  
**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ**

**Дипломна работа на тема**:

**СИСТЕМА ЗА СЪЗДАВАНЕ И ОБРАБОТКА НА ПОРЪЧКИ В РЕСТОРАНТ**

|  |  |
| --- | --- |
| **София 2025** | **Дипломант:**  **Калоян Каменов Борисов КСИ, 121324008**  **Дипломен ръководител:**  **Ас. Иво Гергов** |

# Съдържание

[I. Съдържание 4](#_Toc200053017)

[II. Списък с фигури 5](#_Toc200053018)

[III. Увод 6](#_Toc200053019)

[IV. Постановка на дипломната работа, цели и задачи. 7](#_Toc200053020)

[4.1 7](#_Toc200053021)

[4.2 Цели и задачи 7](#_Toc200053022)

[4.3 Ограничения 8](#_Toc200053023)

[4.4 Използвани технологии 8](#_Toc200053024)

[4.4.1 Python 8](#_Toc200053025)

[4.4.2 Django 8](#_Toc200053026)

[4.4.3 Django REST Framework 8](#_Toc200053027)

[4.4.4 Kotlin 9](#_Toc200053028)

[4.4.5 Android 9](#_Toc200053029)

[4.4.6 Bluetooth 9](#_Toc200053030)

[4.4.7 ESP32 9](#_Toc200053031)

[4.4.8 Docker 10](#_Toc200053032)

[4.4.9 Appliku 10](#_Toc200053033)

[4.4.10 Amazon Web Services 10](#_Toc200053034)

[V. Функционално описание 11](#_Toc200053035)

[5.1 Общ преглед на системата 11](#_Toc200053036)

[5.2 Django сървър 12](#_Toc200053037)

[5.3 Клиентско приложение 14](#_Toc200053038)

[VI. Програмна реализация. 15](#_Toc200053039)

[VII. Ръководство за използване и примери за употреба.  16](#_Toc200053040)

[VIII. Заключение 17](#_Toc200053041)

[IX. Съкращения 18](#_Toc200053042)

[X. Източници 19](#_Toc200053043)

# Списък с фигури

[Фигура 1 общ преглед на системата 11](#_Toc200053011)

[Фигура 2 Административен панел за елементи от менюто 12](#_Toc200053012)

[Фигура 3 Примерен Swagger документ 13](#_Toc200053013)

[Фигура 4 Elastic Cloud сървър в AWS 13](#_Toc200053014)

[Фигура 5 Приложение в Appliku 14](#_Toc200053015)

# Увод

Ресторантьорството е един добре познат бизнес, който има представителства във всеки град. Всеки човек е бил в поне един ресторант, където най-вероятно е открил най-големите недостатъци на бизнеса.

Ресторантите са добре установена формула. Имат кухня, където се приготвя храната, имат сервитьори, които обикалят из салона, вземат поръчките на клиентите и носят поръчаното. Най-голямото забавяне между влизането в ресторанта и започването на приготвянето на поръчката е времето, което отнема на сервитьора да стигне до масата и да вземе поръчката Ви. Това време, може да бъде много, а може да бъде и малко. Никога не можем да бъдем сигурни, тъй като това зависи от броя сервитьори в ресторанта, натовареността им, а и от човешката грешка. Много често се случва, особено при висока натовареност, сервитьор да не посрещне нов клиент, заради това, че е зает с обслужването на съществуващ клиент.

Тук идват автоматизираните системи за поръчка. Такива системи не са непознати. Напоследък започват да се забелязват QR кодове, които позволяват на клиента да зареди менюто на ресторанта на своето мобилно устройство. По този начин, клиентът може предварително да реши какво иска да поръча. Тази система ускорява времето за поръчка, но отново имаме чакането докато се освободи сервитьор, който да ни обслужи.

Съществуват друг вид такива системи, които добавят възможност за поръчка, след сканиране на QR кода. При тези системи, се прави поръчка чрез онлайн формуляр, в който автоматично се попълва номерът на масата, след като е сканиран QR кодът. Тези системи позволяват директна поръчка към кухнята. При тях, храната ни започва да се приготвя в момента, в който я изберем. Големият недостатък при този вид системи, е че зловредни потребители могат да създават фалшиви поръчки от собствения си дом. Достатъчно е потребителят да е посетил ресторанта веднъж, и вече те могат да използват същия QR код и да правят поръчки за дадена маса, без те да бъдат там.

# Постановка на дипломната работа, цели и задачи.

## 4.1 Постановка

Тази разработка ще предложи надграждане на автоматичните системи за приемане на поръчки, като основната цел ще бъде премахване на поръчките, направени от потребители, които не се намират в ресторанта. Ще бъдат използвани технологии за сканиране на маяци, които ще потвърждават дали потребителят се намира в ресторанта или някъде извън него.

## 4.2 Цели и задачи

За тази разработка са поставени няколко цели:

* Да се реши проблемът с фалшивите поръчки при автоматичните системи за поръчка
* Да се предостави удобно приложение за потребителите
* Да се показват създадените поръчки по удобен за четене начин

За да бъдат постигнати тези цели, трябва да бъде изградена система, която да бъде лесна за достъпване от потребителите и да предоставя удобен начин за създаване на поръчка към кухнята на даден ресторант. Системата трябва да бъде достъпна от мобилното устройство на потребителя, тъй като всеки потребител използва собствените си устройства с най-голяма лекота.

Освен мобилното приложение, към системата трябва да бъде изграден сървър, който да приема поръчките и да ги предоставя на кухнята по удобен за персонала начин. Също, трябва да бъдат създадени маяци, които да бъдат разположени по масите в ресторанта, така че когато потребител заеме дадена маса, поръчката да бъде създавана директно за конкретната маса, без необходимост от въвеждане на допълнителни данни от страна на потребителя.

Задачите, които можем да поставим за тази разработка са:

* Да се създаде мобилно приложение, базирано на Kotlin
* Мобилното приложение да използва BLE маяци, чрез които да определя масата на потребителя
* Да се създаде сървърно приложение, което да приема поръчките от клиентите
* Да се създаде интерфейс, който да показва на готвачите и сервитьорите поръчките и техните статуси

## 4.3 Ограничения

При разработката на дадената система трябва да се съобразим с ред ограничения, които не могат да бъдат избегнати. Тези ограничения произлизат от необходимостта за защита на личните данни на потребителите. Това означава, че мобилното приложение трябва да се съобрази с желанията на потребителите, като през това време не предоставя техните данни и също не злоупотребява с тях.

При разработката на мобилното приложение, трябва да се положи особено внимание върху сигурната комуникация между приложението и сървъра. При незащитена комуникация е възможно личните данни на потребителя да бъдат засечени от недоброжелатели и използвани по неоторизиран начин.

## 4.4 Използвани технологии

### 4.4.1 Python

Сървърът, който ще обслужва системата ще бъде написан на Python. Този език предоставя начин за бързо прототипизиране на система, благодарение на големия набор от библиотеки, които поддържа, както и на голямото количество програмисти, които го използват и са активни във форуми за взаимопомощ.

### 4.4.2 Django

Django е библиотека за изграждане на сървъри, които да обслужват данни. Предоставя голям набор от инструменти, които могат да бъдат използвани за лесно създаване на сървър. Основната цел на библиотеката е да предостави среда, която да премахне голямата част от често срещаните проблеми при разработка на сървърни приложения и да улесни и ускори работата на програмистите [1].

Библиотеката е с отворен код, като програмистите са окуражавани да се включват в нейната разработка, като целта е да бъде направена колкото се може по-достъпна. Тя предоставя лесен начин за начинаещи програмисти да пишат уеб приложения, а опитните програмисти могат да се възползват от широкият набор от функционалности, които библиотеката предлага.

### 4.4.3 Django REST Framework

Django REST Framework или DRF представлява библиотека, която разширява способностите на Django, като добавя възможност за създаване на RESTful API. Библиотеката предоставя удобен начин за създаване на API, което да поддържа всички заявки, като GET, POST, PUT, DELETE и др. Библиотеката също предоставя удобен начин за сериализация на обекти, които да могат да бъдат изпращани между клиента и сървъра [2].

### 4.4.4 Kotlin

Kotlin представлява език за програмиране, който може да бъде използван за създаване на приложения за разнообразни платформи. Езикът е направен да работи, използвайки Java и така постига тази платформена независимост. Стандартната библиотека за Kotlin стои директно върху Java библиотеки.

Езикът е широко разпространен при разработка на мобилни приложения, тъй като надгражда Java, която е била най-разпространена, преди Kotlin да се появи. Kotlin предоставя ясен синтаксис и надгражда Java с разширена сигурност [3].

### 4.4.5 Android

Android е най-разпространената операционна система за мобилни устройства. Разработва се от Google, като е базирана върху Linux. В основата си Android е операционна система с отворен код, разработвана от Open Handset Alliance, но популярнта версия на операционната система, която се изпълнява върху почти всички устройства е тази, разработена от Google, която обаче не е с отворен код [4].

### 4.4.6 Bluetooth

Bluetooth представлява технология, която се използва за трансфер на данни между устройства на кратки разстояния. Технологията се е установила като стандарт при комуникацията между устройства като мобилен телефон и слушалки или компютри и мишки или клавиатури. Технологията е доста стара, но нейното развитие не спира. В днешно време, устройствата, които използват Bluetooth, вече могат да комуникират с доста висока скорост и да не използват прекалено много от батерията си. Разбира се, при приложения, които изискват бързо предаване на големи обеми данни безжично, Wi-Fi остава предпочитаният избор [5].

Съществува вариант на Bluetooth, който използва минимално количество енергия, наречен Bluetooth Low Energy или BLE. Тази технология работи много подобно на нормалния Bluetooth, с основната разлика, че при нея устройствата са в спящ режим през повечето време. Те се събуждат само ако някой се опита да инициира връзка с тях [6]. Това ги прави много полезни като маяци, които да обявяват съществуването си и близостта си до устройството, което се опитва да се свърже с тях.

### 4.4.7 ESP32

ESP32 са серия от микроконтролери, разработени от Espressif, които придобиват популярност благодарение на своята ниска цена и висока енергиина ефективност. Тези микроконтролери са използвани за най-различни любителски проекти, тъй като предоставят голямо количество методи за безжична комуникация, която позволява да направим умни устройства вкъщи [7]. Заради тази популярност, тези микроконтролери са добре документирани и много от случаите на употреба, които можем да имаме, са вече разработени и достъпни за използване.

### 4.4.8 Docker

Docker представлява платформа, използвана за разработка и изпълнение на приложения. Платформата ни предоставя начин да разкачим приложението от инфраструктурата, така че да можем да предоставим софтуер бързо, без много конфигуриране. Docker предоставя контейнери, които представляват олекотени среди, в които да се изпълнява приложението ни. Тези среди могат да бъдат споделяни лесно, което улеснява процеса на работа [8].

### 4.4.9 Appliku

Appliku е платформа, която позволява лесно менажиране на сървъри. Платформата предоставя обединено място, където да бъдат менажирани бази данни, приложения, CRON процеси и други. Важно е да се отбележи, че Appliku само по себе си не предлага сървъри. Платформата е BYO Server. Това означава, че трябва да имаме съществуващ сървър, или място, където Appliku да може да създаде сървър. Appliku има директна интеграция с AWS, което прави създаването на сървър лесно [9].

### 4.4.10 Amazon Web Services

Amazon Web Services или по-разпространеното AWS, представлява платформа, където можем да създадем разнообразни услуги. AWS предоставя сървъри, бази данни, среди за работа с изкуствен интелект и машинно обучение и още много други [10].

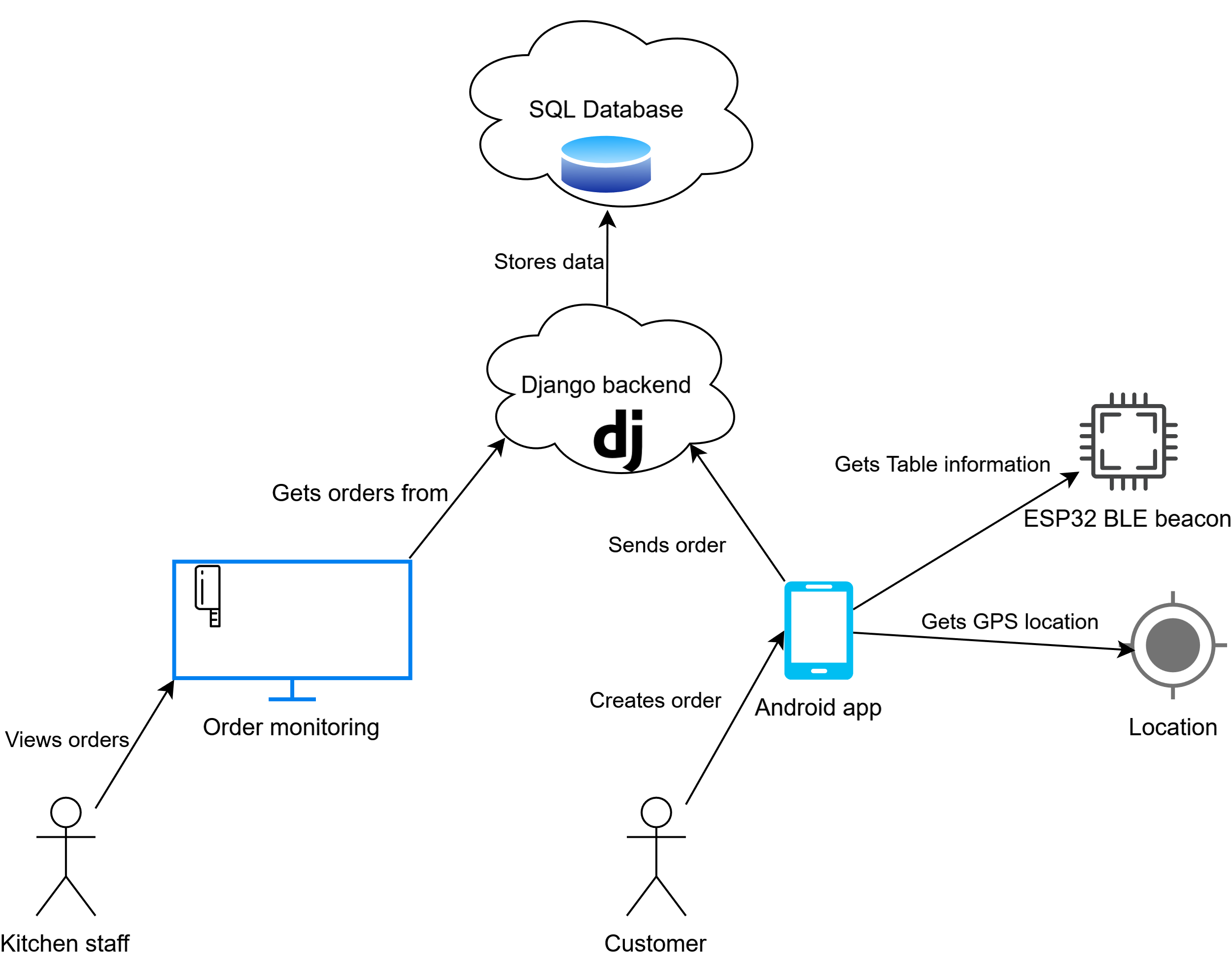
# Функционално описание

## 5.1 Общ преглед на системата

Системата се състои от четири основни компонента, като три от тях са разположени на AWS сървър, а последното се изпълнява на клиентското устройство. Компонентите са: Android приложение, което единствено се изпълнява на мобилно устройство, база данни, която се използва за съхранение на данни, Django сървър, който предоставя Backend за комуникация с базата данни и Frontend, който се използва за визуализация на поръчките.

Комуникацията между различните среди се случва посредством HTTPS заявки. Backend услугата, която обработва всички заявки, използва няколко процеса, които да обработват конкурентно голямо количество заявки. По този начин, ние можем да обслужваме голямо количество клиенти едновременно.

На Фигура 1 е показан общият изглед на системата. Приложението на клиентското устройство използва откритото API на Django сървъра, за да прави HTTPS заявки. Django сървърът обработва приетите заявки и създава записи в базата данни с новополучената поръчка. Кухненският персонал получава новите поръчки, чрез Frontend, който отново е поддържан от Django.

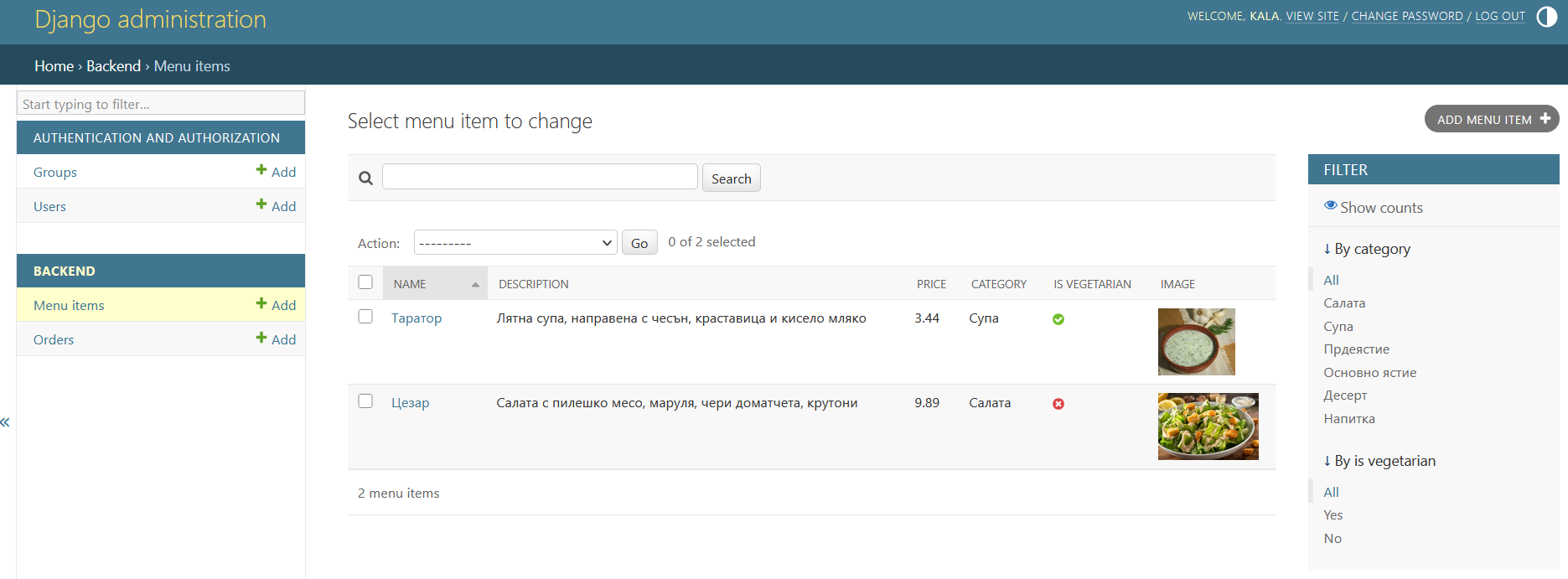


Фигура 1 общ преглед на системата

## 5.2 Django сървър

Django сървърът играе ролята и на Backend и на Frontend. Реализацията ще бъде извършена с тази библиотека, тъй като тя е де-факто стандарт при разработка на подобни сървъри, когато се използва Python.

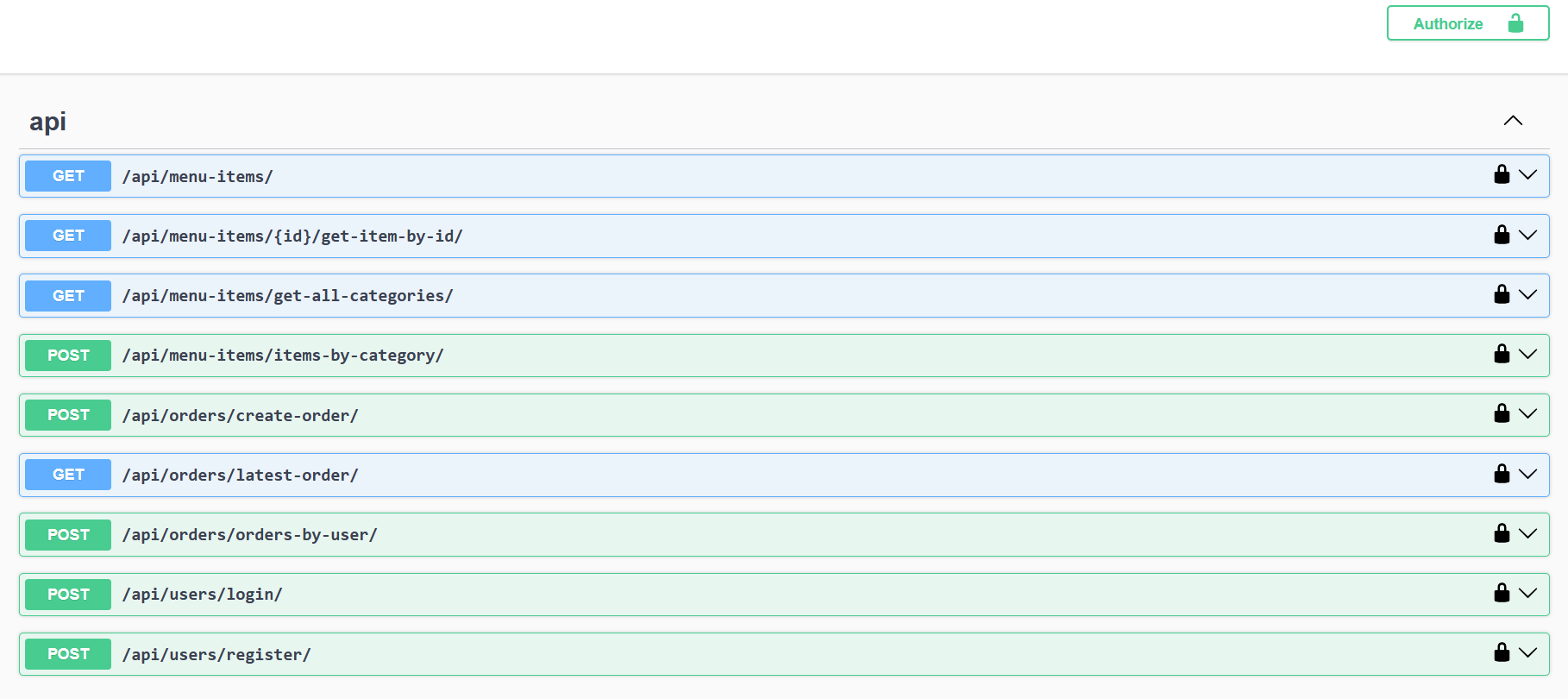
Библиотеката предоставя голямо количество готови функционалности, които могат да бъдат използвани директно, като например управление на потребители. Конфигурацията се случва много лесно, тъй като Django предоставя файл с настройки, в който потребителят може да управлява своят сървър. Освен този файл, разполагаме и с уеб страница, която ни позволява лесно управление на много от приложението ни. Тази страница, наречена административен панел, е стандартна част от всяко Django приложение, която може да бъде конфигурирана с предоставените инструменти, директно от кода. Можем да показваме и управляваме най-различни функционалности на нашето приложение, стига да ги добавим в дадения панел. На Фигура 2 е показана страница от този панел.



Фигура 2 Административен панел за елементи от менюто

Django обществото предоставя голямо количество допълващи библиотеки, които могат лесно да бъдат добавени в съществуващ проект. Една такава допълваща библиотеки е Django REST Framework, която е използвана в тази разработка.

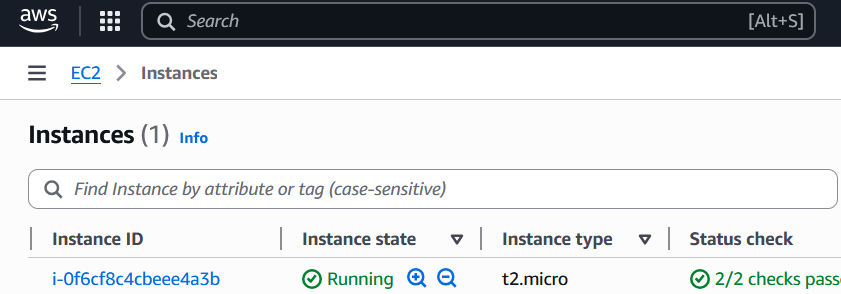
Django REST Framework ни кара да пишем RESTful приложения, за разлика от стандартната Django библиотека, която позволява доста либерално API. Като използваме DRF, ние можем да генерираме Swagger клиенти, които да улеснят клиентски приложения, които използват нашия сървър. На Фигура 3 е показан примерен Swagger документ.



Фигура 3 Примерен Swagger документ

Django сървърът ще се изпълнява в облачното пространство, за да бъде достъпен навсякъде. За добавена сигурност, сървърът използва HTTPS протокол за комуникация, като освен това достъпът е защитен чрез система за управление на потребителите.

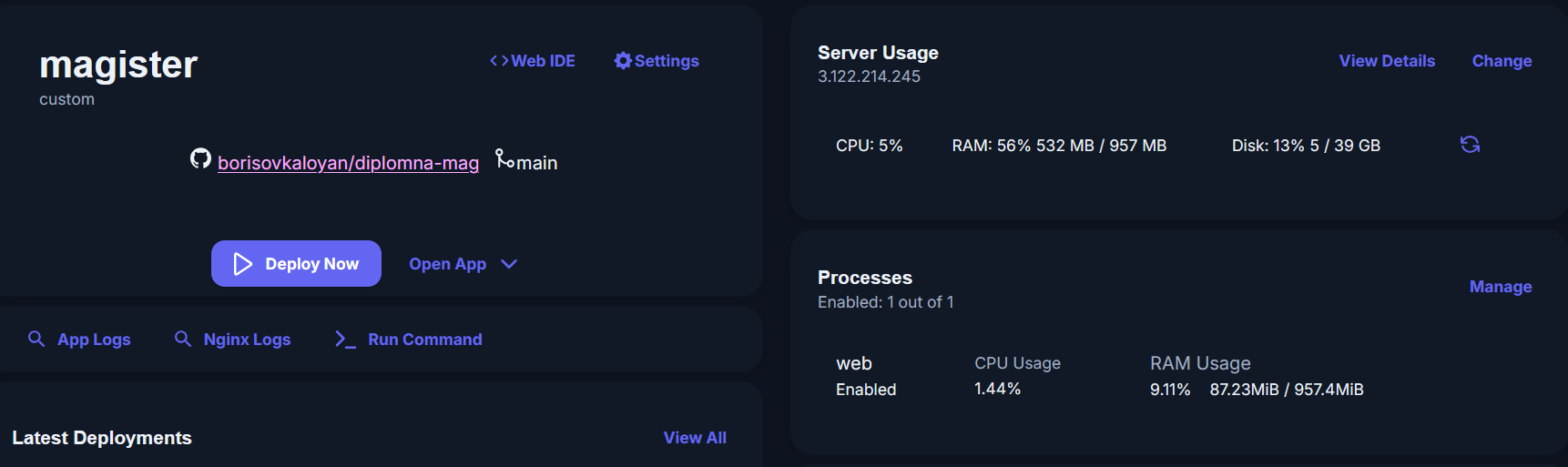
За хостинг платформа ще се използва Amazon Web Services. Тази платформа е една от най-използваните в световен мащаб. Има 24 часова поддръжка и ни гарантира, че нашият сървър ще бъде постоянно достъпен. На Фигура 4 е показан сървърът, върху който ще се изпълнява нашият Django проект.



Фигура 4 Elastic Cloud сървър в AWS

За да се изпълнява приложението върху този сървър, ще се използва Docker контейнер. По този начин, постигаме независимост от средата, в която се изпълнява приложението ни. За по-лесна настройка на контейнера и средата, използваме Appliku. Това е платформа, която е вид обвивка над AWS. Сама по себе си, тя не изпълнява приложение и не предлага сървър. Нейната основна функция е да позволява лесна конфигурация на сървъри и приложения, като използва вече съществуващи платформи (като AWS) за да предоставя услуги.

На Фигура 5 е показано как изглежда примерно приложение в Appliku. Можем ясно да видим удобната интеграция със сървъра в AWS, както и с файловете в GitHub. Appliku също постоянно следи GitHub за промени и автоматично актуализира приложението ни, когато се качат промени. По този начин, постигаме постоянна актуалност на приложението.



Фигура 5 Приложение в Appliku

Базата данни също ще се менажира в Appliku. По същия начин, сървърът работи в AWS, като управлението е изведено в Appliku. По този начин имаме централизирано място за удобно управление на всички ресурси на сървърите ни.

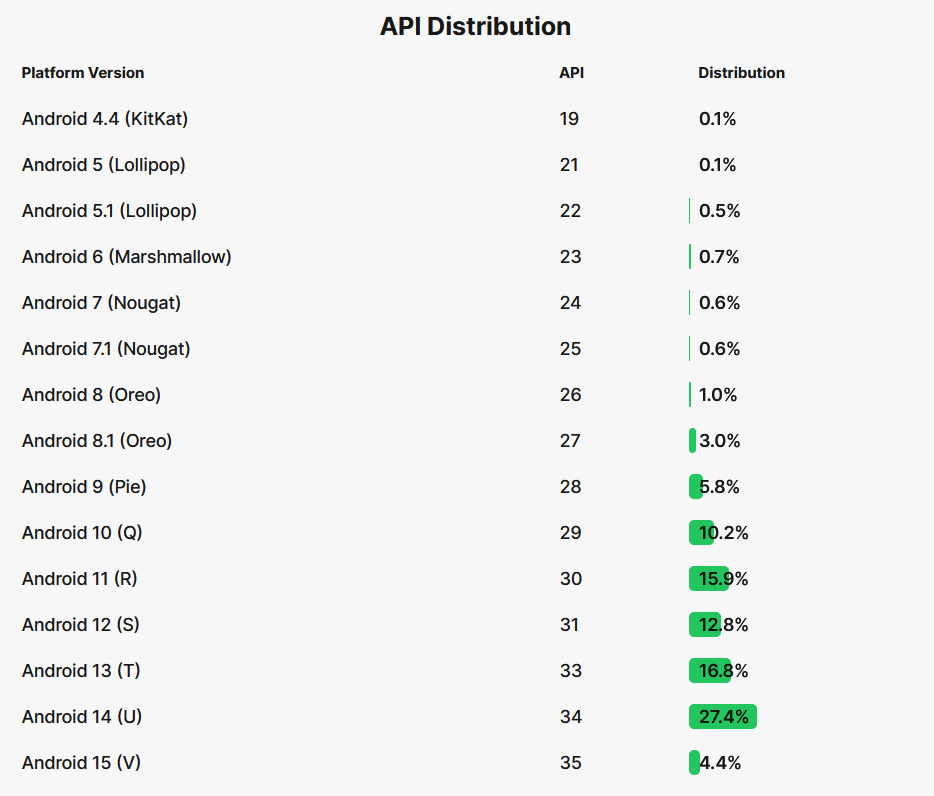
Освен API, Django приложението ще предоставя уеб страница, която да показва актуална информация за всички поръчки, направени за конкретен ден. Тези поръчки ще бъдат групирани по техния статус, което ще улесни изпълнението им в кухнята и последващото им сервиране. Приложението ще пази история на поръчките, което ще помогне за проследяване на активността в ресторанта през времето.

## 5.3 Клиентско приложение

Клиентското приложение ще представлява мобилно приложение, написано на Kotlin. Това е препоръчаният от Google език за писане на приложения. Фокусът ще бъде основно върху Android, тъй като това е най-разпространената операционна система. Съществуват голямо количество ресурси, които да ни улеснят при писането на приложението, благодарение на популярността на платформата и езика за програмиране.

Приложението ще бъде съвместимо с Android устройства с API версия по-висока от 26. Това отговаря на версията 8.0 Oreo. Версията е изкарана на пазара през 2017 година и вече е доста остаряла. Последната версия на Android към Май 2025 е 15. Нашето приложение позволява на потребителите да използват устройства, с операционна система 8 версии по-ниска от текущата. По този начин, ние се подсигуряваме, че мнозинството от устройствата, които се използват днес, ще могат да изпълняват нашето приложение.

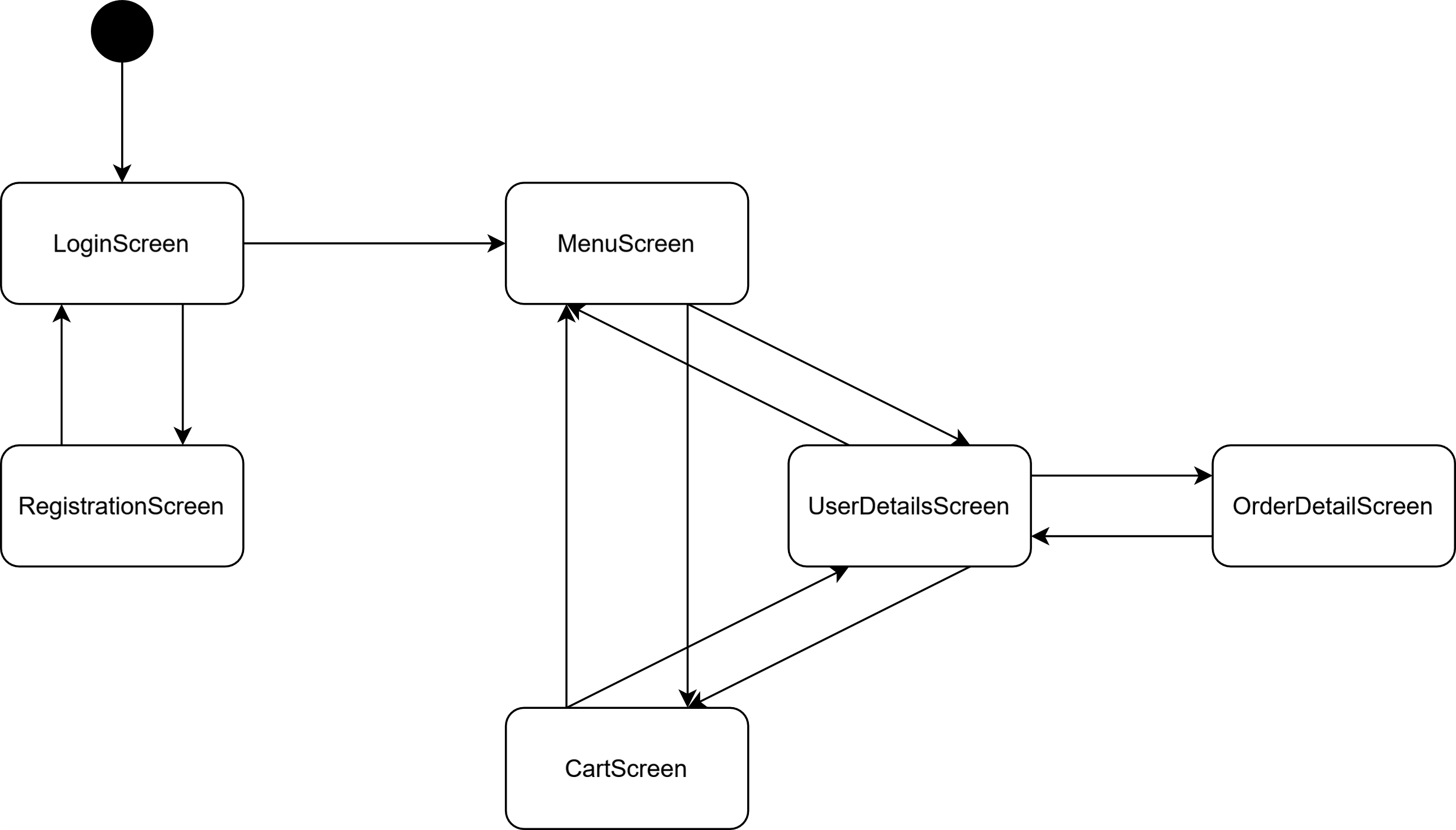
На Фигура 6 е показано разпределението на активните устройства, спрямо версията на операционната им система [11]. Според тази графика, ако изберем версия 26 на API за Android, то около 97% от устройствата на пазара биха могли да изпълняват нашето приложение.



Фигура 6 Разпределение на мобилни устройства според версията им на Android

Мобилното приложение ще се състои от няколко екрана, които ще изпълняват различни функции. Тези екрани, ще бъдат: Екран за вход, Екран за регистрация, Екран меню, Екран потребителска информация, Екран кошница и Екран информация за поръчка. Приложението ще предоставя лесна навигация между екраните, посредством навигационна лента.

На Фигура 7 е показано какви трябва да бъдат екраните в приложението и връзките между тях. Виждаме, че екранът за вход е началната точка на приложението. От там, потребителят може да навигира към екран за регистрация, за да създаде своя профил или, при вече съществуващ профил, той може да навигира към менюто. От менюто, потребителят вече има достъп до навигационна лента, която му позволява да навигира между менюто, своя профил и кошницата само с един клик. Екранът за потребителска информация има връзка с екрана за детайли за дадена поръчка.



Фигура 7 Структура на екраните на приложението

Екранът за вход е най-простият от всички. Той ще служи за входна точка в приложението, като освен функционалността за вход в системата, той също ще има връзка с екрана за регистрация на потребител. Ако потребител няма съществуващ профил, той следва да навигира към екрана за регистрация. След създаване на профил, потребителят може да използва екрана за вход за да навигира навътре в приложението, към екрана с менюто.

Екранът с менюто трябва да позволява на потребителя да разглежда артикулите, предлагани от ресторанта, групирани по категории за по-лесна визуализация. От този екран, когато потребителят избере ястие, трябва да може да бъдат добавяни артикули към кошницата. На този екран, потребителят за първи път вижда навигационната лента, която представлява главния начин за придвижване между различните страници на приложението. Лентата позволява лесно движение между страниците за меню, кошница и информация за потребителя.

След като потребителят е добавил гозби в своята кошница, той може да прегледа съдържанието ѝ от екрана за кошница, достъпен от навигационната лента. На този екран, потребителят може да види артикулите в кошницата, групирани по тип с показан брой. Потребителят ще може да увеличава или намалява броя от даден артикул от този екран. Екранът също ще показва текущата стойност на сметката и ще позволява да бъде направена поръчка.

След като е направена поръчка, потребителят може да я следи, като навигира към страницата с детайли за потребителя. Там могат да бъдат прегледани основни данни за всички поръчки, създадени от дадения потребител, като всяка поръчка може да бъде натисната за да бъде разгледана по-детайлно.

При натискане на някоя поръчка, приложението ще зареди страницата за детайли за поръчка, като ще я популира с конкретните детайли за избраната поръчка. На тази страница навигационната лента вече не е видима, тъй като искаме единственият път назад да бъде обратно към страницата с информация за потребителя.

## 5.4 Уеб приложение

Уеб приложението ще представлява поредица от HTML страници, които ще дават информация за създадени поръчки от всички потребители. Поръчките ще бъдат показвани само за конкретния ден, като приложението ще предоставя възможност за преглеждане на поръчки от предходни дни.

Уеб приложението трябва да бъде достъпно само за администратори и работници на ресторанта. Поради тази причина, за да бъде достъпено то, е създаден механизъм за вход. Ако потребител се опита да достъпи приложението, без да е влизал преди това, той ще бъде пренасочен към страницата за вход в приложението. Там той трябва да въведе своите данни. Ако потребителят е определен като персонал, той ще бъде допуснат до таблото за управление. Ако не е, той няма да може да достъпи таблото.

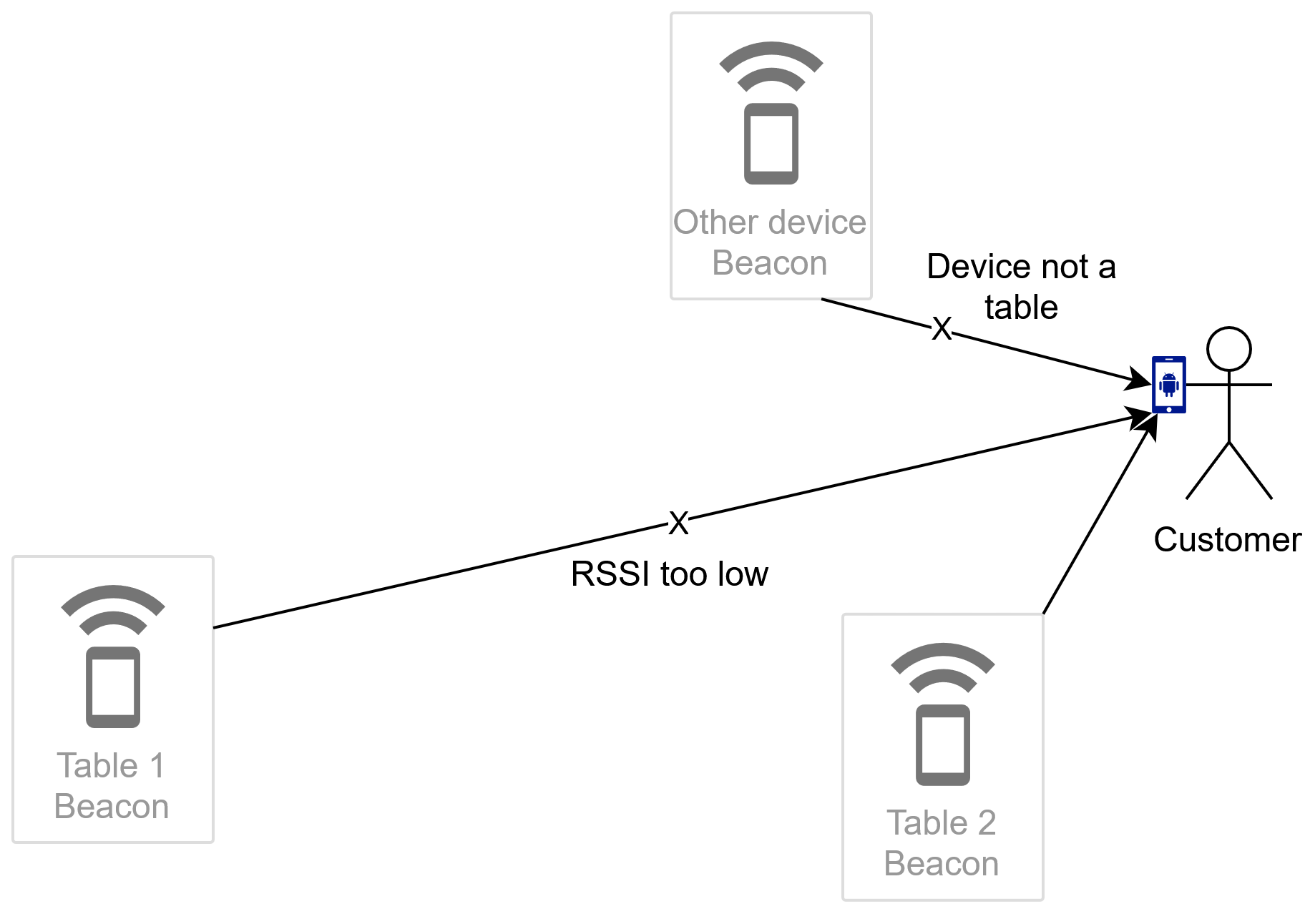
Таблото за управление представлява главният екран на уеб приложението. Там ще могат да бъдат разглеждани поръчките за конкретен ден, като ще бъдат групирани според своя статус, за лесно проследяване на процеса по приготвяне и доставяне. По този начин, процесите в кухнята ще бъдат улеснени и приготвянето на храната ще се случва по-ефективно.

## 5.5 Маяци

Маяците ще бъдат метода за идентификация на клиенти, които са решили да посетят ресторанта и да поръчат през приложението. Те ще представляват малки платки, поставени върху масите и замаскирани като декорация. Всеки маяк ще използва BLE за да обявява себе си и номера на масата, на която е поставен.

Когато клиент посети ресторанта и опита да направи поръчка през приложението, то ще опита да разпознае маяк около себе си. Ако е успешно, то ще трябва да определи дали този маяк е маяк на маса или е друго устройство. Ако маякът е разпознат като маяк за маса, той ще бъде сравнен с околните маяци за маси и ще бъде определена най-близката маса по RSSI на сигнала. След като е определен най-близкия маяк, ще бъде направена поръчка, с индикатор за масата.

На Фигура 8 е показана интеракцията на клиентско устройство с близки маяци. При тази ситуация, приложението трябва да идентифицира, че потребителя седи на втора маса.



Фигура 8 Идентификация на маяк

# Програмна реализация.

# Ръководство за използване и примери за употреба.

# Заключение

# Съкращения

8

API - Application Programming Interface 8

AWS - Amazon Web Services 10

BLE - Bluetooth Low Energy 9

BYO - Bring Your Own 10

DRF - Django REST Framework 8

REST - Representational State Transfer 8

# Източници

|  |  |
| --- | --- |
| [1] | Django, „Meet Django,“ [Онлайн]. Available: https://www.djangoproject.com/. |
| [2] | A. H. Maki, „Create REST API using Django REST Framework,“ [Онлайн]. Available: https://medium.com/@ahmalopers703/getting-started-with-django-rest-api-for-beginners-9c121a2ce0d3. |
| [3] | J. Dwivedi, „Mastering Kotlin for Android: A Deep Dive Learning Guide,“ [Онлайн]. Available: https://medium.com/@jaidwivedi20/mastering-kotlin-for-android-a-deep-dive-learning-guide-5db852c277e6. |
| [4] | E. Mixon, „Android OS,“ [Онлайн]. Available: https://www.techtarget.com/searchmobilecomputing/definition/Android-OS. |
| [5] | B. Collins, „What is Bluetooth,“ [Онлайн]. Available: https://www.forbes.com/sites/technology/article/what-is-bluetooth/. |
| [6] | B. Proctor, „Bluetooth vs Bluetooth Low Energy,“ [Онлайн]. Available: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy. |
| [7] | Espressif, „ESP32 SoC,“ [Онлайн]. Available: https://www.espressif.com/en/products/socs/esp32. |
| [8] | Docker, „Docker overview,“ [Онлайн]. Available: https://docs.docker.com/get-started/docker-overview/. |
| [9] | Appliku, „What is Appliku,“ [Онлайн]. Available: https://appliku.com/. |
| [10] | Amazon, „Amazon Web Services,“ [Онлайн]. Available: https://aws.amazon.com/. |