



**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.	Увод .....	7
II.	Постановка на дипломната работа, цели и задачи. ....	8
2.1	Постановка.....	8
2.2	Цели и задачи .....	8
2.3	Ограничения .....	8
2.4	Използвани технологии.....	9
2.4.1	Python.....	9
2.4.2	Django .....	9
2.4.3	Django REST Framework.....	9
2.4.4	Kotlin.....	9
2.4.5	Android.....	10
2.4.6	Bluetooth .....	10
2.4.7	ESP32.....	10
2.4.8	Docker .....	11
2.4.9	Appliku.....	11
2.4.10	Amazon Web Services .....	11
III.	Функционално описание.....	12
3.1	Общ преглед на системата .....	12
3.2	Django сървър.....	13
3.3	Клиентско приложение .....	15
3.4	Уеб приложение .....	18
3.5	Маяци .....	18
IV.	Програмна реализация. ....	20
4.1	Реализация на Django сървър.....	20
4.1.1	Модели .....	20
4.1.2	Входни точки с OpenAPI спецификация.....	22
4.1.3	Изгледи на приложението .....	30
4.1.4	Реализация на Django Уеб Страница.....	37
4.2	Реализация на маяк .....	39
4.3	Реализация на мобилно приложение .....	39
4.3.1	Екран за вход.....	41

4.3.2 Екран за регистрация .....	41
4.3.3 Екран за преглед на меню .....	42
4.3.4 Екран за кошница .....	43
4.3.5 Екран за потребител .....	45
4.3.6 Екран за преглед на детайли на поръчка .....	46
V. Ръководство за използване и примери за употреба. ....	48
5.1 Използване на системата от клиент .....	48
5.1.1 Създаване на регистрация .....	48
5.1.2 Влизане в системата .....	48
5.1.3 Разглеждане на менюто и добавяне в кошницата .....	48
5.1.4 Създаване на поръчка .....	49
5.1.5 Проследяване на поръчка .....	49
5.2 Използване на системата от персонал .....	49
VI. Заключение .....	51
Съкращения .....	52
Източници .....	53
Приложения .....	55

## Списък с фигури

Фигура 1	Общ преглед на системата .....	12
Фигура 2	Административен панел за елементи от менюто.....	13
Фигура 3	Примерен Swagger документ .....	14
Фигура 4	Elastic Cloud сървър в AWS .....	14
Фигура 5	Приложение в Appliku .....	15
Фигура 6	Разпределение на мобилни устройства според версията им на Android .....	16
Фигура 7	Структура на екраните на приложението .....	17
Фигура 8	Идентификация на маяк .....	19
Фигура 9	Клас диаграма на моделите.....	21
Фигура 10	Entity Relations диаграма на базата данни .....	22
Фигура 11	Входна точка за получаване на всички елементи от менюто .....	23
Фигура 12	Входна точка за получаване на елемент по неговото ID .....	24
Фигура 13	Входна точка за вземане на категориите на елементите от менюто .....	24
Фигура 14	Входна точка за получаване на всички елементи от дадена категория.....	25
Фигура 15	Входна точка за създаване на поръчка .....	26
Фигура 16	Входна точка за получаване на всички поръчки за даден потребител .....	27
Фигура 17	Входна точка за подаване на входни данни за потребител .....	28
Фигура 18	Входна точка за създаване на нов потребител .....	28
Фигура 19	Входна точка за получаване на спецификация .....	29
Фигура 20	Поток на работа за метода get_all_categories .....	30
Фигура 21	Поток на работа на метода items_by_category .....	31
Фигура 22	Поток на работа на метода get_all_items .....	32
Фигура 23	Поток на метода get_item_by_id .....	33
Фигура 24	Поток на работа на метода orders_by_user .....	34
Фигура 25	Поток на изпълнение на метода create_order .....	35
Фигура 26	Поток на метода register .....	36
Фигура 27	Поток на метода login .....	37
Фигура 28	Страница за вход в системата .....	38
Фигура 29	Страница с информация за поръчки .....	39
Фигура 30	Екран за вход в приложението .....	41
Фигура 31	Екран за регистрация в системата .....	42
Фигура 32	Екран за преглед на меню .....	43
Фигура 33	Екран за кошница.....	44
Фигура 34	Екран за завършване на поръчката при засечен маяк .....	45
Фигура 35	Екран за поръчките на потребител.....	46
Фигура 36	Екран за информация за поръчка .....	47

## I. УВОД

Ресторантьорството е един добре познат бизнес, който има представители 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 автоматизирани 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.

## **II. ПОСТАНОВКА НА ДИПЛОМНАТА РАБОТА, ЦЕЛИ И ЗАДАЧИ.**

### **2.1 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.

### **2.2 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

### **2.3 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.

## **2.4 Използвани технологии**

### **2.4.1 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.

### **2.4.2 Django**

Django е библиотека за изграждане на сървърни приложения. Библиотеката предоставя голям набор от инструменти, които могат да бъдат използвани за лесно създаване на приложение. 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. 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 широкия набор от функционалности, които библиотеката предлага.

### **2.4.3 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 видове HTTP 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].

### **2.4.4 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].

### **2.4.5 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 популярната версия на операционната система, която се изпълнява върху почти всички устройства, е тази, разработена от Google, която, обаче, не е с отворен код [4].

### **2.4.6 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 само когато устройство се опитва да се свърже с тях.

### **2.4.7 ESP32**

ESP32 са серия от микроконтролери, разработени от Espressif, които придобиват популярност благодарение на своята ниска цена и висока енергийна 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.

### **2.4.8 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].

### **2.4.9 Appliku**

Appliku е платформа, която позволява лесно менажиране на сървъри. Платформата предоставя обединено място, където да бъдат управлявани бази данни, приложения, CRON процеси и други. Важно е да се отбележи, че Appliku само по себе си не предлага сървъри. Платформата е BYO (Bring Your Own) 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].

### **2.4.10 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].

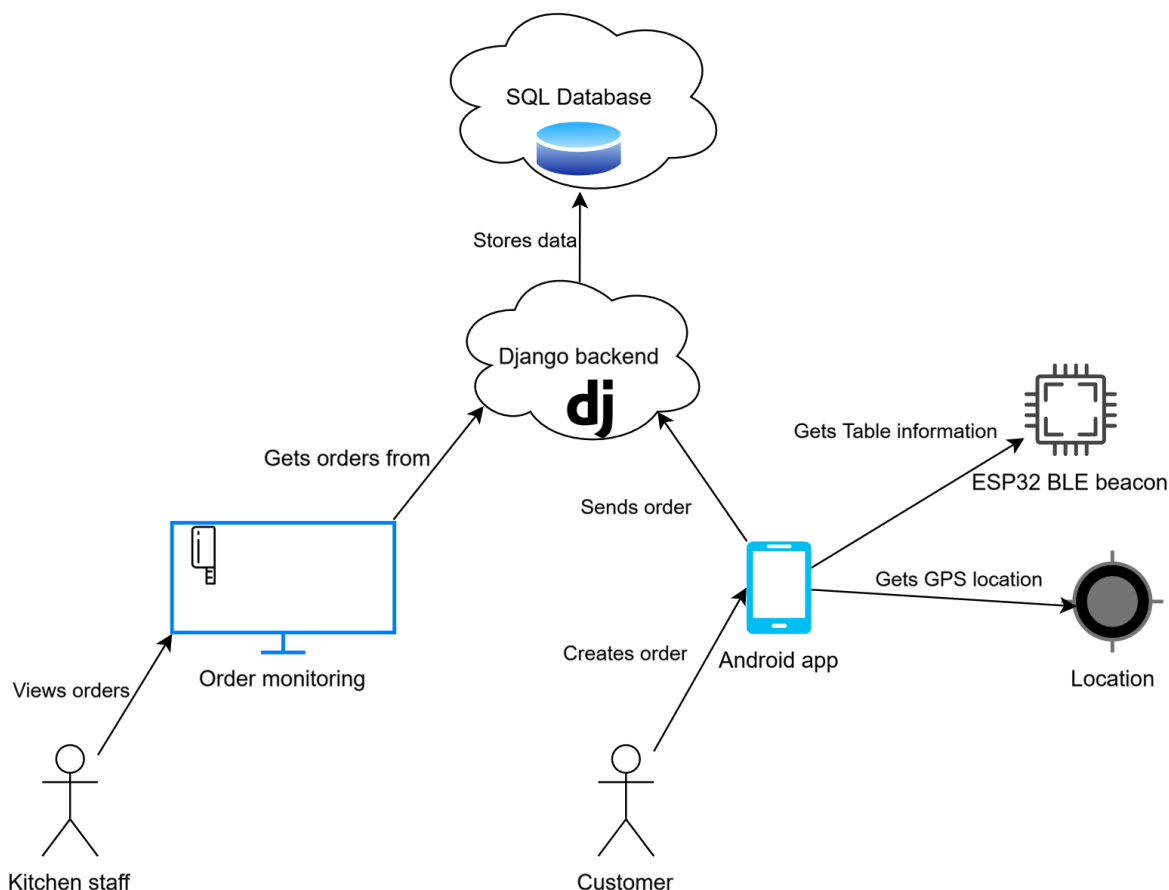
### III. ФУНКЦИОНАЛНО ОПИСАНИЕ

#### 3.1 Общ преглед на системата

Системата се състои от пет основни компонента, като три от тях са разположени на AWS сървър, един се изпълнява на клиентското устройство и един работи върху микроконтролер. 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, Frontend, който се използва за визуализация на поръчките и приложение, което създава BLE маяк.

Комуникацията между мобилното приложение и облачната среда се случва посредством HTTPS заявки. Backend услугата, която обработва всички заявки, използва няколко процеса, които да обработват голямо количество заявки конкурентно. 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 Backend 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.

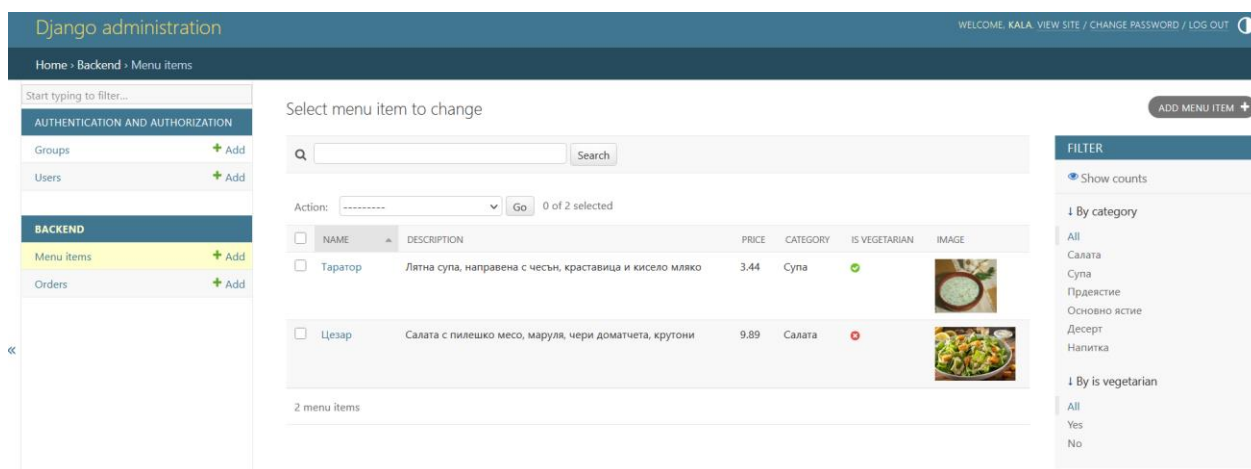


ФИГУРА 1 ОБЩ ПРЕГЛЕД НА СИСТЕМАТА

### 3.2 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 програмистът може да управлява своя 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 административния 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 библиотека е Django REST Framework, която е използвана в тази разработка.

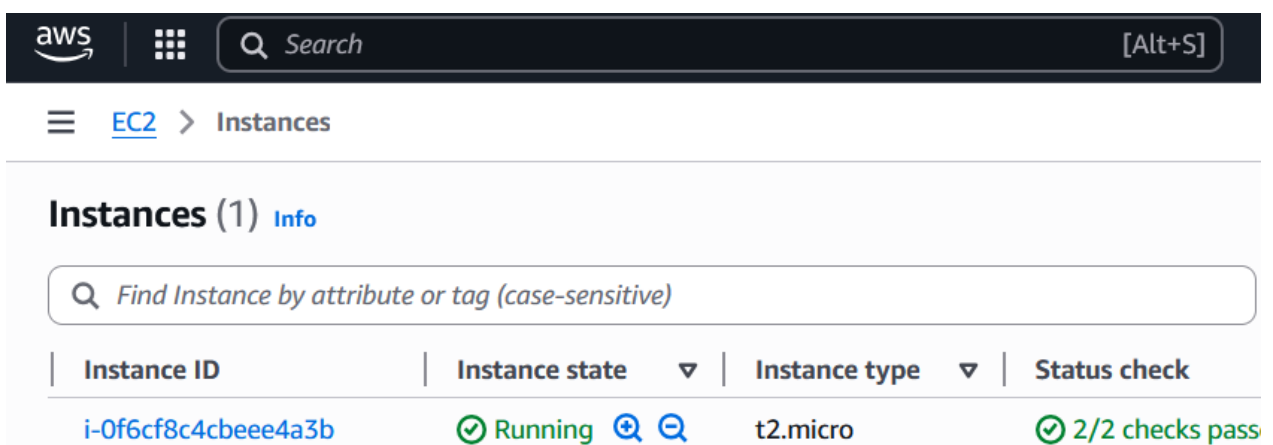
Django REST Framework ни кара да пишем RESTful приложения, за разлика от стандартната Django библиотека, която позволява доста либерално API. Като използваме DRF, ние можем да генерираме Swagger клиенти, които да улеснят клиентски приложения, които използват нашия сървър. На Фигура 3 е показан примерен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

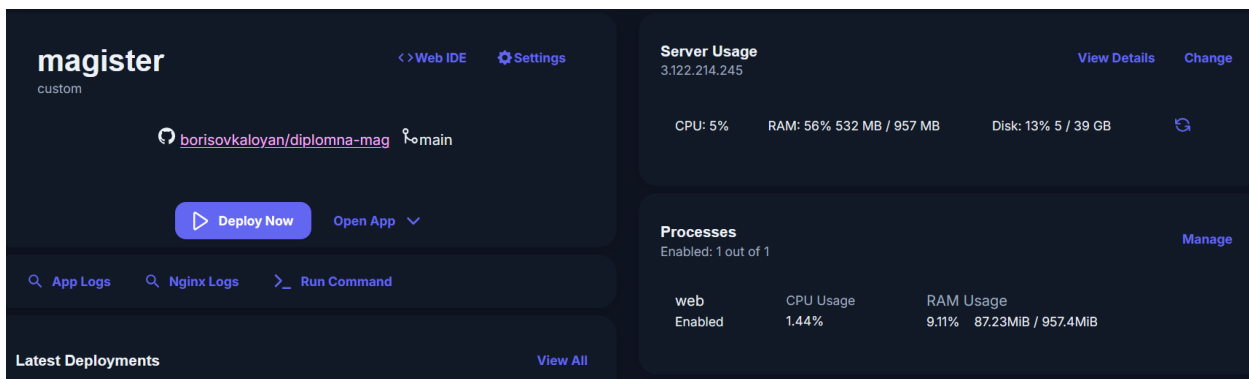


ФИГУРА 4 ELASTIC CLOUD СЪРВЪР В AWS

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.

### 3.3 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 7 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.

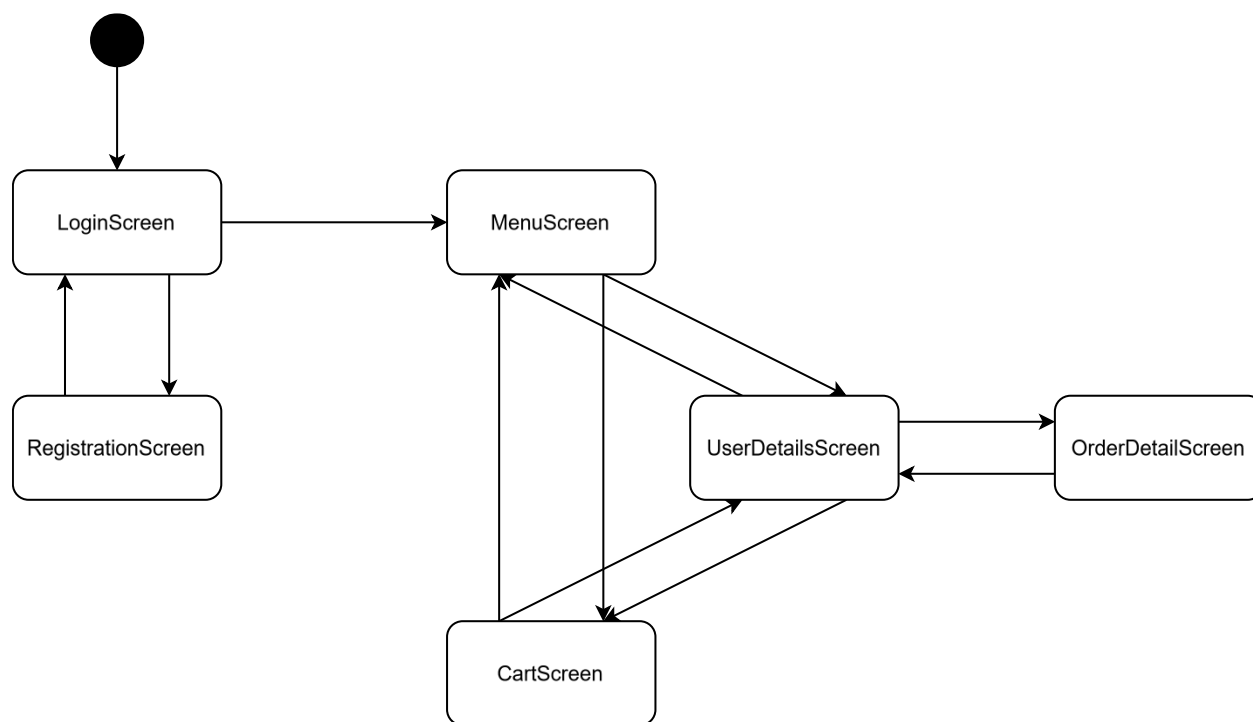
API Distribution		
Platform Version	API	Distribution
Android 4.4 (KitKat)	19	0.1%
Android 5 (Lollipop)	21	0.1%
Android 5.1 (Lollipop)	22	0.5%
Android 6 (Marshmallow)	23	0.7%
Android 7 (Nougat)	24	0.6%
Android 7.1 (Nougat)	25	0.6%
Android 8 (Oreo)	26	1.0%
Android 8.1 (Oreo)	27	3.0%
Android 9 (Pie)	28	5.8%
Android 10 (Q)	29	10.2%
Android 11 (R)	30	15.9%
Android 12 (S)	31	12.8%
Android 13 (T)	33	16.8%
Android 14 (U)	34	27.4%
Android 15 (V)	35	4.4%

**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 ще може да я следи, като навигира към страницата с детайли за потребителя. Там ще 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 попълни 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.

### **3.4 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 достъпно 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.

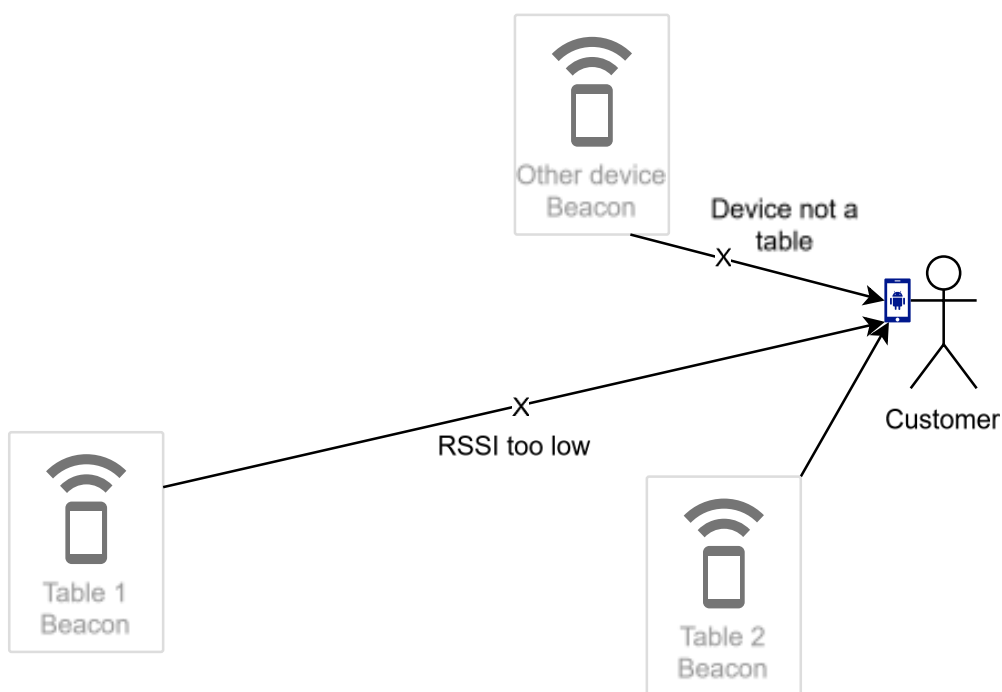
Таблото за управление представлява главният екран на уеб приложението. Там ще могат да бъдат разглеждани поръчките за конкретен ден, като ще бъдат групирани според своя статус, за лесно проследяване на процеса по приготвяне и доставяне. По този начин, процесите в кухнята ще бъдат улеснени и приготвянето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

### **3.5 Маяци**

Маяците ще бъдат методът 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**

## IV. ПРОГРАМНА РЕАЛИЗАЦИЯ.

### 4.1 Реализация на Django сървър

Django сървърът е реализиран, следвайки документираните процеси [12]. Сървърът има стандартните за Django модели и изгледи, като изгледите са надградени чрез Django REST Framework, за да можем да използваме генерирани RESTful клиенти за да достъпваме сървъра по-лесно. Освен това са създадени сериализиращи класове, които да конвертират Python обектите към JSON, който да може да бъде изпращан и приеман.

Имплементацията може да бъде разбита на няколко основни части:

- Модели, които позволяват съхранение на данните
- Входни точки, документираните чрез OpenAPI
- Изгледи, които съдържат основната функционалност
- Frontend приложение за визуализация

#### 4.1.1 Модели

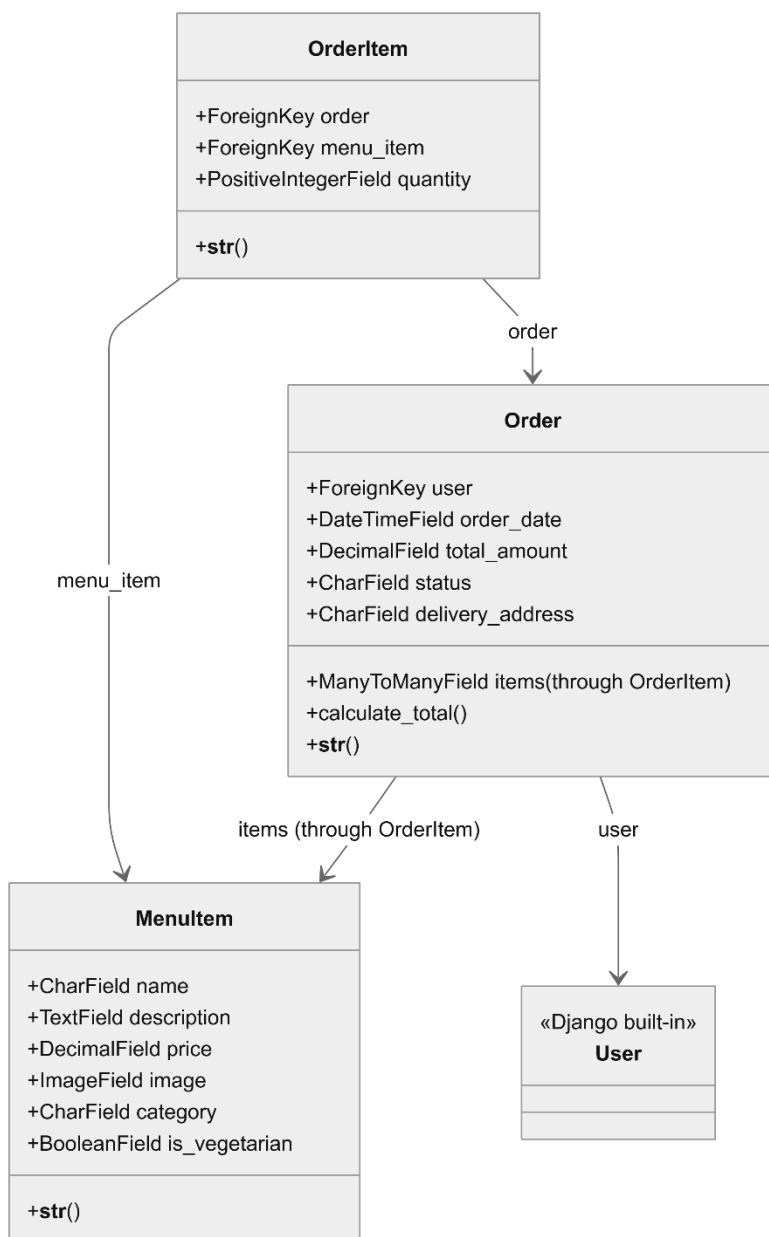
Моделите в Django представляват класове, които описват таблици от базата данни. Всеки модел трябва да наследява `Model` класа на Django, така че да може да използва предоставените от Django функции за работа с база данни.

Всеки клас, наследник на `Model`, трябва да съдържа полета, инстанции на класа `Field`. Класът `Field` предоставя основни функции, които са необходими за да бъде дефинирана колоната в базата данни. Класът е базов и не е препоръчително да се използва, освен ако не искаме да изградим наше собствено поле. Django предоставя голям набор от наследници на този клас, които можем да използваме за да дефинираме колони от различен тип. Примери за полета са: `DateField`, `CharField`, `IntegerField` и други подобни. Тези полета ни позволяват да създаваме колони за основните типове данни. Освен колони за данни, Django ни позволява да правим и колони за релации, като `ForeignKey` и `ManyToManyField`. Тези полета позволяват използването на релационни бази данни и изграждане на връзки между таблиците.

Системата с модели на Django позволява да работим с таблици и редове в таблиците, точно както работим с Python обекти. Полетата са интуитивни и можем да обработваме лесно информацията, която съхраняват, благодарение на системата за преобръщане на данни от таблицата към Python обекти – Django ORM (Object Relation Mapper)

На Фигура 9 е показана класовата диаграма на моделите в приложението. Виждат се два основни класа, `Order` и `MenuItem`. Третият клас `OrderItem` представлява `ManyToMany` връзката между другите два класа. Обикновено такъв клас не е необходим, тъй като повечето `ManyToMany` връзки предполагат по една инстанция на различни обекти. Например може да имаме няколко различни `MenuItem` в няколко различни `Order`. Случаят на нашето приложение е малко по-различен, тъй като в един `Order` трябва да може да бъдат добавени няколко бройки от няколко различни `MenuItem`. Например, в една поръчка може

да има три еднакви супи и две еднакви салати. Този междинен клас ни позволява да запазим бройката на различните MenuItem в даден Order. Ако не е създаден, Django ще създаде таблица за ManyToMany връзка само с двата ключа.



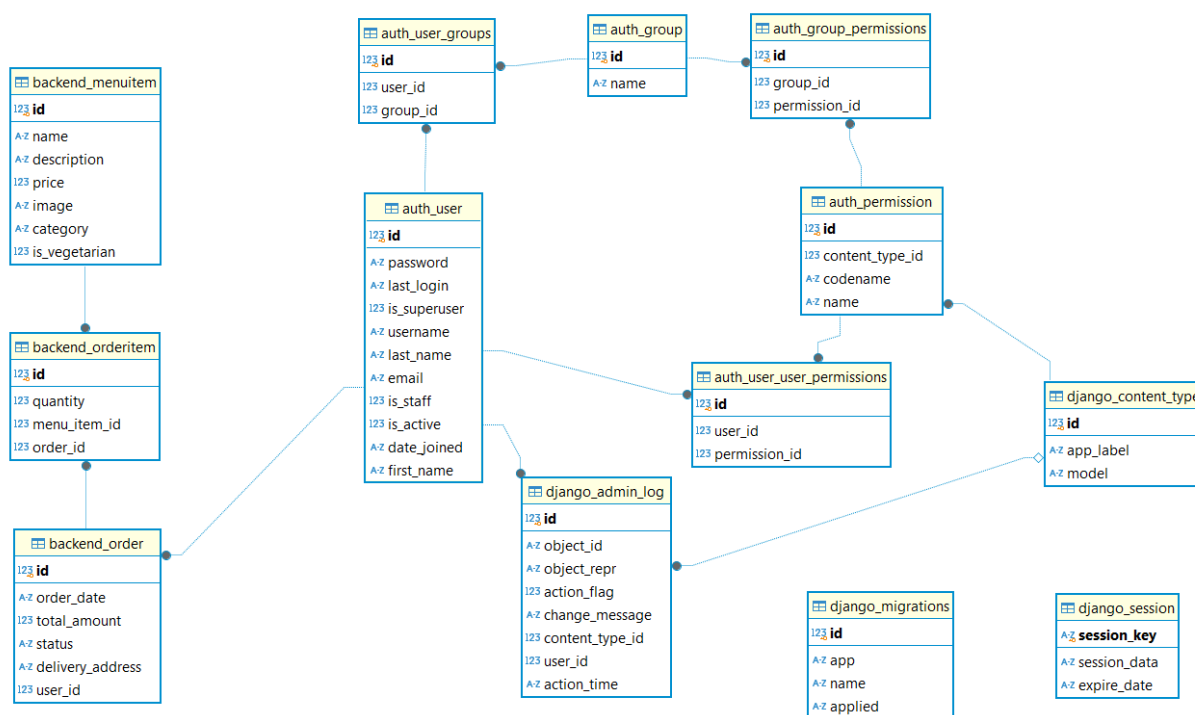
**ФИГУРА 9 КЛАС ДИАГРАМА НА МОДЕЛИТЕ**

На Фигура 10 са показани релациите между всички таблици в базата данни. Виждаме много повече таблици от дефинираните класове, тъй като Django създава свои служебни таблици за някои от функционалностите, вградени в библиотеката. Например, виждаме таблицата `auth_user`, която е таблицата, отговорна за потребителският модел на Django. Този

User модел ни дава допълнителни функционалности, които са особено полезни при безопасното съхранение на потребителските данни. Django съхранява паролите на потребителите в криптиран формат, така че при евентуален пробив в системата, атакуващите лица да нямат достъп до чувствителните данни на потребителите.

Това е една от основните добавени функционалности от библиотеката, която се занимава с автентикация на потребители, обработка на права на потребители и дори бисквитки за автентикация. Този модел може да бъде допълнително разширен от нас, за да добавим необходими полета или функционалност.

Таблиците, които започват с префикс backend\_ са таблици от нашето backend приложение. На Фигурата виждаме, че тези таблици са три. Имаме основните таблици за menuitem и order, както и свързващата таблица orderitem. Виждаме, че Django е създал таблиците много точно според предоставените моделни класове, като е запазил имената на полетата, които сме дали, и ги е използвал за имена на колоните в таблицата.



ФИГУРА 10 ENTITY RELATIONS ДИАГРАМА НА БАЗАТА ДАННИ

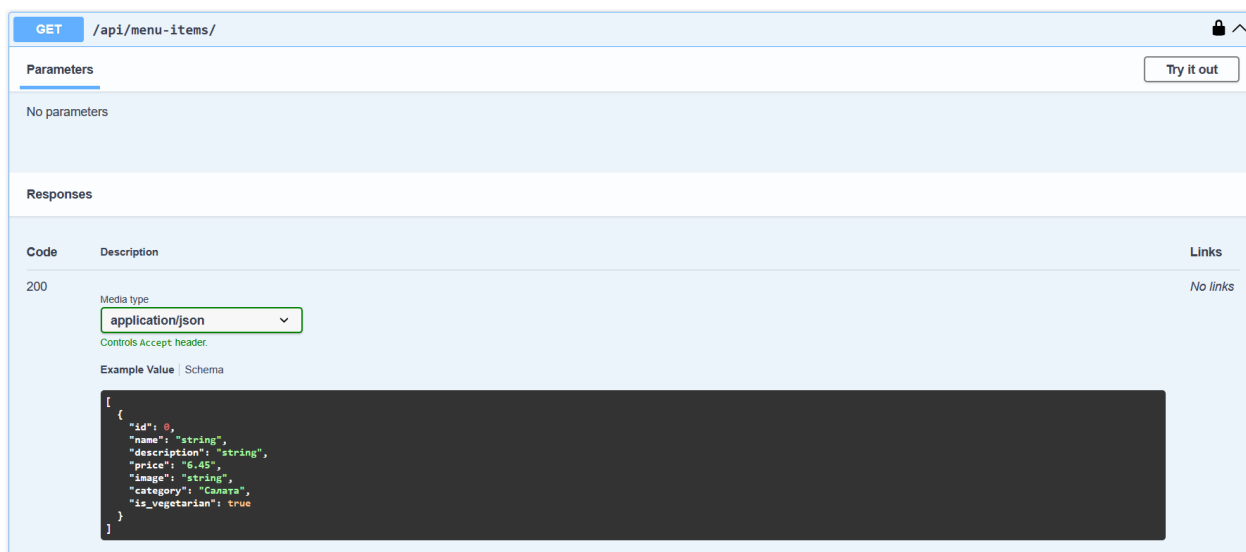
#### 4.1.2 Входни точки с OpenAPI спецификация

Входните точки на backend приложението са документирани чрез OpenAPI спецификация. Тя описва всички адреси, където може да бъде достъпено приложението, както и необходимите данни за да бъде изготвена заявка. Освен това, спецификацията описва възможните данни, които можем да получим като отговор от приложението, както и типовете данни, които са дефинирани специфично за приложението ни.

Тези входни точки не се предоставят от Django в базовия му вид. За да получим удобната спецификация, трябва да използваме Django REST Framework. Разширяващата библиотека ни позволява да пренапишем изгледите на Django, като добавим информация, необходима за изготвяне на дадената спецификация. При създаването на изглед, ние определяме сериализиращ клас, който да форматира данните ни. Този клас ще бъде използван при интроспекция за да бъде създадена спецификацията за OpenAPI [13]. Тази интроспекция позволява на DRF сам да открие необходимата му информация за изготвяне на спецификацията, с минимално включване от страна на разработчика. Както с всичко, обаче, е важно да се отбележи, че интроспекцията не винаги сработва, особено при по-сложни изгледи. Заради това, DRF ни предоставя начин да опишем ръчно части от спецификацията, от които не сме доволни.

Използвайки DRF, са описани осем входни точки в програмата, които са разделени на няколко подгрупи. Съществуват входни точки в група за работа с потребител, работа с поръчки и работа с елементи от менюто. Всяка категория съдържа една или повече входни точки, които позволяват работа с даден ресурс.

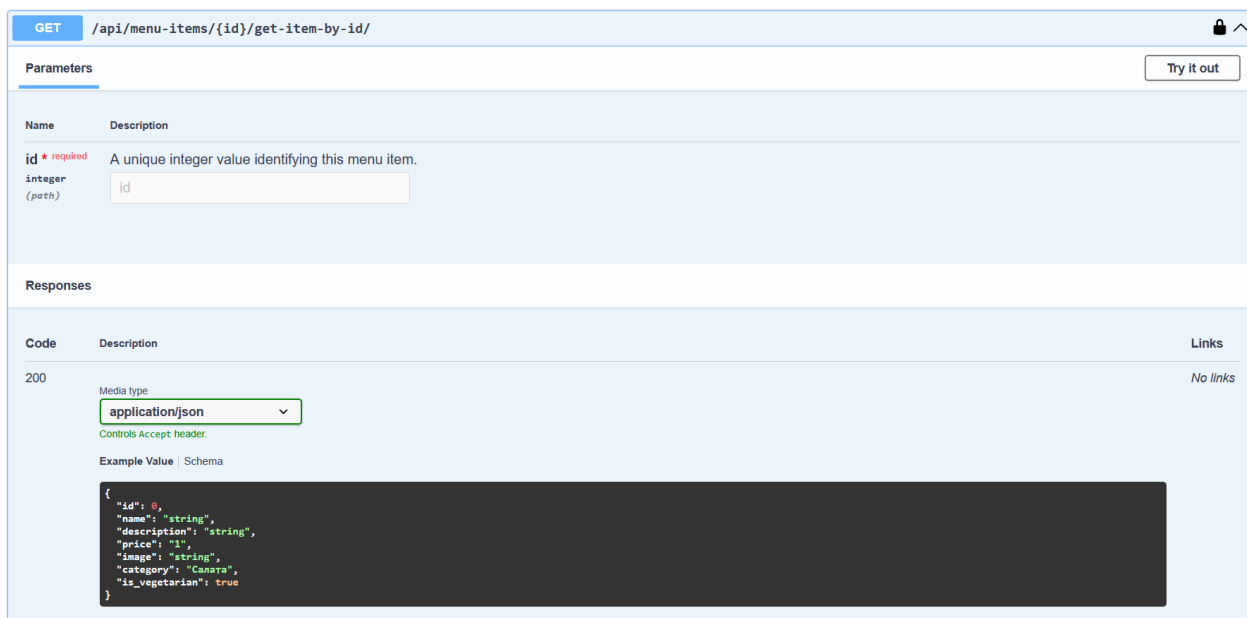
На Фигура 11 е показана входната точка за получаване на всички елементи от менюто. Тя е в групата menu-items, като представлява HTTP GET заявка. При успешно изпълнение на заявката, потребителят може да очаква списък от елементи на менюто и код HTTP\_200\_OK, като за пример е даден елемент от категория Салата.



**ФИГУРА 11** ВХОДНА ТОЧКА ЗА ПОЛУЧАВАНЕ НА ВСИЧКИ ЕЛЕМЕНТИ ОТ МЕНЮТО

На Фигура 12 е показана друга входна точка, отново от групата menu-items. На тази точка, чрез HTTP GET заявка, потребителят може да получи информацията за един елемент по неговият идентификационен номер (ID). Това се случва чрез параметър ID, който се подава директно в пътеката към входната точка. Този параметър ще бъде използван за извличане на конкретния елемент от базата данни и предоставянето му на потребителя.

Отново е показан примерен отговор, но този път не е списък от елементи, а е само един елемент, който би бил върнат при безпроблемно излизане със статус код HTTP\_200\_OK

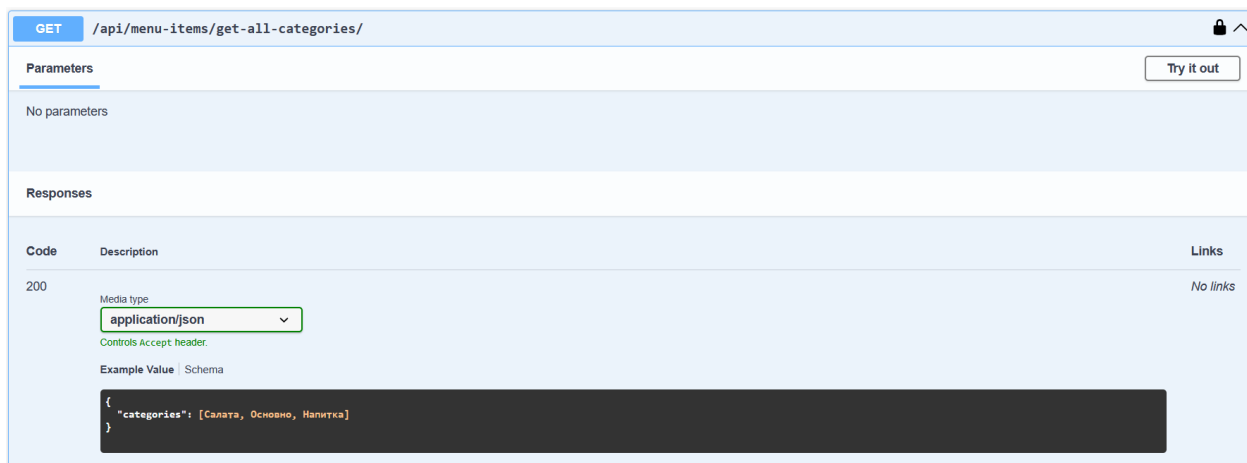


The image shows a Swagger UI interface for the endpoint `GET /api/menu-items/{id}/get-item-by-id/`. The **Parameters** section shows a required integer path parameter `id` with a description: "A unique integer value identifying this menu item." The **Responses** section shows a 200 status code with a description: "Media type" set to `application/json`. An example JSON response is displayed in a dark box:

```
{
  "id": 0,
  "name": "string",
  "description": "string",
  "price": "1",
  "image": "string",
  "category": "Canara",
  "is_vegetarian": true
}
```

ФИГУРА 12 ВХОДНА ТОЧКА ЗА ПОЛУЧАВАНЕ НА ЕЛЕМЕНТ ПО НЕГОВОТО ID

На Фигура 13 е показана още една входна точка от групата `menu-items`. Тази входна точка предоставя списък от категориите на елементи от менюто. Тази входна точка отново използва HTTP GET. При успешен отговор, връща HTTP\_200\_OK като статус код и JSON обект, състоящ се от ключ `categories` и списък със съществуващите в приложението категории на елементите от менюто.



The image shows a Swagger UI interface for the endpoint `GET /api/menu-items/get-all-categories/`. The **Parameters** section indicates "No parameters". The **Responses** section shows a 200 status code with a description: "Media type" set to `application/json`. An example JSON response is displayed in a dark box:

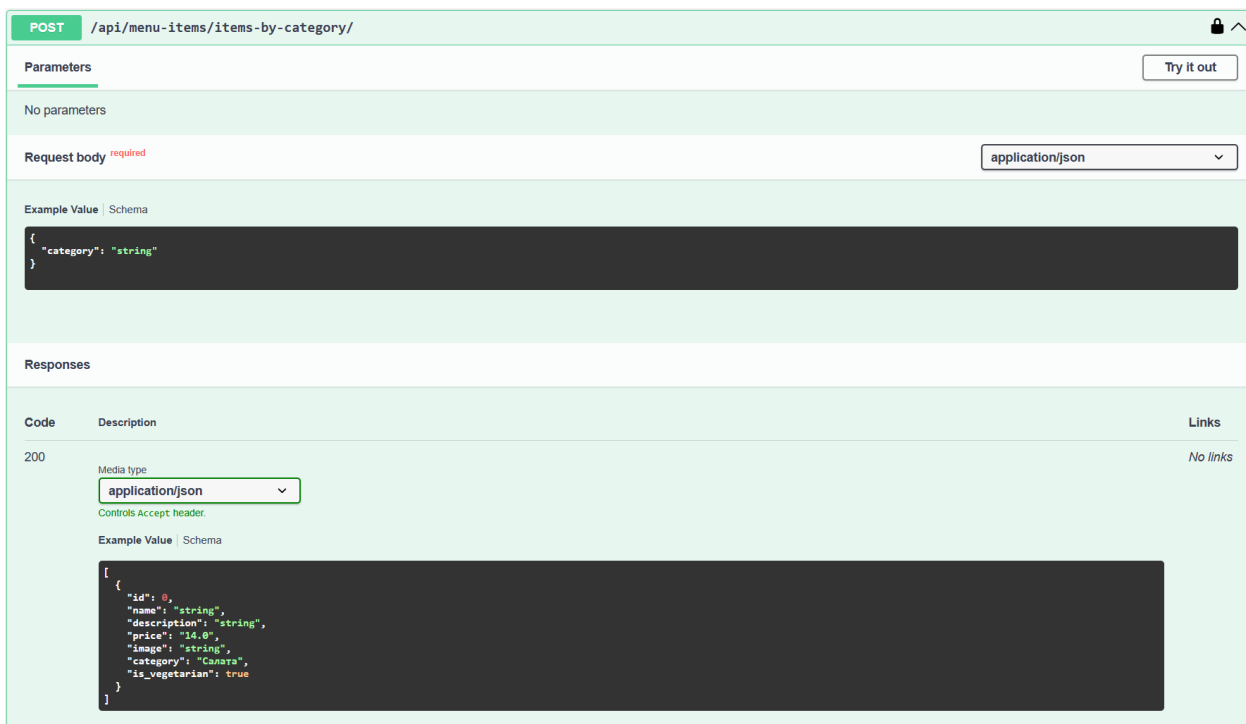
```
{
  "categories": ["Салата", "Основно", "Напитка"]
}
```

ФИГУРА 13 ВХОДНА ТОЧКА ЗА ВЗЕМАНЕ НА КАТЕГОРИИТЕ НА ЕЛЕМЕНТИТЕ ОТ МЕНЮТО



На Фигура 14 е показана последната входна точка от категорията menu-items. Тази входна точка се различава от разгледаните до сега по това, че тя е от тип HTTP POST и приема тяло към заявката, която получава, което е задължителен елемент на заявката.

В тялото на заявката се иска да има JSON обект, който да описва една категория от менюто. При подаване на категория, налична в базата данни, приложението ще попълни списък от елементи на менюто, които отговарят на тази категория и ще ги върне на потребителят, чрез статус код HTTP\_200\_OK и самия списък, отново в JSON формат.



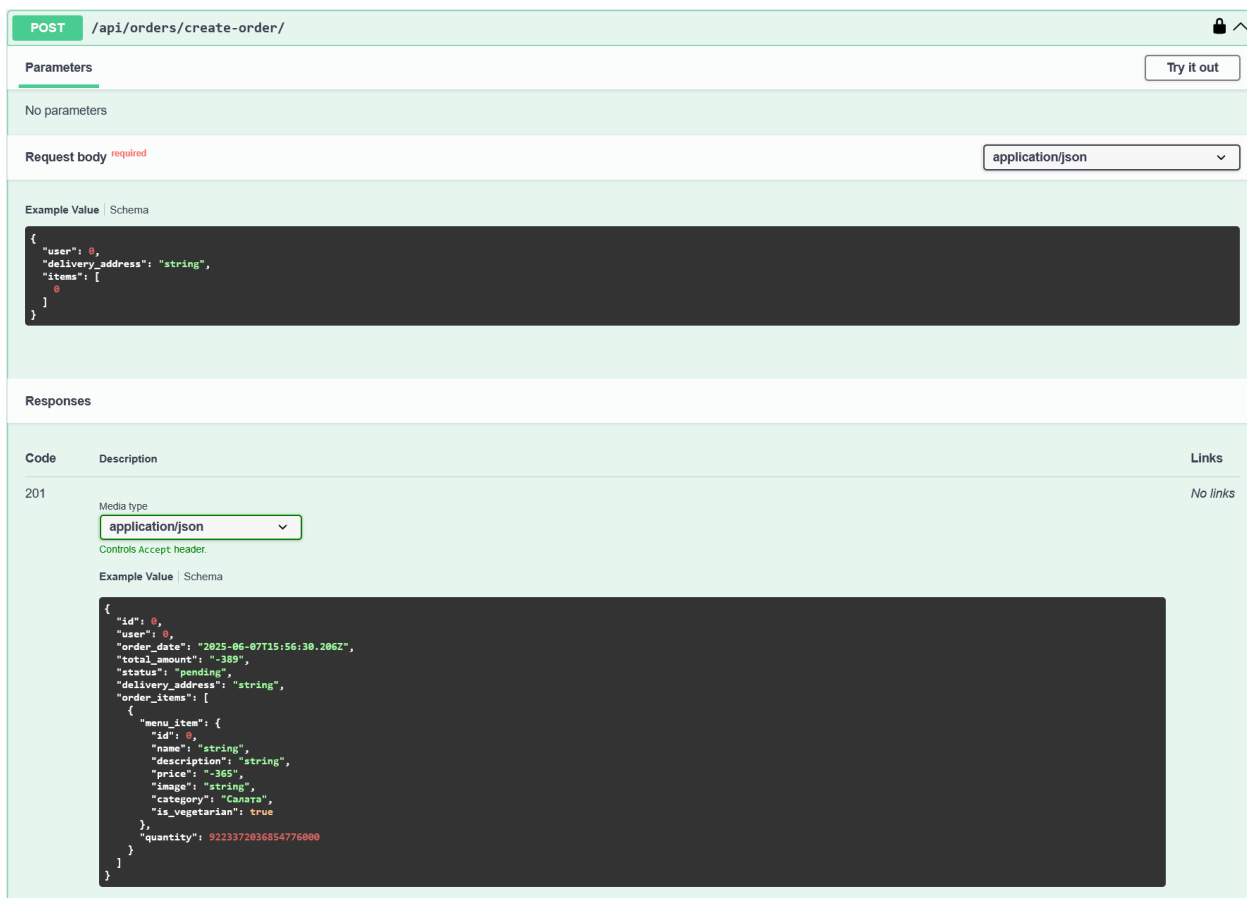
**ФИГУРА 14** ВХОДНА ТОЧКА ЗА ПОЛУЧАВАНЕ НА ВСИЧКИ ЕЛЕМЕНТИ ОТ ДАДЕНА КАТЕГОРИЯ

На Фигура 15 е показана първата входна точка от групата orders. Това е входната точка за създаване на поръчка. Входната точка е от тип HTTP POST, като изисква задължително тяло на заявката. В тялото трябва да има описани идентификатор на потребителя, адрес за доставката, както и списък от идентификатори на елементи от менюто.

Когато поръчката се обработва, тя трябва да бъде попълнена с останалите данни, необходими за базата данни. В отговорът на системата, потребителят трябва да получи статус код HTTP\_201\_CREATED, както и пълен обект на поръчка, която съдържа елементите от релацията с `order_items` и `menu_items`. По този начин, потребителското приложение може да валидира, че всичко е записано правилно.

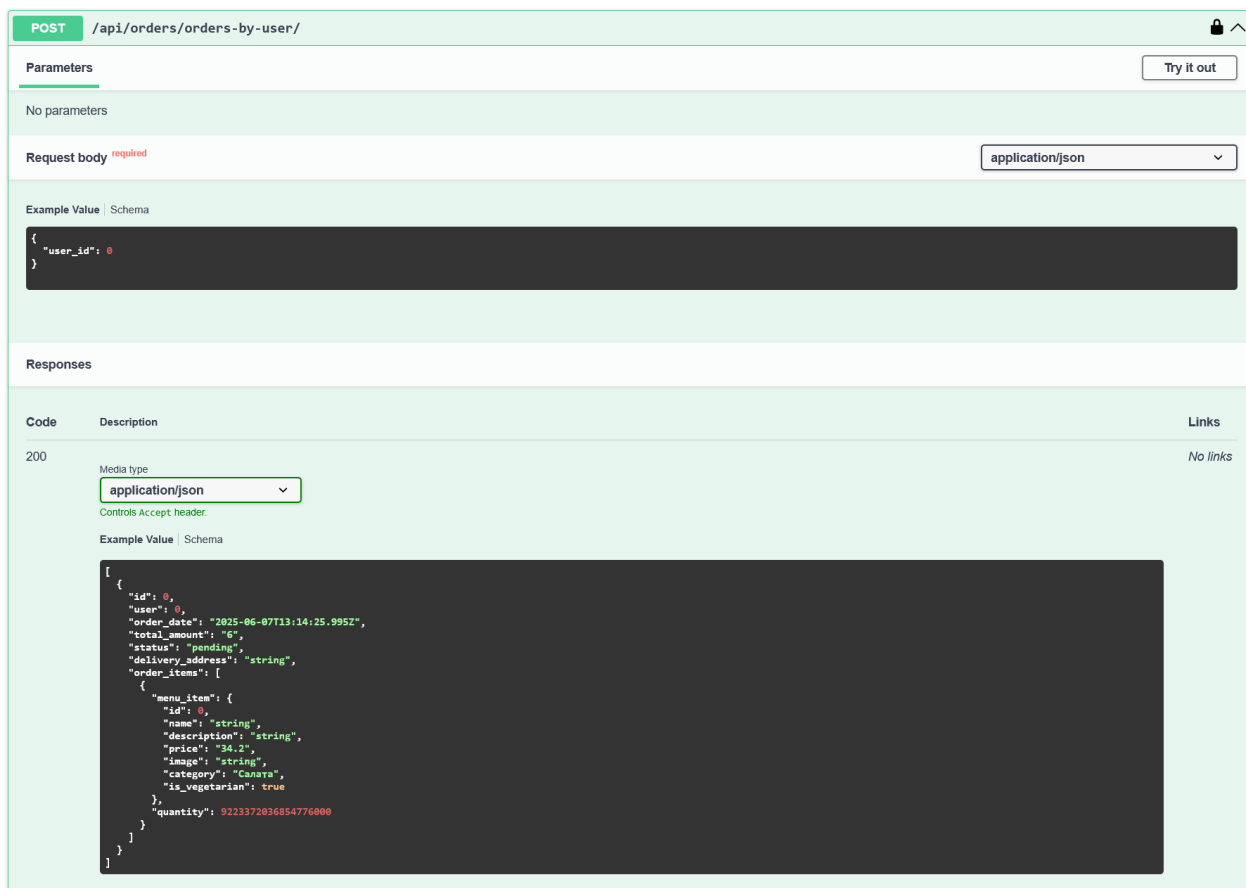
В примерният отговор виждаме създадената поръчка за потребител 0, която съдържа 2 елемента с идентификатор 0. Общата сума на поръчката се смята автоматично, на базата

на единичната цена. Броят на елементите за поле `order_items` също се смята автоматично, на базата на количеството еднакви идентификатори в заявката.



ФИГУРА 15 ВХОДНА ТОЧКА ЗА СЪЗДАВАНЕ НА ПОРЪЧКА

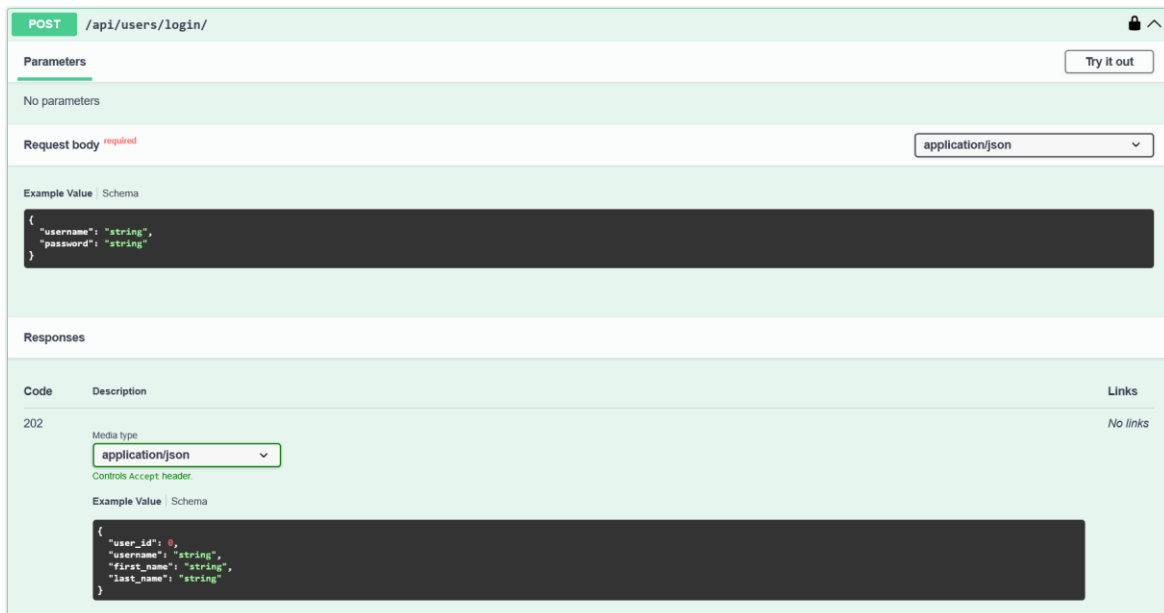
На Фигура 16 е показана входната точка за получаване на всички поръчки, създадени от конкретен потребител. Заявката е последната от групата на `orders` и е от тип HTTP POST. Имаме задължително тяло на заявката, което трябва да съдържа уникалният идентификатор на потребителя. Спрямо този индикатор, приложението трябва да филтрира всички поръчки и да даде тези, които се отнасят за потребителя. Върнатият отговор представлява списък от поръчките, като една поръчка означава детайлите за поръчката, всички елементи на `order_items` за поръчката, както и всички `menu_items` към всеки `order_item`. Освен този списък, потребителското приложение ще получи статус код HTTP\_200\_OK.



**Фигура 16** Входна точка за получаване на всички поръчки за даден потребител

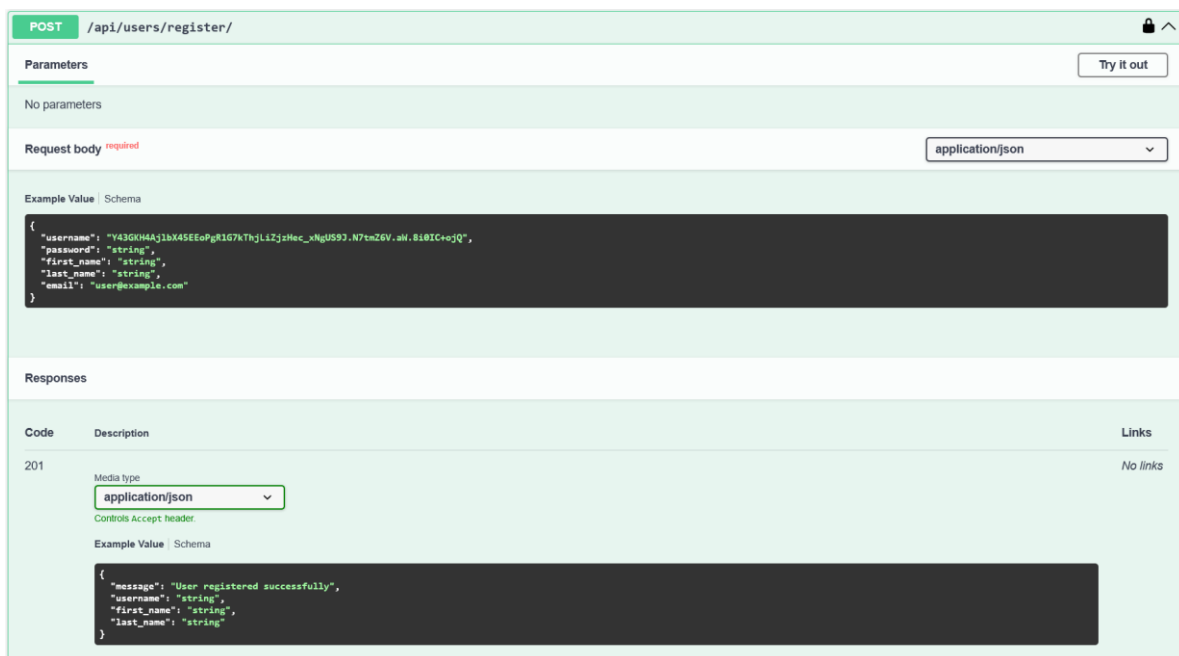
На Фигура 17 е показана първата входна точка от групата `users`. Тази входна точка представлява функционалността за влизане в системата. На тази входна точка потребителят трябва да изпрати данните си за вход в системата. Това се случва чрез заявка от тип HTTP POST и подаване на тяло към заявката, което е задължително. Тялото трябва да съдържа потребителското име и паролата на даденият потребител. Препоръчително е тези данни да бъдат хеширани предварително, за да не бъдат засечени при комуникация. Освен хеширането, комуникацията се извършва по протокол HTTPS. Това дава допълнително ниво на сигурност, за да подсигури, че потребителските данни ще останат непокътнати.

След като е подадена заявка за вход, приложението трябва да валидира данните спрямо своята база данни от потребители и да върне отговор дали потребителят е валидиран успешно. Това се случва с отговор на приложението със статус код `HTTP_202_ACCEPTED`, придружен с тяло на отговора, състоящо се от допълнителни данни за потребителя, като например неговият идентификационен код и неговите първо и фамилно име.



ФИГУРА 17 ВХОДНА ТОЧКА ЗА ПОДАВАНЕ НА ВХОДНИ ДАННИ ЗА ПОТРЕБИТЕЛ

На Фигура 18 е показана входната точка за регистрация на нов потребител. Входната точка е към групата users, като типът ѝ е HTTP POST. Има задължително тяло, което трябва да предостави информацията за потребителя, като потребителско име, парола, email, първо име и фамилно име. При успешно създаване на новата потребителска регистрация, приложението ще върне отговор със статус код HTTP\_201\_CREATED, съобщение за успешно създаване, както и имената на новосъздадения потребител.



ФИГУРА 18 ВХОДНА ТОЧКА ЗА СЪЗДАВАНЕ НА НОВ ПОТРЕБИТЕЛ

Входната точка за създаване на потребител работи директно с User класа на Django. Това значи, че при създаване на потребител, неговите чувствителни данни, като например паролата, се криптират с алгоритъм SHA256, използвайки таен ключ за допълнителна сигурност.

На Фигура 19 е показана допълнителна входна точка, която не се използва в стандартната работа с програмата. Тя е отделена от стандартната API група и е самостоятелна. Тази входна точка се използва за получаване на OpenAPI спецификация. Чрез нея, клиентски приложения могат да генерират клиент, който да комуникира с нашето приложение. Генерацията на клиент представлява създаване на клас, който има функции, които отговарят на входните точки на спецификацията. Чрез тях, клиентското приложение може да достъпва нашите входни точки и да извършва комуникация. Освен този клас ще бъдат генерирани и обекти, които да отговарят на всички възможни заявки и на всички възможни отговори, за да не се налага разработчикът на приложение да гадае какви данни са му били върнати, а да може директно да работи с данните посредством класове.

Такива клиенти могат да бъдат генерирани за голямо количество програмни езици с най-разнообразни настройки за генерацията, използвайки публични инструменти. Това е стандартът при работа с различни API и се използва от много компании по света. Най-разпространеният инструмент идва от разработчиците на спецификацията и се нарича OpenAPI Generator [14].

The screenshot displays a web interface for an OpenAPI schema endpoint. At the top, a blue header bar contains a 'GET' method indicator and the path '/schema/'. Below this, a light blue box provides information about the OpenAPI3 schema and lists supported media types: 'application/vnd.oai.openapi' (YAML) and 'application/vnd.oai.openapi+json' (JSON). A 'Parameters' section follows, featuring a table with columns for 'Name' and 'Description'. Two parameters are listed: 'format' (string, query) with a dropdown menu showing 'Available values: json, yaml', and 'lang' (string, query) with a dropdown menu showing 'Available values: af, ar, ar-dz, ast, az, be, bg, bn, br, bs, ca, ckb, cs, cy, da, de, dsb, el, en, en-au, en-gb, eo, es, es-ar, es-co, es-mx, es-ni, es-ve, et, eu, fa, fi, fr, fy, ga, gd, gl, he, hi, hr, hsb, hu, hy, ia, id, ig, io, is, it, ja, ka, kab, kk, km, kn, ko, ky, lb, lt, lv, mk, ml, mn, mr, ms, my, nb, ne, nl, nn, os, pa, pl, pt, pt-br, ro, ru, sk, sl, sq, sr, sr-latn, sv, sw, ta, te, tg, th, tk, tr, tt, udm, ug, uk, ur, uz, vi, zh-hans, zh-hant'. To the right of the parameters is a 'Try it out' button. Below the parameters is a 'Responses' section, which contains a table with columns for 'Code', 'Description', and 'Links'. A single response is listed with a status code of '200'. The description for this response includes a 'Media type' dropdown set to 'application/vnd.oai.openapi', a note about the 'Controls Accept header', and an 'Example Value' section showing a JSON object with three 'additionalProp1' fields, each containing a 'string' value.

Name	Description
format string (query)	Available values : json, yaml --
lang string (query)	Available values : af, ar, ar-dz, ast, az, be, bg, bn, br, bs, ca, ckb, cs, cy, da, de, dsb, el, en, en-au, en-gb, eo, es, es-ar, es-co, es-mx, es-ni, es-ve, et, eu, fa, fi, fr, fy, ga, gd, gl, he, hi, hr, hsb, hu, hy, ia, id, ig, io, is, it, ja, ka, kab, kk, km, kn, ko, ky, lb, lt, lv, mk, ml, mn, mr, ms, my, nb, ne, nl, nn, os, pa, pl, pt, pt-br, ro, ru, sk, sl, sq, sr, sr-latn, sv, sw, ta, te, tg, th, tk, tr, tt, udm, ug, uk, ur, uz, vi, zh-hans, zh-hant --

Code	Description	Links
200	Media type application/vnd.oai.openapi Controls Accept header Example Value   Schema <pre>{  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"}</pre>	No links

ФИГУРА 19 Входна точка за получаване на спецификация

### 4.1.3 Изгледи на приложението

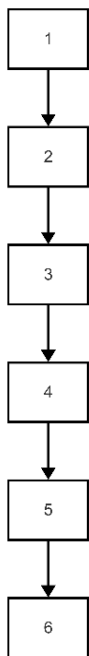
В Django логиката зад входните точки се разполага в така наречените изгледи. Те предоставят удобен начин за групиране на входните точки според какво обслужват. В нашето приложение имаме три изгледа: `MenuItemViewSet`, `OrderViewSet` и `UserViewSet`.

Всеки изглед наследява от клас `ViewSet`, който предоставя основни CRUD операции. Нашите изгледи са специализирани за работа с модела `'MenuItem'`, `'Order'` и `'User'`. `ViewSet`-ите са част от Django REST Framework и позволяват лесно създаване на API за работа с данни. Класът `'ViewSet'` предоставя основни методи за работа с HTTP заявки, като `'get'`, `'post'`, `'put'`, `'patch'` и `'delete'`, които могат да бъдат използвани за извършване на базови CRUD операции върху ресурсите.

В нашия случай, изгледите са специализирани за работа с модела `'MenuItem'`, `'Order'` и `'User'`. Всеки изглед предоставя методи за извличане на данни, създаване на нови записи, актуализиране на съществуващи и други.

`MenuItemViewSet` е изглед, който предоставя операции за меню елементи. Съдържа методите `get_all_categories`, `items_by_category`, `get_all_items` и `get_item_by_id`, които дават връзката към входните точки за извличане на всички категории, елементи по категория, списък от всички елементи и елемент според неговия идентификатор. Тези методи използват Django ORM за получаване на данни от базата данни и връщат JSON сериализирани отговори.

Потокът на работа на метода `get_all_categories` е показан на Фигура 20. Методът представлява последователност от действия, без разклонения.

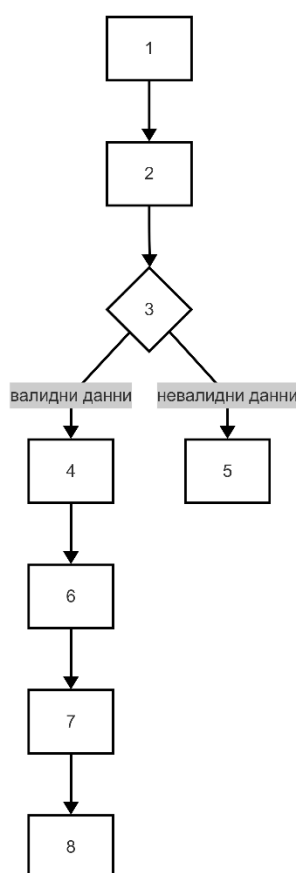


**ФИГУРА 20 ПОТОК НА РАБОТА ЗА МЕТОДА `GET_ALL_CATEGORIES`**

Стъпките на метода са:

1. Създаване на GET заявка към `api/menu-item/get-all-categories`
2. Обработване на заявката от сървъра
3. Извличане на всички уникални категории на елементите в базата данни
4. Формиране на списък с намерените категории
5. Създаване на отговор с категориите в JSON формат
6. Връщане на отговора към клиента

Потокът на работа на метода `items_by_category` е представен на Фигура 21. Методът има една проверка, в която се установява дали данните, които са получени, са валидни.



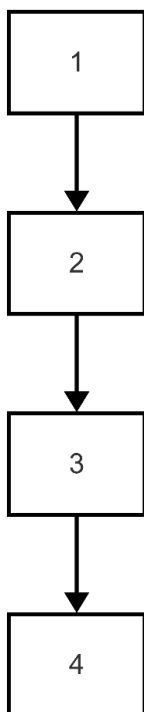
**ФИГУРА 21 ПОТОК НА РАБОТА НА МЕТОДА `ITEMS_BY_CATEGORY`**

Потокът на работа на метода е както следва:

1. Получаване на POST заявка към `api/menu-item/items-by-category`
2. Десериализиране на входните данни
3. Проверка за валидност на данните.
  - a. Ако са валидни, премини към 4.
  - b. Ако не, премини към 5.

4. Извличане на стойността на категорията от валидираните данни
5. Връщане на грешка със статус 400
6. Извличане на елементи от базата данни, които принадлежат към подадената категория
7. Сериализиране на намерените елементи
8. Връщане на отговор към клиента със списък от елементи в JSON формат

На Фигура 22 е показана последователността на изпълнение на метода `get_all_items`. Този метод не се разклонява, тъй като не проверява данните, а директно връща на потребителя каквото е успял да прочете от базата данни.



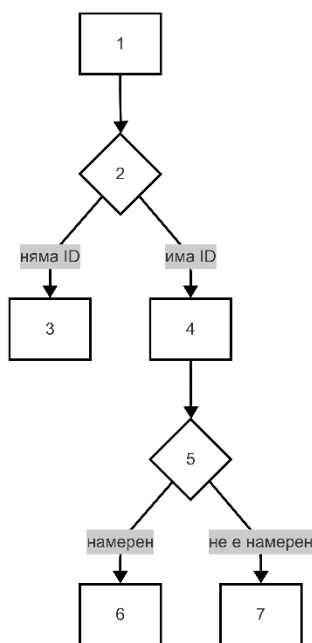
**ФИГУРА 22 ПОТОК НА РАБОТА НА МЕТОДА `GET_ALL_ITEMS`**

Последователността е:

1. Получаване на GET заявка към `api/menu-items`
2. Извличане на всички записи от базата данни
3. Сериализиране на списъка с елементи
4. Връщане на отговор към клиента със сериализираните данни в JSON формат

На Фигура 23 е показан потока на работа на метода `get_item_by_id`. Този метод извлича елемент от базата данни, според това какъв идентификатор му е подаден като част от заявката.





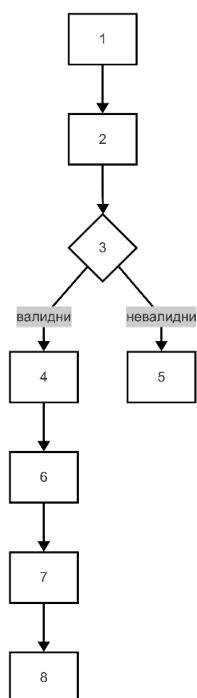
**ФИГУРА 23 ПОТОК НА МЕТОДА GET\_ITEM\_BY\_ID**

Потокът на метода е:

1. Получаване на GET заявка към `api/menu-items/<id>/get-item-by-id`
2. Проверка дали е подаден `item_id`
  - a. Ако няма ID, премини към 3
  - b. Ако има ID, премини към 4
3. Връщане на грешка със статус 400
4. Опит за извличане на елемента от базата по ID
5. Проверка дали елементът съществува
  - a. Ако съществува, премини към 6
  - b. Ако не съществува, премини към 7
6. Сериализиране и връщане на намерения елемент в JSON формат
7. Връщане на грешка със статус 404

OrderViewSet е изглед, който предоставя операции за поръчки. Съдържа методите `orders_by_user` и `create_order`, които дават връзката към входните точки за извличане на поръчки по идентификатор на потребител и създаване на поръчка. Тези методи също използват Django ORM за извличане на данни от базата данни и връщат JSON сериализирани отговори.

На Фигура 24 е показан потокът на работа на метода `orders_by_user`, който извлича поръчки по идентификатор на потребител. Методът има проверка, в която валидира входните данни от заявката и връща статус различен от HTTP\_200\_OK ако има проблем.

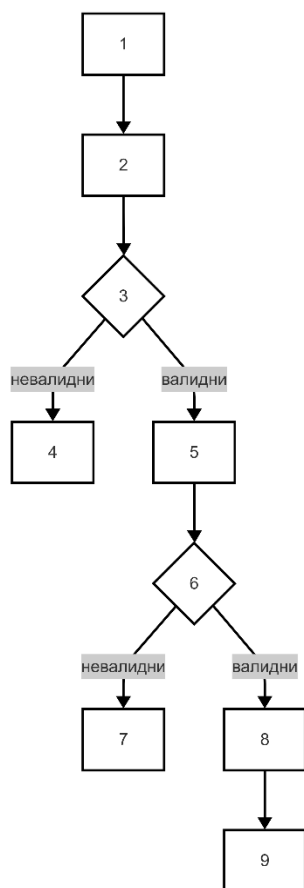


**ФИГУРА 24 ПОТОК НА РАБОТА НА МЕТОДА ORDERS\_BY\_USER**

Потокът на изпълнение на метода е:

1. Получаване на POST заявка към `api/orders/orders-by-user`
2. Десериализиране на входните данни
3. Проверка за валидност на подадените данни.
  - a. Ако данните са валидни, преминава към 4.
  - b. Ако не са, преминава към 5
4. Извличане на `user_id` от валидираните данни
5. Връщане на грешка със статус 400
6. Извличане на всички поръчки, направени от съответния потребител, сортирани по дата в низходящ ред
7. Сериализиране на списъка с поръчки
8. Връщане на отговор с поръчките в JSON формат

На Фигура 25 е показан потока на метода `create_order`, който създава поръчка. Данните за поръчката се вземат от тялото на заявката, като преди самото създаване се валидират.



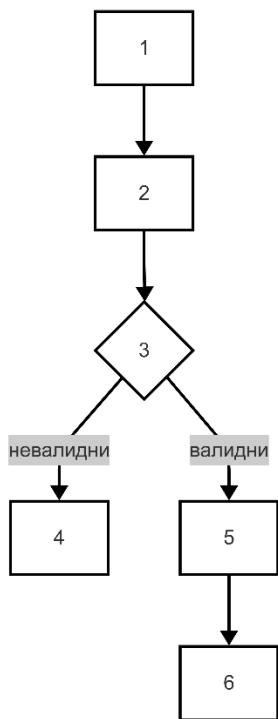
**ФИГУРА 25 ПОТОК НА ИЗПЪЛНЕНИЕ НА МЕТОДА `CREATE_ORDER`**

Потокът на изпълнение е:

1. Получаване на POST заявка към `api/orders/create-order`
2. Десериализиране на подадените данни
3. Проверка дали данните са валидни
  - a. Ако са невалидни, премини към 4
  - b. Ако са валидни, премини към 5
4. Връщане на грешка със статус 400
5. Създаване на сериализирана поръчка с вече валидираните данни
6. Проверка дали поръчката е валидна
  - a. Ако не е, премини към 7
  - b. Ако е, премини към 8
7. Връщане на грешка със статус 400
8. Записване на новата поръчка в базата данни
9. Връщане на сериализирания отговор с новосъздадената поръчка и статус 201 CREATED

UserViewSet е изглед, който предоставя операции за потребители. Съдържа методите register и login, които дават връзката към входните точки за регистрация и вход на потребител. Тези методи използват Django ORM за работа с потребителския модел и предоставят JSON сериализирани отговори.

На Фигура 26 е показан потокът на работа на метода register, който позволява на потребител да се регистрира в системата. Методът трябва да провери дали данните на потребителя са валидни. Това включва валидация на e-mail, проверка за вече съществуващи потребители и други.

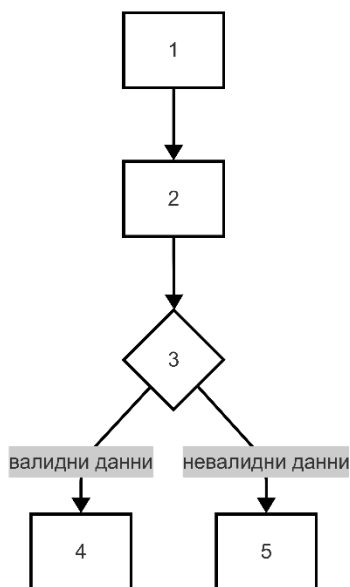


**ФИГУРА 26 ПОТОК НА МЕТОДА REGISTER**

Потокът на изпълнение е както следва:

1. Получаване на POST заявка към api/users/register
2. Десериализиране на входните данни
3. Проверка за валидност на сериализатора
  - a. Ако не е валиден, премини към 4
  - b. Ако е валиден, премини към 5
4. Връщане на грешка със статус 400
5. Създаване на нов потребител
6. Връщане на отговор със статус 201 CREATED и съобщение, заедно с информация за създадения потребител

На Фигура 27 е показана последователността на работа на метода login, който позволява на потребителите да влизат в системата. Методът трябва да провери дали входните данни са коректни и ако са, да върне подобаващ статус код.



**ФИГУРА 27 ПОТКОК НА МЕТОДА LOGIN**

Потокът на изпълнение на метода е:

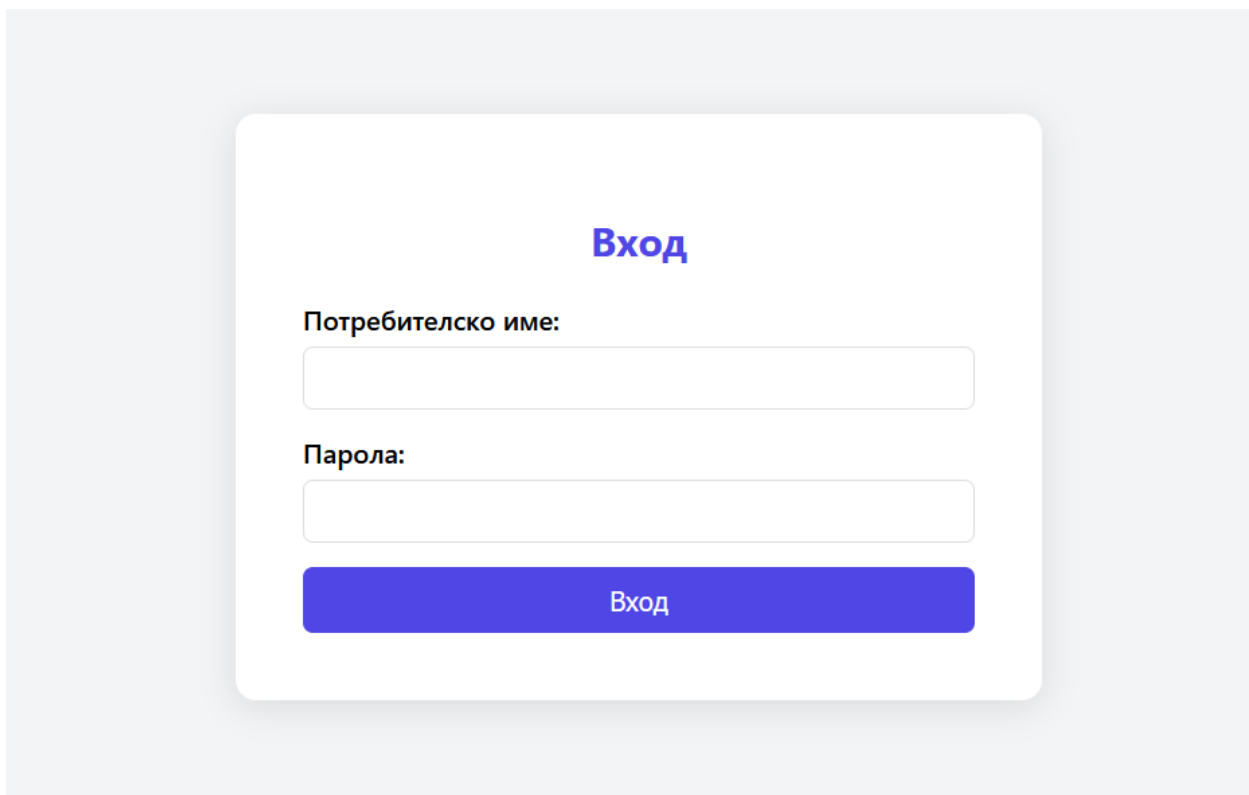
1. Получаване на POST заявка към `api/users/login`
2. Десериализиране на входните данни
3. Проверка за валидност на данните
  - a. Ако данните са валидни, премини към 4
  - b. Ако данните не са валидни, премини към 5
4. Връщане на отговор със статус 202 и информация за потребителя
5. Връщане на грешка със статус 400

#### **4.1.4 Реализация на Django Уеб Страница**

За да позволява на персонала на кухнята да следи постъпващите поръчки, е създадено уеб приложение. Django позволява да изградим допълнителни изгледи, които не са част от нашето API, за визуализация на HTML-базирани уеб страници. Този вид изгледи добавя известна динамика и интерактивност на приложението. Както с всички останали функционалности, Django обработва нашите HTML страници сам, без да се налага някаква особена конфигурация от наша страна. Освен това, ни можем да използваме HTML темплейти, които да включват Python код, който Django по-късно да интерпретира при зареждането на страницата [15].

За целите на приложението са разработени две уеб страници, които предоставят достъп на персонала до данните. Едната страница се използва за вход в системата, а другата за достъпване на данните.

За вход в системата потребителят трябва да бъде добавен през административния панел на Django. По този начин се подsigуряваме, че не всеки потребител може да си направи регистрация и да достъпи нашата система. На Фигура 28 е показана страницата за вход в системата. Виждаме доста проста страница, чиято единствена цел е да не позволява неоторизиран достъп.



**ФИГУРА 28** СТРАНИЦА ЗА ВХОД В СИСТЕМАТА

Страницата за визуализация на поръчки е основната за уеб приложението. Тя позволява да се разгледат поръчките от деня, като също позволява да се разглеждат и отминали дни. Страницата е показана на Фигура 29. Там виждаме общият изглед, където се виждат 3 поръчки, създадени от административният акаунт „kala”. Страницата сама смята общата сума приход за деня, като я визуализира в горният ляв край на страницата. На дадения пример се виждат само първите три поръчки за дадения ден. Също можем да видим, че поръчките са групирани според своя статус. На показания ден, първите три поръчки са оставени със статус Pending. Освен поръчките и сумата за деня, можем да филтрираме данните по дата. Това се случва с календара от дясната страна на екрана. Когато кликнем върху него, получаваме календарен изглед, от където можем да изберем дата. Важно е да се отбележи, че е невъзможно да се избере дата, на която няма поръчки.

Добре дошли, kala

Изход

## Поръчки за June 8, 2025 групирани по статус

Общо за деня: 91.18лв.

Изберете дата: 2025-06-08

Филтрирай

### Поръчки със статус изчакване

Клиент	Email	Артикули	Общо	Дата	Адрес	Действие
Giovanni Giorgio	jojo@ja.co	<ul style="list-style-type: none"> <li>Чипс от моркови - 4.99лв. x 1</li> <li>Гризини от нахут - 7.88лв. x 1</li> </ul>	12.87лв.	June 8, 2025, 8:39 p.m.	Маса 7	Премести към Приготвяне
Giovanni Giorgio	jojo@ja.co	<ul style="list-style-type: none"> <li>Бяла риба с чери домати - 13.45лв. x 1</li> <li>Наденица с картофи на фурна - 9.99лв. x 1</li> <li>Пилешко филе с топено сирене - 11.45лв. x 1</li> <li>Айрян - 2.50лв. x 1</li> <li>Бананов пудинг - 8.47лв. x 1</li> <li>Бананов сладолед с фъстъци - 5.55лв. x 1</li> </ul>	51.41лв.	June 8, 2025, 8:39 p.m.	ул. Люлин, 2351 Dragichevo, Bulgaria	Премести към Приготвяне

### Поръчки със статус приготвяне

ФИГУРА 29 СТРАНИЦА С ИНФОРМАЦИЯ ЗА ПОРЪЧКИ

## 4.2 Реализация на маяк

Маяците, които ще бъдат използвани за маркиране на различните маси, ще използват чипове ESP32. Тези чипове ни позволяват бързо и лесно да създадем маяк, като също имат ниска цена за това, което предлагат.

За да бъдат лесни за разпознаване маяците, те имат фиксиран идентификатор, чрез който да бъдат различавани от останалите устройства. Освен това, тяхното име представлява идентификаторът на отделните маси. Когато маяците излъчват сигнала си, те споделят точно това име и индикатор, чрез които да могат да бъдат разпознати. Когато мобилно устройство разпознае няколко такива маяци, то може да използва RSSI на получения сигнал и да определи кое от устройствата е най-близо до него. Чрез този процес, мобилното приложение може да определи как се казва масата, на която е седнал клиентът, и да изпрати коректно поръчката към кухнята.

## 4.3 Реализация на мобилно приложение

Мобилното приложение е разработено за операционната система Android, като поддържа Android API version 26 и всички последващи версии. Съобразени са специалните изисквания за използване на услуги като Bluetooth сканиране, което е необходимо за идентификация на маса, Location access, който е необходим ако се окаже, че потребителят не е в близост до маяк и трябва да се вземе текущото му местоположение.

Разработката е извършена с помощта на Android Studio, като е използван Kotlin езикът за програмиране. Архитектурата на приложението следва модела MVVM (Model-View-ViewModel), който разделя логиката на приложението от потребителския интерфейс,

позволявайки по-добра поддръжка и разширяемост. Приложението е проектирано да бъде интуитивно и лесно за използване, като предоставя потребителски интерфейс, който позволява на потребителите да взаимодействат с ресторантската система. Потребителите могат да разглеждат менюто, да правят поръчки и да преглеждат историята на поръчките си.

Мобилното приложение използва REST API, предоставено от Django приложението, за да комуникира с backend частта. Това позволява на мобилното приложение да извлича данни за менюто, да създава поръчки и да управлява потребителските акаунти. Всички заявки към API-то са асинхронни, което осигурява плавно потребителско изживяване без забавяне при зареждане на данни.

Кодът за използване REST API-то е генериран с помощта на OpenAPI Generator, който автоматично създава клиентски код за Android приложение на базата на OpenAPI спецификацията. Това позволява бързо и лесно интегриране на API-то в мобилното приложение, като се минимизира ръчната работа и се намалява вероятността от грешки.

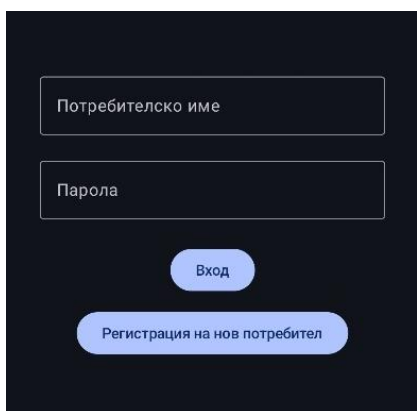
Приложението е разделено на няколко основни пакета, които организират кода по логически групи:

- **apiclient:** Съдържа генерирания клиентски код за взаимодействие с REST API-то. Този пакет включва класове и методи за извършване на HTTP заявки към бекенда, обработка на отговорите и управление на грешки.
- **data:** съдържа класове за работа с Bluetooth услуги, които позволяват на приложението да сканира за маяци и да идентифицира маси в ресторанта. Този пакет включва логика за управление на Bluetooth връзки и обработка на данни от маяците.
- **di:** Съдържа класове за внедряване на зависимости, които осигуряват инстанции на различни компоненти на приложението. Този пакет използва Dagger [16] за управление на зависимостите и осигурява лесно конфигуриране и тестване на компонентите.
- **presentation:** Съдържа класове за потребителския интерфейс, които реализират различните екрани на приложението. Тук са включени класове за управление на навигацията, показване на екрани с меню елементи, поръчки и потребителски профили. Този пакет използва Android Jetpack [17] компоненти за управление на жизнения цикъл и навигацията. Освен това се използва Jetpack Compose [18], което е част от Android Jetpack за създаване на визуалните елементи на приложението.
- **ui:** Съхранява някои теми и ресурси за потребителския интерфейс, като икони, цветове и стилове. Този пакет осигурява визуалната част на приложението и позволява лесно персонализиране на външния вид.
- **utils:** Съдържа помощни класове и функции, които се използват в различни части на приложението. Този пакет включва логика за обработка на данни, форматиране на дати и други общи операции, които не са свързани с конкретен компонент на приложението.



### 4.3.1 Екран за вход

Това е началният екран на приложението. При зареждане, той се отваря първи. Тук потребителят може да въведе своите данни и да влезе в приложението или, ако няма създаден профил, да направи нова регистрация. На Фигура 30 е показан екранът за вход в приложението. Виждаме двете полета за потребителско име и парола, както и бутона за вход. При опит за вход, приложението ще направи POST заявка към `/api/users/login/`. От тук, потребителят може да навигира към екрана за регистрация или към менюто.

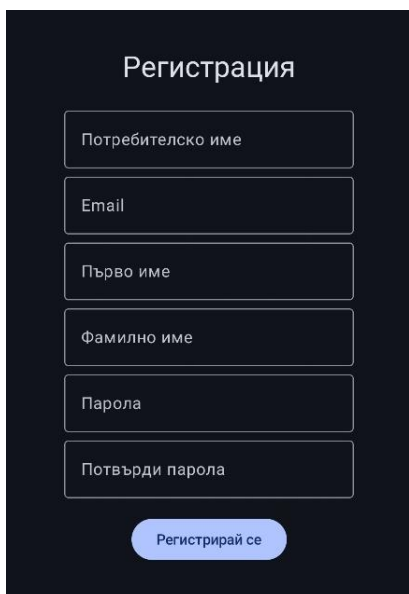


**ФИГУРА 30** ЕКРАН ЗА ВХОД В ПРИЛОЖЕНИЕТО

Важно е да се отбележи, че при изпълнение на заявката за вход, паролата не се подава като текст, а преди това се хешира. По този начин, ние предоставяме допълнително ниво на сигурност, освен HTTPS.

### 4.3.2 Екран за регистрация

Екранът за регистрация предоставя възможност за новите потребители да създадат профил в системата. Нов потребител трябва да въведе уникално потребителско име, уникален e-mail адрес, първо име, фамилно име и парола. Паролата трябва да бъде повторена, за да бъде сигурно, че не е въведена грешно по невнимание. При създаване на регистрация се прави POST заявка към `/api/users/register/`. На Фигура 31 е показан екрана за регистрация в системата.

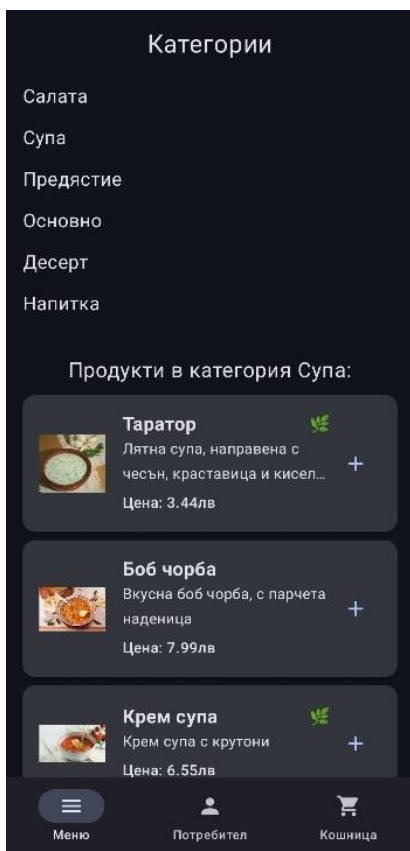
A screenshot of a registration form titled "Регистрация" (Registration) on a dark background. The form contains six input fields stacked vertically: "Потребителско име" (Username), "Email", "Първо име" (First name), "Фамилно име" (Last name), "Парола" (Password), and "Потвърди парола" (Confirm password). Below these fields is a blue button labeled "Регистрирай се" (Register).

**ФИГУРА 31** ЕКРАН ЗА РЕГИСТРАЦИЯ В СИСТЕМАТА

Тук отново, при създаване на заявката за регистрация, паролата на потребителя се хешира, за да не разчитаме само на HTTPS за защита на данните.

#### **4.3.3 Екран за преглед на меню**

След като потребителят вече е регистриран и е влязъл в системата, той бива пренасочен към меню екрана. Този екран дава възможност на потребителя да навигира из приложението, както и да разглежда менюто. Тук различните храни са групирани по категории, като на екрана се показват храните само от една категория, за да се подобри четливостта. При избиране на категория, приложението прави POST заявка с избраната категория към `/api/menu-items/items-by-category`. По този начин, то получава елементите на менюто за дадената категория и може да ги визуализира. След като елементите са визуализирани, потребителят ще може да види кратка информация за всяко ястие, както и ще може да добави ястието към своята кошница. На Фигура 32 е показана страницата с менюто, като са заредени артикули от категория Супа.



**ФИГУРА 32** ЕКРАН ЗА ПРЕГЛЕД НА МЕНЮ

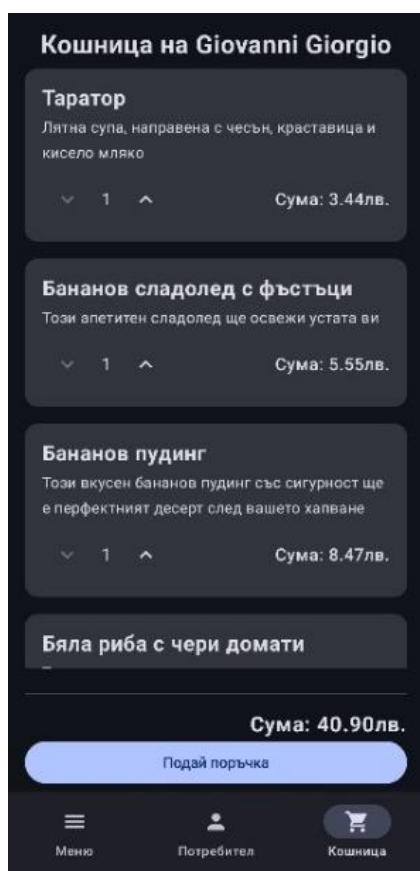
От тази страница, потребителят може да навигира към кошницата или към страницата с предходни поръчки. За целта се използва долната навигационна лента, която позволява бързо движение между различните страници. Тази лента е видима на страниците за меню, потребител и кошница. Страниците за вход, регистрация и детайли на поръчка нямат достъп до тази лента.

Изображенията на храните, използвани за тази разработка, са взети от Mate Kitchen [19]. Водният знак на всяко изображение е запазен. Mate Kitchen е уебсайт, който предлага голямо разнообразие от рецепти, които могат да задоволят всеки вкус. Освен богатият избор от рецепти, Mate Kitchen има онлайн магазин, от който могат да бъдат закупени кухненски пособия.

#### 4.3.4 Екран за кошница

Екранът за кошница се използва за приключване на поръчката. На този екран са визуализирани всички продукти, които потребителят е добавил от екрана за меню. Освен това, те са групирани според типа си. Ако потребителят е добавил продукт, например таратор, два или повече пъти, то продуктът няма да бъде визуализиран по веднъж за всяко добавяне, ами ще бъде визуализиран веднъж, заедно с бройката повторни добавяния. От там потребителят може да промени броя поръчан артикул, ако е направил грешка при

добавянето. В долния край на екрана, потребителя може да види колко е общата сума на сметката му и също бутон за създаване на поръчка. На Фигура 33 е показан екранът за кошница.



ФИГУРА 33 ЕКРАН ЗА КОШНИЦА

При натискането на бутон „Подай поръчка“, приложението ще опита да определи къде се намира клиентът. Това може да се случи по един от три начина: чрез BLE маяк, чрез локацията на мобилното устройство или чрез ръчно въвеждане. Трите метода за откриване на локация се изпълняват последователно, като с най-висок приоритет е използването на BLE маяк. Ако това не е успешно, приложението преминава към използването на локацията на мобилното устройство. Ако и това не е успешно, то се преминава към ръчно въвеждане.

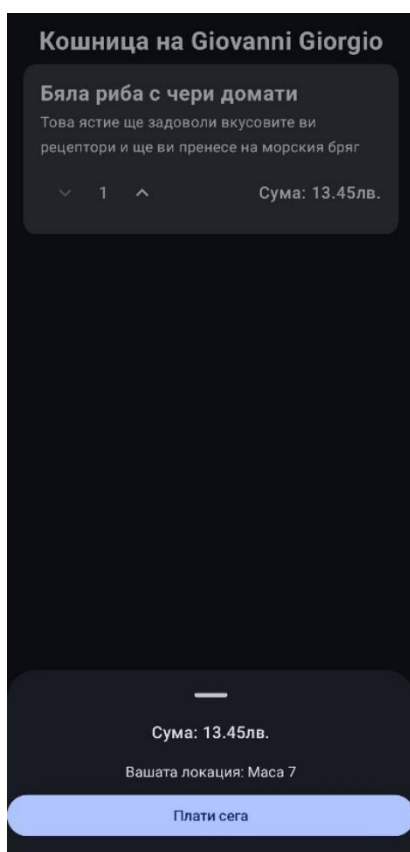
При варианта с BLE маяк, приложението ще се опита да определи дали клиентът се намира в ресторант. Ако това е така, то трябва да има BLE маяк наблизо. Приложението ще сканира за маяци и ако успее да определи някой маяк като маса във физически ресторант, то за локация ще покаже идентификатора на масата.

Ако маяк не е засечен, то приложението ще приеме, че потребителят иска да получи храната си на адрес извън ресторанта. В този случай, то ще се опита да използва локацията на мобилното устройство за да определи къде се намира потребителят. При успешно

определяне на местоположението, то ще бъде конвертирано към уличен адрес и този адрес ще бъде използван за поръчката.

Ако приложението не може успешно да определи близост до маяк или локация, то ще информира потребителя, че не е успяло да определи локацията му и ще го помоли да я въведе ръчно. Ако потребителят опита да направи поръчка без въведен адрес, то приложението няма да му позволи.

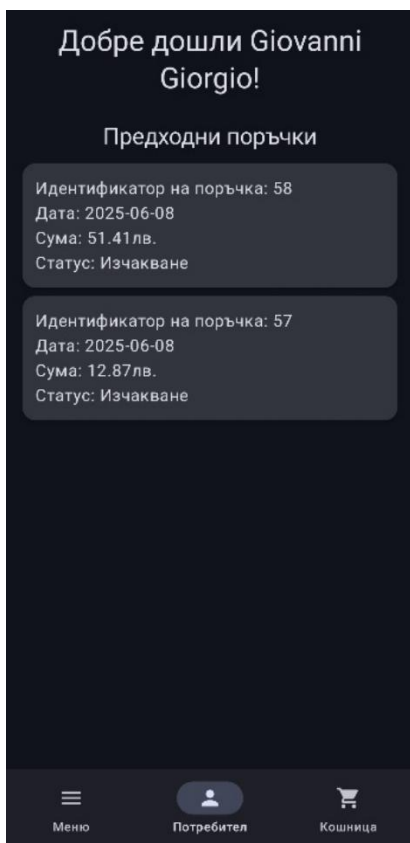
След като е определена локация, поръчката се създава чрез POST заявка към `/api/orders/create-order/`. На Фигура 34 е показано как изглежда завършването на поръчката, когато приложението е засякло маяк.



**ФИГУРА 34** ЕКРАН ЗА ЗАВЪРШВАНЕ НА ПОРЪЧКАТА ПРИ ЗАСЕЧЕН МАЯК

#### 4.3.5 Екран за потребител

На екранът за потребител се показва информацията за предходни поръчки на дадения потребител. Там, той може да проследи своите разходи, както и да разгледа подробно предходни поръчки, чрез навигация към екран за детайли на поръчка. При зареждане на страницата се прави заявка GET към `/api/orders/orders-by-user/`, която зарежда данните за поръчките на даденият потребител. След като поръчките са заредени, потребителят може да избере конкретна поръчка, чрез докосване, за да разгледа по-детайлно информацията за нея. На Фигура 35 е показан екранът за информация за поръчките на потребител.



**ФИГУРА 35** ЕКРАН ЗА ПОРЪЧКИТЕ НА ПОТРЕБИТЕЛ

#### **4.3.6 Екран за преглед на детайли на поръчка**

Този екран предоставя възможност на потребителите да разглеждат детайлите за техни съществуващи поръчки. Детайлите на поръчката включват информация като датата на поръчката, адресът на поръчката и поръчаните продукти. Предоставя се информация за броят поръчани продукти, както и индивидуалната им цена, така че потребителят да може да разгледа предишните си разходи.

На Фигура 36 е показан екранът за информация за конкретна поръчка.



**ФИГУРА 36** ЕКРАН ЗА ИНФОРМАЦИЯ ЗА ПОРЪЧКА

## **V. РЪКОВОДСТВО ЗА ИЗПОЛЗВАНЕ И ПРИМЕРИ ЗА УПОТРЕБА.**

### **5.1 Използване на системата от клиент**

Когато даден клиент желае да използва нашата система, то той трябва първо да изтегли приложението. Това се случва от магазина за приложения, където приложението ще бъде достъпно.

#### **5.1.1 Създаване на регистрация**

Преди да може да използва приложението, потребителят първо трябва да направи своята регистрация. За да се случи това, той трябва да зареди приложението.

След като приложението е стартирано, потребителят ще бъде посрещнат от екрана за вход. Там той трябва да натисне върху бутон „Регистрация на нов потребител“. След това приложението ще го пренасочи към екранът за регистрация. Там потребителят ще трябва да въведе своите данни, като потребителско име, e-mail, парола и имена. След въвеждане той трябва да използва бутон „Регистрирай се“, за да създаде своята регистрация.

При успешна регистрация, потребителят ще бъде върнат обратно на страницата за вход, където ще бъде посрещнат от съобщение, потвърждаващо регистрацията.

При неуспешна регистрация, потребителят ще остане на същата страница и ще получи съобщение с причината за неуспешната регистрация.

#### **5.1.2 Влизане в системата**

Когато приложението е отворено от потребител с вече съществуваща регистрация или от потребител с току що създадена регистрация, то този потребител трябва да използва страницата за вход за да достъпи същинската част на приложението.

За да влезе потребителят е необходимо просто да попълни потребителското си име и паролата в полетата на страницата и да натисне „Вход“

#### **5.1.3 Разглеждане на менюто и добавяне в кошницата**

След като потребителят вече е влязъл в системата, той има достъп до менюто. Менюто представлява списък от предлаганите от ресторанта храни, които са групирани в категории. Потребителят избира категория и получава списъка от предлагани ястия. Потребителят разглежда ястията и когато избере желаното ястие, използва „+“ бутона за да добави желаният артикул в количката. Потребителят може да повтори това многократно, ако иска да добави други артикули към кошницата си.



#### **5.1.4 Създаване на поръчка**

След като потребителят е избрал какво иска да поръча, той трябва да използва навигационната лента в долния край на екрана, за да завърши поръчката си. Това се случва чрез избор на бутон „Кошница“, който го отвежда на екрана с избраните от него продукти.

На този екран, потребителят вижда само продукти, които той изрично е добавил в кошницата си. Те са групирани според своя тип, като потребителят може да модифицира броя на поръчания продукт.

След като потребителят е готов с избора на броя на продукти, той може да види общата стойност на поръчката си, преди да я създаде. Тази калкулация е представена в долния край на екрана, точно над бутона за създаване на поръчка.

Когато потребителят реши да създаде своята поръчка, той може да използва бутона „Подай поръчка“. При натискането на бутона, потребителят ще види допълнителна умалена страница в долния край на екрана, където ще види отново общата сума на поръчката и текст, където ще бъде попълнено местоположението на потребителя, ако приложението е успяло да го определи. Ако не е, потребителят ще бъде помолен да въведе своята локация. Точно под тази локация се намира бутонът „Плати сега“, при чието натискане поръчката се изпраща към ресторанта.

#### **5.1.5 Проследяване на поръчка**

След като е създадена поръчката, потребителят може да следи нейния статус. За да се случи това, той трябва да навигира до меню „Потребител“, използвайки долната навигационна лента.

Когато отвори страницата, той ще види списък с всички поръчки от създаването на регистрацията му до сега. В този списък, най-отгоре ще бъде представена последната създадена поръчка. Всяка поръчка на този екран показва базова информация за поръчката, като нейния статус, платената сума, датата на създаване и нейният идентификатор.

Ако потребителят иска да види допълнителна информация за която и да е от старите си поръчки, той трябва да избере поръчка, чрез докосване. Ще бъде зареден екран, където се визуализират допълнителни данни за поръчката, като всички продукти, които са част от нея.

### **5.2 Използване на системата от персонал**

Персоналът на ресторанта трябва да използва системата, за да обработва постъпващите от клиентите поръчки. Това се случва чрез уеб приложение, което може да бъде достъпно само от специално добавени от администратор потребители.

Когато член на персонала се опита да отвори страницата за обработка на поръчки, той ще бъде пренасочен към страница за вход, където ще трябва да въведе своите данни.

След като е упълномощен да влезе в системата, членът на персонала няма да бъде питан за входни данни отново, поне до следващото рестартиране на браузъра.

След упълномощаването от системата за вход, членът на персонала ще бъде пренасочен към страницата за управление на поръчки. Там той може да види всички поръчки за деня, като те ще бъдат групирани според статуса им. Така, той може да избере поръчка, по която да работи, и когато е готов да я придвижи към следващата стъпка от приготвянето. Важно е да се отбележи, че поръчки могат да бъдат придвижвани в различен статус, само ако са от текущия ден. Поръчки от предходни дни могат да бъдат разглеждани, но не и променяни.

За преглеждане на архивирани дни, членът на персонала може да използва филтъра от дясната страна на екрана. При избиране на филтър менюто се отваря календар, който показва датите, в които има поръчки. Дати без поръчки са оцветени в сиво и не могат да бъдат избирани като филтър. След като е избрана дата е необходимо да бъде натиснат бутон „Filter“, който ще презареди страницата.

## VI. ЗАКЛЮЧЕНИЕ

В настоящата дипломна работа беше разработена и внедрена система за създаване и обработка на поръчки в ресторант, която автоматизира ключови процеси и улеснява работата на персонала и клиентите. В хода на разработката бяха анализирани съществуващите решения, дефинирани бяха изискванията към системата и беше реализирана цялостна архитектура, включваща Backend, Frontend, интеграция с база данни, мобилно приложение и BLE маяци за идентификация на маси.

Системата предоставя лесен за използване интерфейс за клиентите, който им позволява да разглеждат менюто, да правят поръчки и да проследяват статуса на своите заявки. От друга страна, персоналът на ресторанта има достъп до уеб приложение, което му позволява да управлява поръчките, да ги придвижва през различните етапи на обработка и да следи общата сума на приходите за деня.

Системата е изградена с използването на съвременни технологии и инструменти, като Django за Backend разработка, Android Studio за мобилното приложение и ESP32 чипове за BLE маяците. Това осигурява висока производителност, сигурност и възможност за лесно разширение на функционалността в бъдеще.

Важна част от реализираната система са BLE (Bluetooth Low Energy) маяците, които осигуряват автоматична идентификация на клиентите и асоцииране на поръчките с конкретна маса в ресторанта. Всеки маяк, базиран на ESP32 микроконтролер, излъчва уникален идентификатор чрез BLE сигнал, съответстващ на номера на масата. Мобилното приложение сканира за налични BLE сигнали и определя най-близкия маяк по сила на сигнала (RSSI), което гарантира, че поръчки за маса могат да се правят само от клиенти, които физически присъстват в ресторанта. Това елиминира възможността за злоупотреби с фалшиви поръчки извън обекта. Използването на BLE технологията допринася за ниска консумация на енергия, надеждна работа на къси разстояния и лесна интеграция със съвременни мобилни устройства. Маяците работят автономно и не изискват сложна поддръжка, което ги прави подходящи за реална ресторантска среда.

Системата е проектирана с акцент върху сигурността на данните, като се използват хеширане и криптиране на пароли, както и HTTPS протокол за комуникация. Това гарантира, че личната информация на потребителите е защитена и че комуникацията между клиентите и сървъра е сигурна.

В заключение, разработената система представлява цялостно решение за управление на поръчки в ресторант, което може да бъде адаптирано и разширено според нуждите на конкретния бизнес. Тя демонстрира как съвременните технологии могат да бъдат използвани за оптимизиране на процесите в ресторантьорството и подобряване на клиентското изживяване.

## СЪКРАЩЕНИЯ

API - Application Programming Interface .....	9
AWS - Amazon Web Services .....	11
BLE - Bluetooth Low Energy.....	10
BYO - Bring Your Own .....	11
CRUD - Create, Read, Update, and Delete .....	30
DRF - Django REST Framework .....	9
JSON - JavaScript Object Notation.....	24
MVVM Model View ViewModel .....	39
ORM - Object Relation Mapper .....	30
REST - Representational State Transfer.....	9
RSSI - Received Signal Strength Indicator .....	19

## 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/>.
- [11] Composables, „Android Distribution Chart,“ [Онлайн]. Available: <https://composables.com/android-distribution-chart>.
- [12] Django, „Getting started with Django,“ [Онлайн]. Available: <https://www.djangoproject.com/start/>.
- [13] D. Spectacular, „Schema customization,“ [Онлайн]. Available: <https://drf-spectacular.readthedocs.io/en/latest/customization.html>.
- [14] O. Tools, „OpenAPI Generator Source,“ [Онлайн]. Available: <https://github.com/OpenAPITools/openapi-generator>.

- [15] S. Gawande, „Creating Views and Templates in Django,“ [Онлайн]. Available: <https://medium.com/django-unleashed/5-creating-views-and-templates-in-django-d43c4f590009>.
- [16] Dagger, „Dagger home,“ [Онлайн]. Available: <https://dagger.dev/>.
- [17] Android, „Getting started with Jetpack,“ [Онлайн]. Available: <https://developer.android.com/jetpack>.
- [18] Android, „Getting started with Jetpack Compose,“ [Онлайн]. Available: <https://developer.android.com/compose>.
- [19] „Mate Kitchen,“ [Онлайн]. Available: <https://matekitchen.com/>.

## ПРИЛОЖЕНИЯ

Изглед за визуализация на поръчки в уеб приложението. Позволява филтриране по дата. Преди показване, изгледът групира данните по статус, както и пресмята общата сума за деня.

```
@login_required
def orders_list(request):
    # Get distinct dates with orders
    available_dates = list(
        Order.objects.annotate(order_day=TruncDate('order_date'))
        .values_list('order_day', flat=True)
        .distinct()
    )

    selected_date = request.GET.get('date')
    filter_date = parse_date(selected_date) if selected_date else date.today()

    orders = Order.objects.select_related(
        'user'
    ).prefetch_related(
        'items'
    ).filter(order_date__date=filter_date)

    grouped_orders = {
        'Изчакване': orders.filter(status='Изчакване'),
        'Приготвяне': orders.filter(status='Приготвяне'),
        'Доставяне': orders.filter(status='Доставяне'),
        'Завършена': orders.filter(status='Завършена'),
    }

    total_money = sum(order.total_amount for order in orders)

    return render(
        request,
        'frontend/orders_list.html',
        {
            'grouped_orders': grouped_orders,
            'STATUS_FLOW': STATUS_FLOW,
            'selected_date': filter_date,
            'today': date.today(),
            'available_dates': [d.isoformat() for d in available_dates],
            'total_amount_for_day': total_money
        }
    )
```

Изглед за работа с потребители. Включва метод за регистрация, както и метод за вход в системата

```
class UserViewSet(viewsets.ViewSet):

    @extend_schema(
        request=UserRegistrationSerializer,
        responses=UserRegistrationResponseSerializer
    )
    @action(detail=False, methods=['post'])
    def register(self, request):
        serializer = UserRegistrationSerializer(data=request.data)
        if serializer.is_valid():
            user = serializer.save()
            return Response({
                'message': 'User created successfully',
                'username': user.username,
                'first_name': user.first_name,
                'last_name': user.last_name,
            }, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    @extend_schema(
        request=UserLoginSerializer,
        responses=UserLoginResponseSerializer
    )
    @action(detail=False, methods=['post'])
    def login(self, request):
        serializer = UserLoginSerializer(data=request.data)
        if serializer.is_valid():
            user = serializer.validated_data
            return Response(
                {
                    'user_id': user.id,
                    'username': user.username,
                    'first_name': user.first_name,
                    'last_name': user.last_name,
                },
                status=status.HTTP_202_ACCEPTED
            )
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```



Функционалност за Bluetooth Low Energy сканиране, част от View Model на страницата. Използва съществуваща имплементация на BLE Manager клас.

```
@HiltViewModel
class BLEViewModel @Inject constructor(
    private val bleManager: BLEDataReceiveManager
) : ViewModel() {

    private val _devices = MutableStateFlow<List<DeviceInfo>>(emptyList())
    val devices: StateFlow<List<DeviceInfo>> = _devices

    private val _topDevice = MutableStateFlow<DeviceInfo?>(null)
    val topDevice: StateFlow<DeviceInfo?> = _topDevice

    private val _message = MutableStateFlow<String>("")
    val message: StateFlow<String> = _message

    private fun collectScanResults() {
        viewModelScope.launch {
            bleManager.data.collectLatest { resource ->
                when (resource) {
                    is Resource.Success -> {
                        _devices.value = resource.data.devices
                        _topDevice.value = resource.data.topDevice
                    }
                    is Resource.Loading -> {
                        _message.value = resource.message ?: ""
                    }
                    is Resource.Error -> {
                        _message.value = resource.errorMessage
                    }
                }
            }
        }
    }

    fun scan() {
        collectScanResults()
        bleManager.discover() // Start scanning
    }
}
```