulnerabilities.

Also, from a production point of view, the whole platform needs to be tested from the user's perspective and be open for any suggestion from a user community and based on a suite of opinions to reevaluate the application workflow and user experience in order to ensure the success of the project on the competitive market.

From the current state of the project, even though the foundation of the application is developed, for it to be in a production environment, there is still a lot of effort to be submitted in order to deliver the best possible product to the market.

### 6.2 Obtained Results

In the end, the results are the following:

- Implemented a service that can handle multiple users at once on a personal Linux machine on the personal home network.
- Implemented a mobile application that can be supported by both Android and iOS with help of a cross-platform application offered by Google that is able to create dynamic user interfaces based on data fetched by the server.
- Implement a reliable connection and data transfer between server and all clients using the mobile application.
- Designed a database infrastructure that can hold information about restaurants, restaurant's personnel, and orders, that is hosted on the local server.
- Host a presentation site that is meant to deliver, to the customers, the idea behind this project and encourage people to embrace this change.

### 6.3 Optimizations

Optimizations and improvements can be made on various fields for it to be placed in a production area:

1) Security:

- The server application can benefit from an end to end security analysis to ensure information is safely stored and ensure, in case of a breach, the data will not be corrupted or stolen. Also, periodical data backups must be performed to ensure the reliability and security of the system.

- Provide SSL certificate for the domain that is used for the whole platform to ensure encrypted data transmission.
- 2) Server-side:
- Ensure the reliability of the server application with an organized test suite that will guarantee the correct functionality of the data handling and processing of the server application.
  - Create a data logging infrastructure that has the purpose of tracking user actions and preferences to further improve the application, based on user problems and actions during use.
  - Enhance the server application with more functionality to add to the interactivity of the program.
- 3) Front end:
- Provide a presentation platform for restaurants to make a presentation about themselves and give potential clients a brief understanding about restaurant's services. This platform must be based on a template that contains a presentation page, another page which displays the menu and other pages based on the needs of the restaurants. This front-end idea will come as a bazaar like idea to the restaurants that are Qord partners, just like Foodpanda or Glovo advertises restaurants in the application.
  - Collaborate with an application designer to give personality to the whole project.
- 4) Mobile application:
- Provide a restaurant advertising page inside of the app so that clients can visualize the location and menus of the Qord partners. This feature will add to the functionality of the application and will provide a restaurant user experience before the client visit the restaurant.
  - Provide an order history for the clients to visualize the items they ordered the most.
  - Offer the ability to pay within the application itself and to send the receipt on email, so that another traditional ordering step is replaced.
  - Provide a HELP feature to the application so that whenever a client is confused about the application or needs information, a waiter will come to help the client.
  - Implement a tour feature of the application so that clients will get an introduction about the usability of the application.
  - Develop a feature that allows users to pay for other people at the same table or to pay for the whole order.
- 5) Administrator Application:
- To make the application viable for a wide range of restaurants, it should cover more tasks in a restaurant. For example, the application should provide a regular report to the administrator about the items that were ordered by the clients.

- Add a feature that will automate the financial maintenance of the restaurant and make automatic taxes payments as much as possible.

Overall, the application needs a lot more effort to become a viable option for the restaurant's customers. Because the restaurants come in a wide variety and the offered service come in a wide range, the application must be flexible and must cover the needs of as many restaurants as possible.

## 7 Bibliography

- M. M., „<https://leftronic.com/>,” Leftronic, 19 04 2021. [Interactiv]. Available:  
1] <https://leftronic.com/blog/android-vs-ios-market-share/>. [Accesat 03 06 2021].
- F. Blog, „<https://fortyseven47.com/>,” FortySeven, 16 10 2020. [Interactiv].  
2] Available: <https://fortyseven47.com/news/best-programming-languages-for-mobile-app-development/>. [Accesat 03 06 2021].
- Google, „<https://flutter.dev/>,” Google, [Interactiv]. Available:  
3] <https://flutter.dev/docs/development/ui/widgets-intro>. [Accesat 04 06 2021].
- Google, „<https://dart.dev/>,” Google, [Interactiv]. Available:  
4] <https://dart.dev/overview>. [Accesat 04 06 2021].
- „<https://www.djangoproject.com/>,” Django, [Interactiv]. Available:  
5] <https://www.djangoproject.com/>. [Accesat 04 06 2021].
- Wikipedia, „<https://en.wikipedia.org/>,” Wikipedia, [Interactiv]. Available:  
6] [https://en.wikipedia.org/wiki/Django\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework)). [Accesat 04 06 2021].
- „<https://darly.solutions/>,” Darly Solutions, 05 06 2021. [Interactiv]. Available:  
7] <https://darly.solutions/the-most-popular-programming-languages-in-2021/>. [Accesat 05 06 2021].
- R. Zwetsloot, „[magpi.raspberrypi.org/](https://magpi.raspberrypi.org/),” Raspberry Pi Foundation, [Interactiv].  
8] Available: <https://magpi.raspberrypi.org/articles/raspberry-pi-4-specs-benchmarks>. [Accesat 18 6 2021].