  
**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](#_Toc200216703)

[II. Постановка на дипломната работа, цели и задачи. 8](#_Toc200216704)

[4.1 Постановка 8](#_Toc200216705)

[4.2 Цели и задачи 8](#_Toc200216706)

[4.3 Ограничения 8](#_Toc200216707)

[4.4 Използвани технологии 9](#_Toc200216708)

[4.4.1 Python 9](#_Toc200216709)

[4.4.2 Django 9](#_Toc200216710)

[4.4.3 Django REST Framework 9](#_Toc200216711)

[4.4.4 Kotlin 9](#_Toc200216712)

[4.4.5 Android 10](#_Toc200216713)

[4.4.6 Bluetooth 10](#_Toc200216714)

[4.4.7 ESP32 10](#_Toc200216715)

[4.4.8 Docker 11](#_Toc200216716)

[4.4.9 Appliku 11](#_Toc200216717)

[4.4.10 Amazon Web Services 11](#_Toc200216718)

[III. Функционално описание 12](#_Toc200216719)

[5.1 Общ преглед на системата 12](#_Toc200216720)

[5.2 Django сървър 13](#_Toc200216721)

[5.3 Клиентско приложение 15](#_Toc200216722)

[5.4 Уеб приложение 18](#_Toc200216723)

[5.5 Маяци 18](#_Toc200216724)

[IV. Програмна реализация. 20](#_Toc200216725)

[4.1 Реализация на Django сървър 20](#_Toc200216726)

[4.1.1 Модели 20](#_Toc200216727)

[4.1.2 Входни точки с OpenAPI спецификация 22](#_Toc200216728)

[4.1.3 Изгледи на приложението 29](#_Toc200216729)

[4.2 Реализация на мобилно приложение 37](#_Toc200216730)

[V. Ръководство за използване и примери за употреба.  38](#_Toc200216731)

[VI. Заключение 39](#_Toc200216732)

[Съкращения 40](#_Toc200216733)

[Източници 41](#_Toc200216734)

**Списък с фигури**

[Фигура 1 Общ преглед на системата 12](#_Toc200216387)

[Фигура 2 Административен панел за елементи от менюто 13](#_Toc200216388)

[Фигура 3 Примерен Swagger документ 14](#_Toc200216389)

[Фигура 4 Elastic Cloud сървър в AWS 14](#_Toc200216390)

[Фигура 5 Приложение в Appliku 15](#_Toc200216391)

[Фигура 6 Разпределение на мобилни устройства според версията им на Android 16](#_Toc200216392)

[Фигура 7 Структура на екраните на приложението 17](#_Toc200216393)

[Фигура 8 Идентификация на маяк 19](#_Toc200216394)

[Фигура 9 Клас диаграма на моделите 21](#_Toc200216395)

[Фигура 10 Entity Relations диаграма на базата данни 22](#_Toc200216396)

[Фигура 11 Входна точка за получаване на всички елементи от менюто 23](#_Toc200216397)

[Фигура 12 Входна точка за получаване на елемент по неговото ID 24](#_Toc200216398)

[Фигура 13 Входна точка за вземане на категориите на елементите от менюто 24](#_Toc200216399)

[Фигура 14 Входна точка за получаване на всички елементи от дадена категория 25](#_Toc200216400)

[Фигура 15 Входна точка за създаване на поръчка 26](#_Toc200216401)

[Фигура 16 Входна точка за получаване на всички поръчки за даден потребител 27](#_Toc200216402)

[Фигура 17 Входна точка за подаване на входни данни за потребител 28](#_Toc200216403)

[Фигура 18 Входна точка за създаване на нов потребител 28](#_Toc200216404)

[Фигура 19 Входна точка за получаване на спецификация 29](#_Toc200216405)

[Фигура 20 Поток на работа за метода get\_all\_categories 30](#_Toc200216406)

[Фигура 21 Поток на работа на метода items\_by\_category 31](#_Toc200216407)

[Фигура 22 Поток на работа на метода get\_all\_items 32](#_Toc200216408)

[Фигура 23 Поток на метода get\_item\_by\_id 33](#_Toc200216409)

[Фигура 24 Поток на работа на метода orders\_by\_user 34](#_Toc200216410)

[Фигура 25 Поток на изпълнение на метода create\_order 35](#_Toc200216411)

[Фигура 26 Поток на метода register 36](#_Toc200216412)

[Фигура 27 Поткок на метода login 37](#_Toc200216413)

# 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.

## 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 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.

### 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 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 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].

### 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 до устройството, което 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 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.

### 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 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].

### 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].

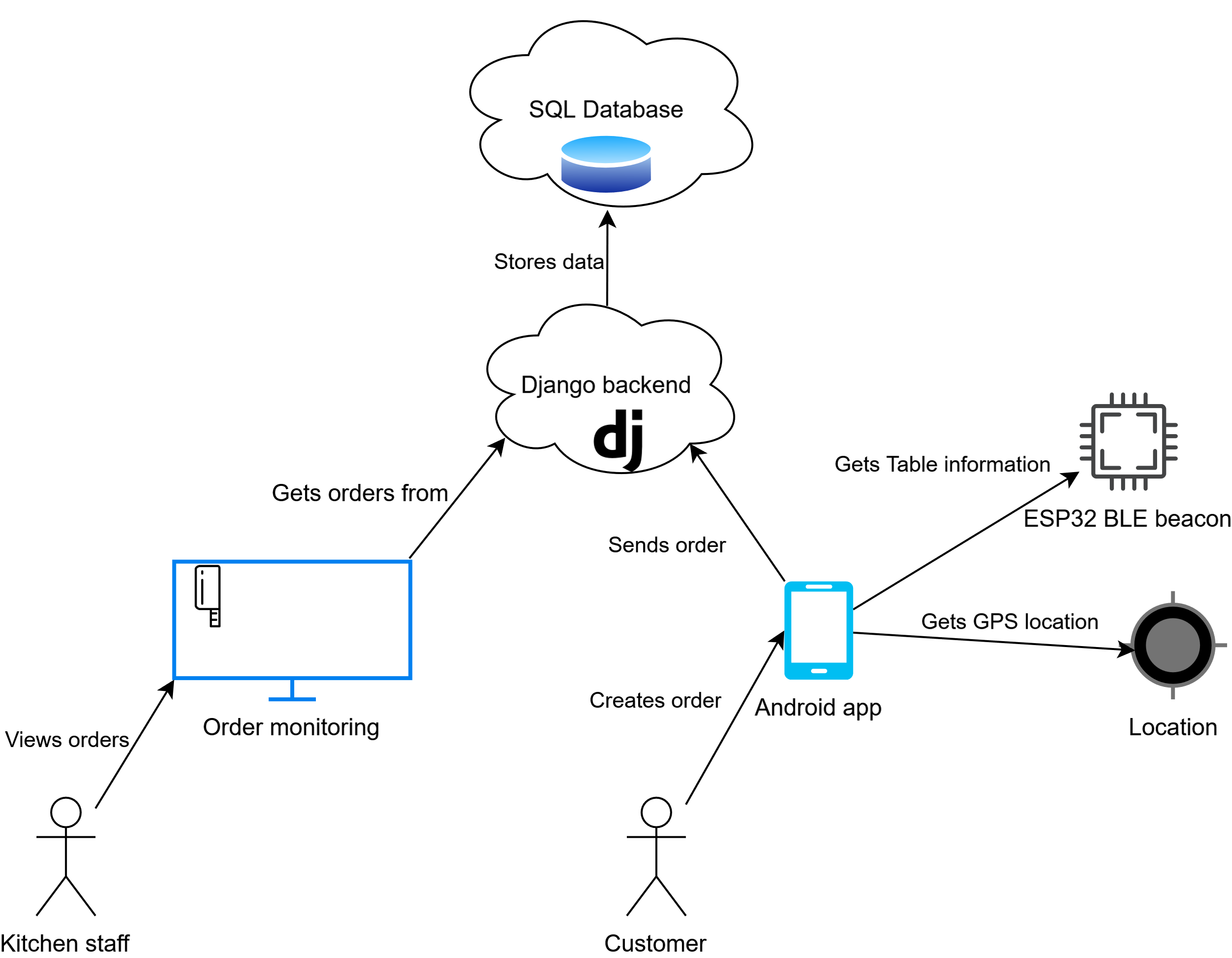
# Функционално описание

## 3.1 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.

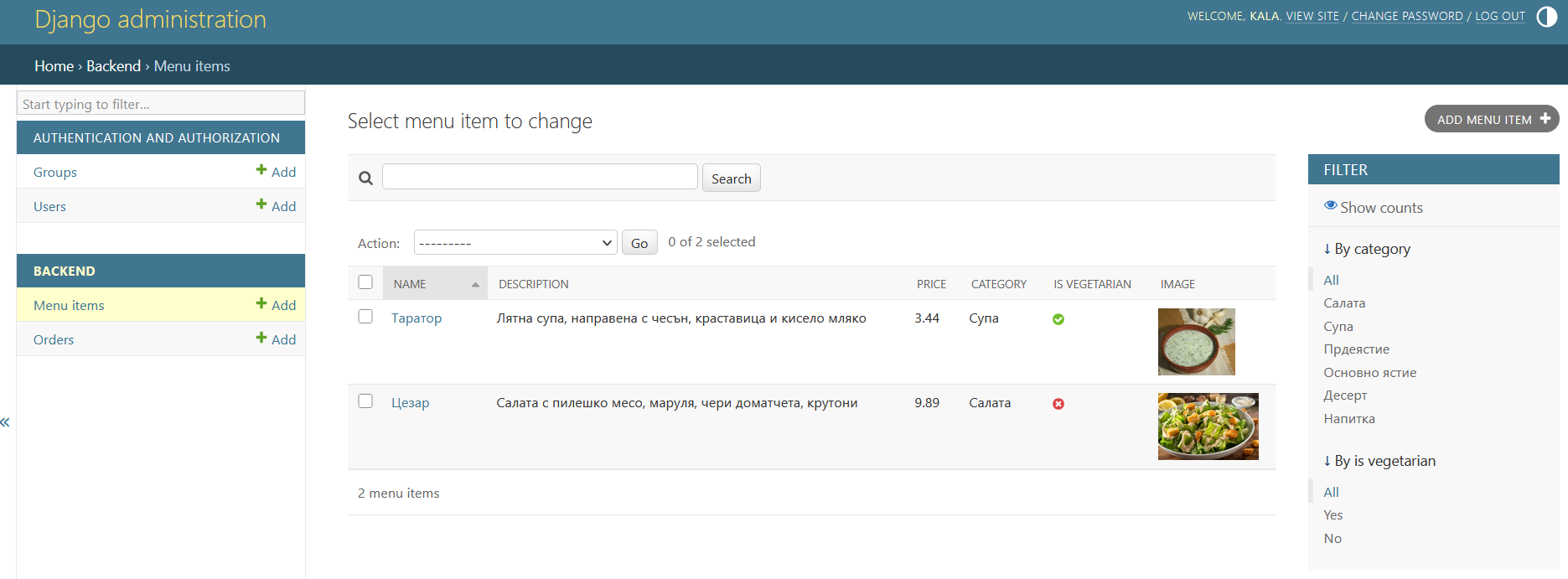


Фигура 1 Общ преглед на системата

## 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.

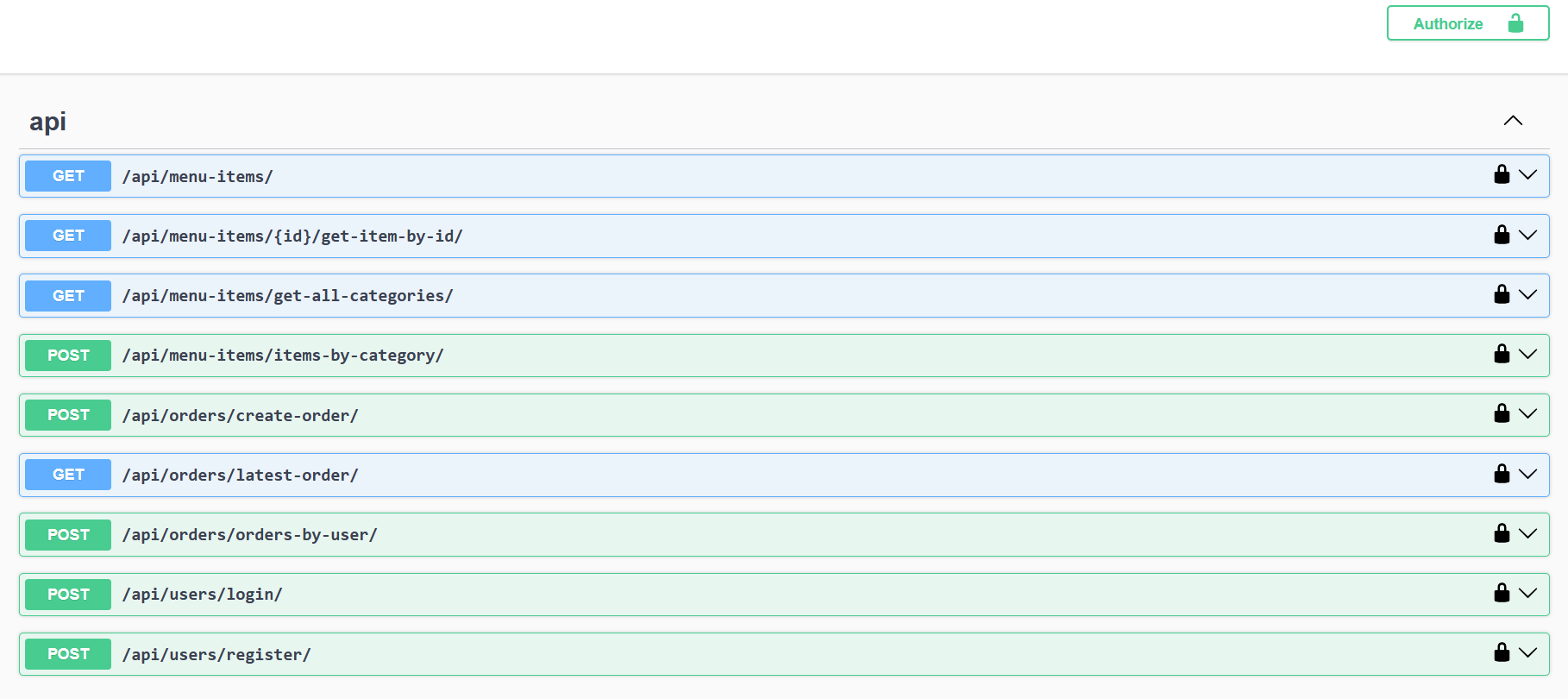
Библиотеката предоставя голямо количество готови функционалности, които могат да бъдат използвани директно, като например управление на потребители. Конфигурацията се случва много лесно, тъй като Django предоставя файл с настройки, в който потребителят може да управлява своят сървър. Освен този файл, 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.

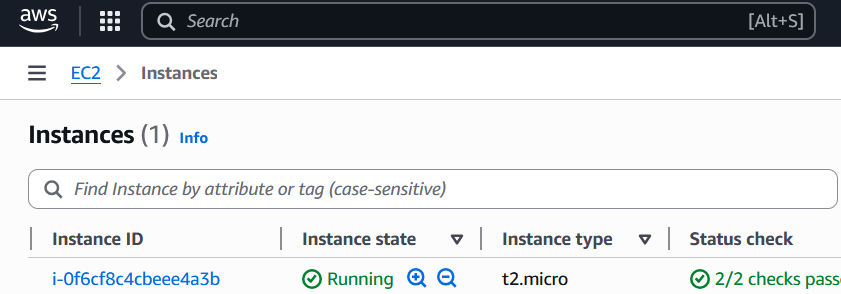
Django REST Framework ни кара да пишем RESTful приложения, за разлика от стандартната Django библиотека, която позволява доста либерално API. Като използваме DRF, ние можем да генерираме Swagger клиенти, които да улеснят клиентски приложения, които използват нашия сървър. На Фигура 3 е показан примерен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

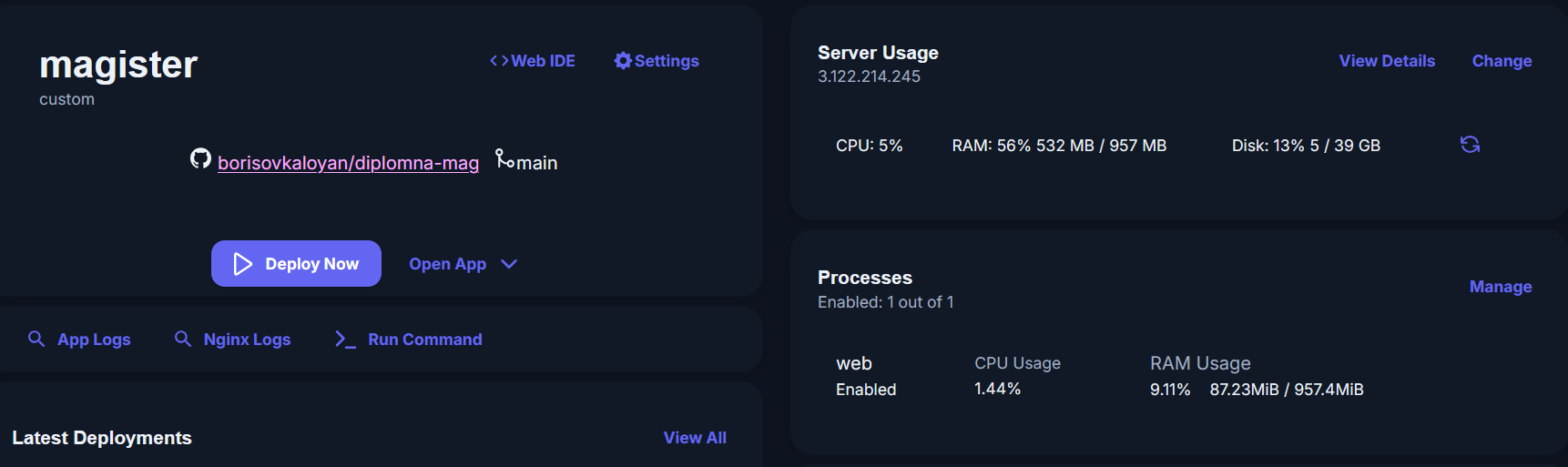
За хостинг платформа ще се използва Amazon Web Services. Тази платформа е една от най-използваните в световен мащаб. Има 24 часова поддръжка и ни гарантира, че нашият сървър ще бъде постоянно достъпен. На Фигура 4 е показан сървърът, върху който ще се изпълнява нашият Django проект.



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.

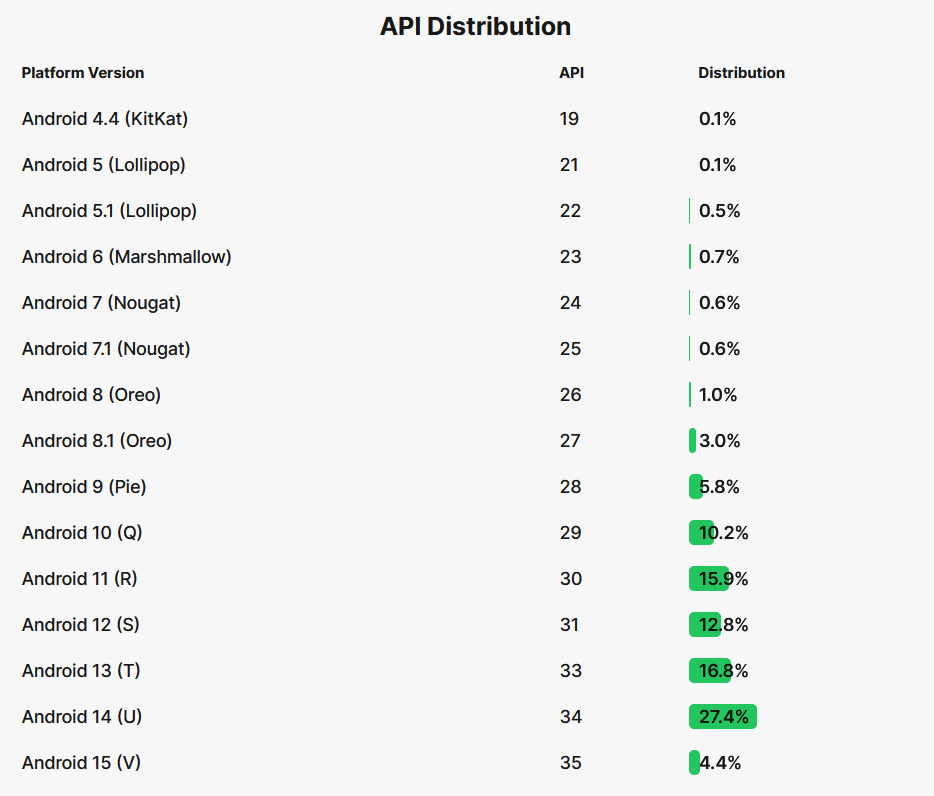
Освен API, Django приложението ще предоставя уеб страница, която да показва актуална информация за всички поръчки, направени за конкретен ден. Тези поръчки ще бъдат групирани по техния статус, което ще улесни изпълнението им в кухнята и последващото им сервиране. Приложението ще пази история на поръчките,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

## 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 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.

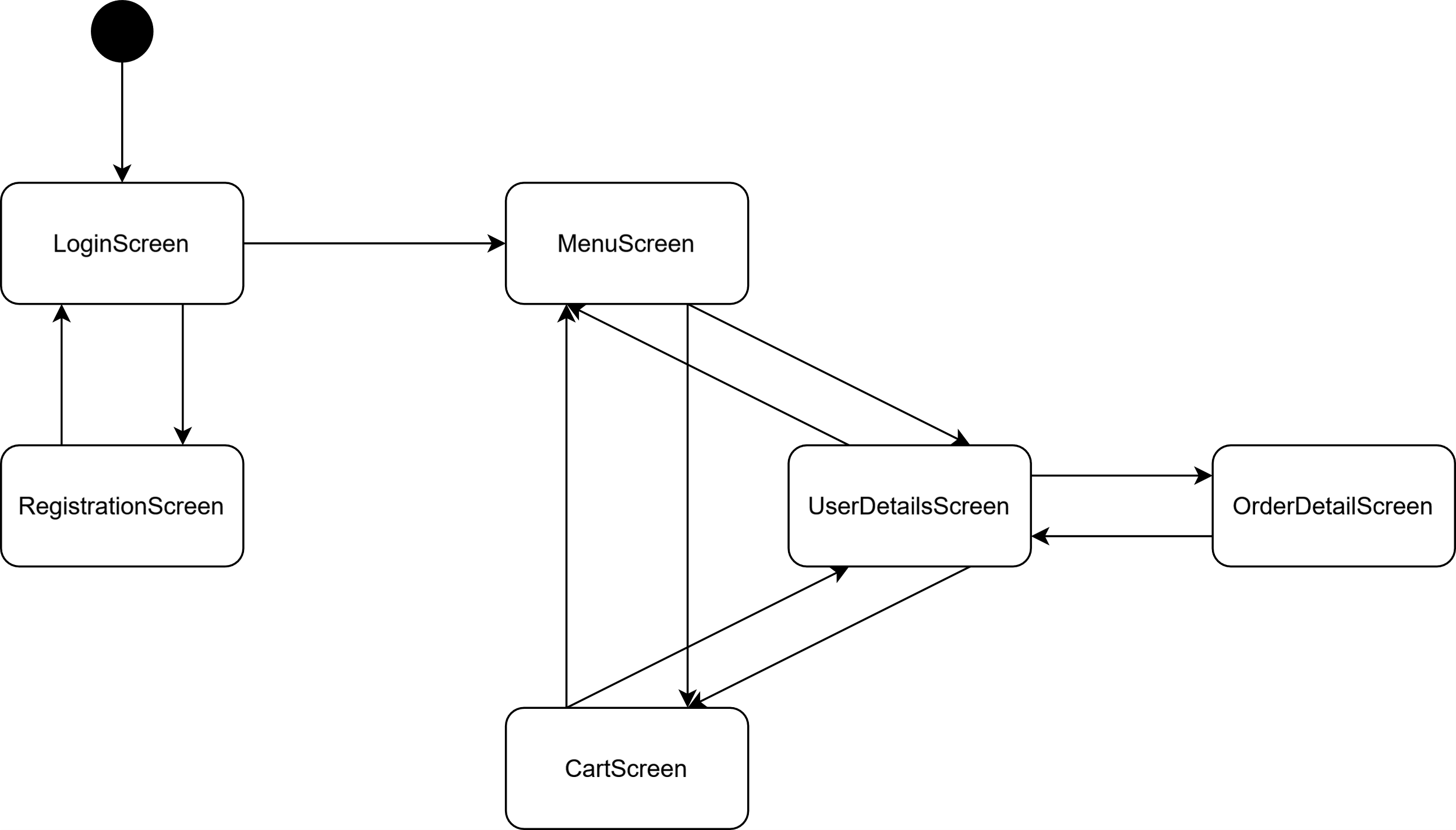
На Фигура 6 е показано разпределението на активните устройства, спрямо версията на операционната им система [11]. Според тази графика, ако изберем версия 26 на API за Android, то около 97% от устройствата на пазара биха могли да изпълняват нашето приложение.



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.

## 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 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.

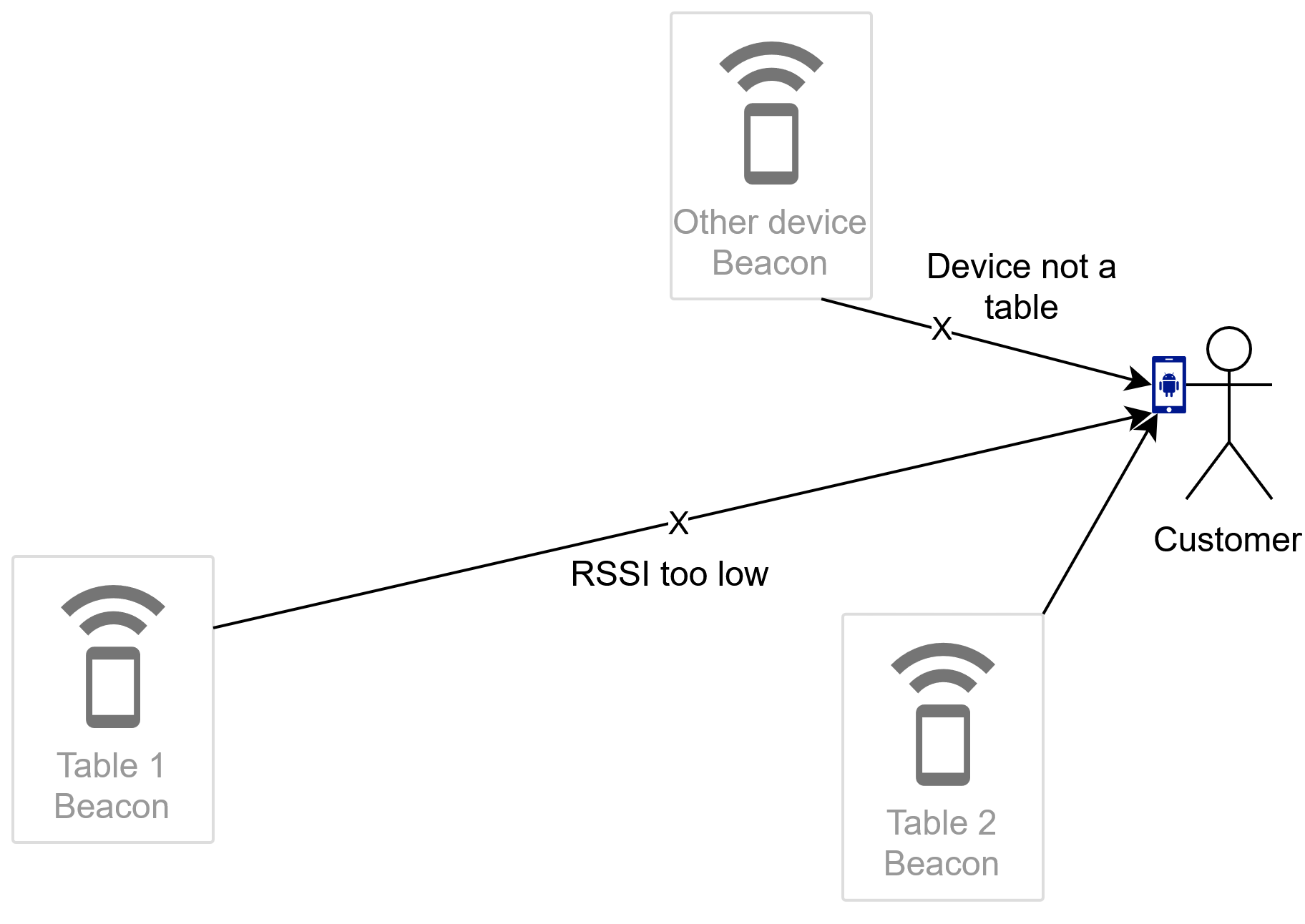
Таблото за управление представлява главният екран на уеб приложението. Там ще могат да бъдат разглеждани поръчките за конкретен ден, като ще бъдат групирани според своя статус, за лесно проследяване на процеса по приготвяне и доставяне. По този начин, процесите в кухнята ще бъдат улеснени и приготвянето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

## 3.5 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.

## 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, документирани чрез OpenAPI
* Изгледи, които съдържат основната функционалност

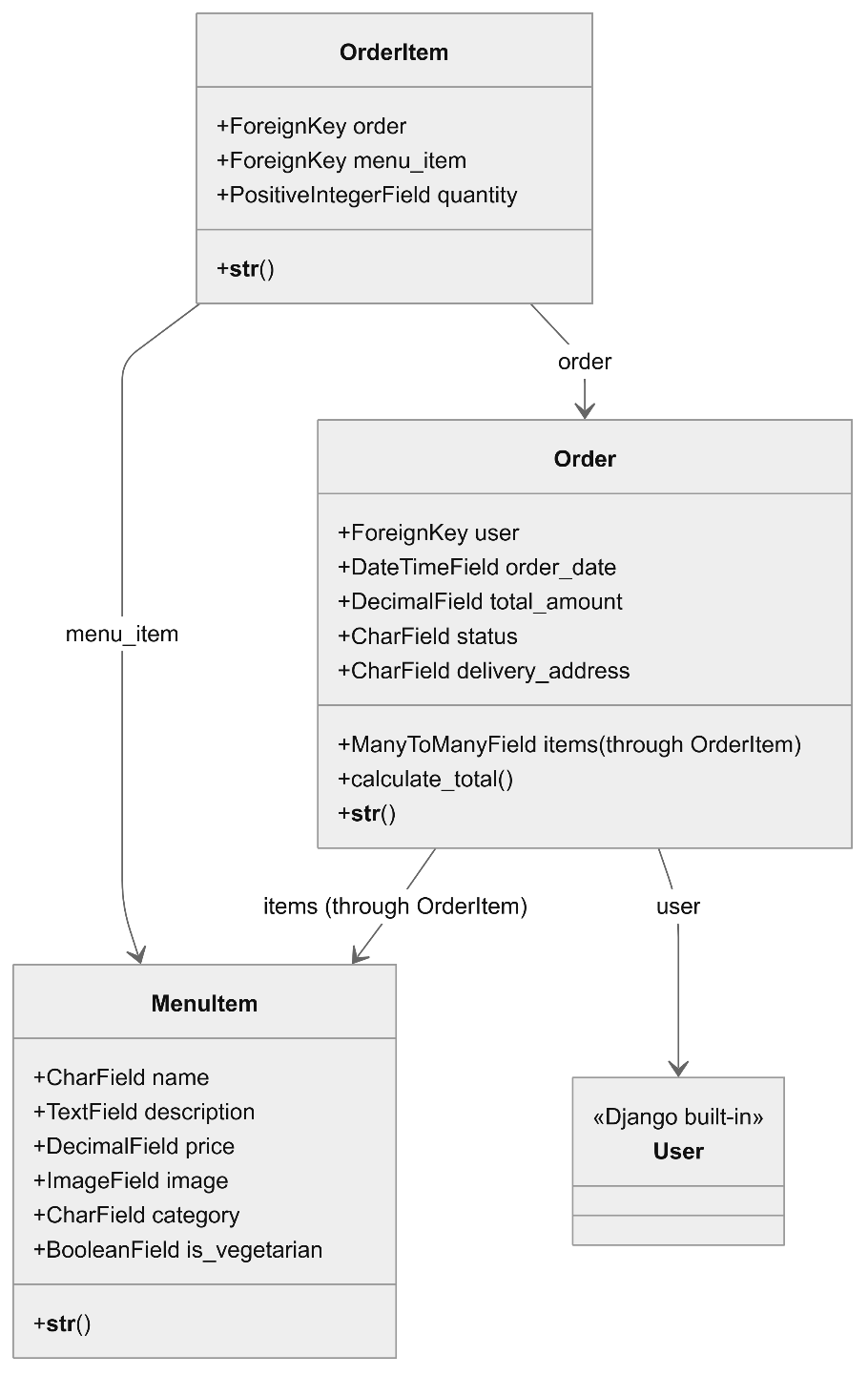
### 4.1.1 Модели

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 да бъде дефинирана колона в база от 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 от 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.

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.

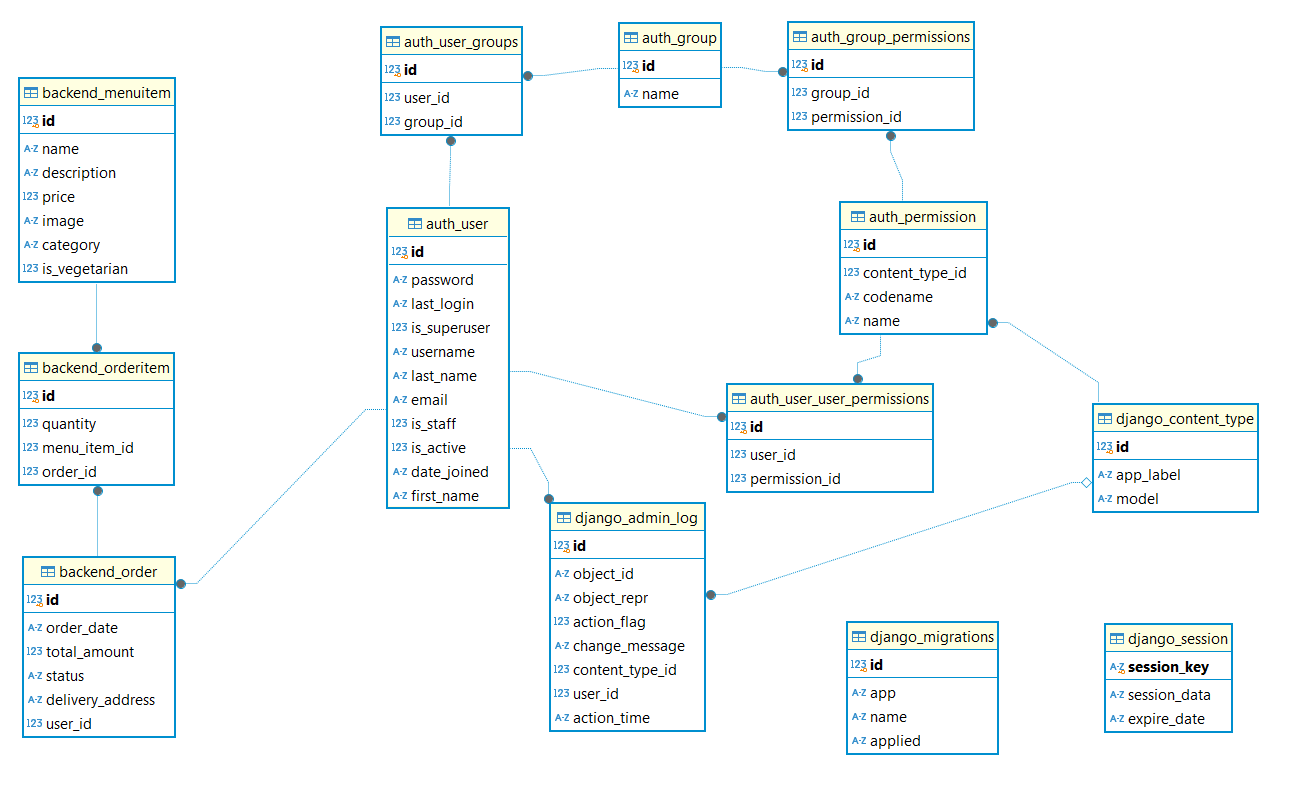


Фигура 9 Клас диаграма на моделите

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 таблицеата auth\_user, която е таблицата, отговорна за потребителският модел на Django. Т

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 фунцкционалност.

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

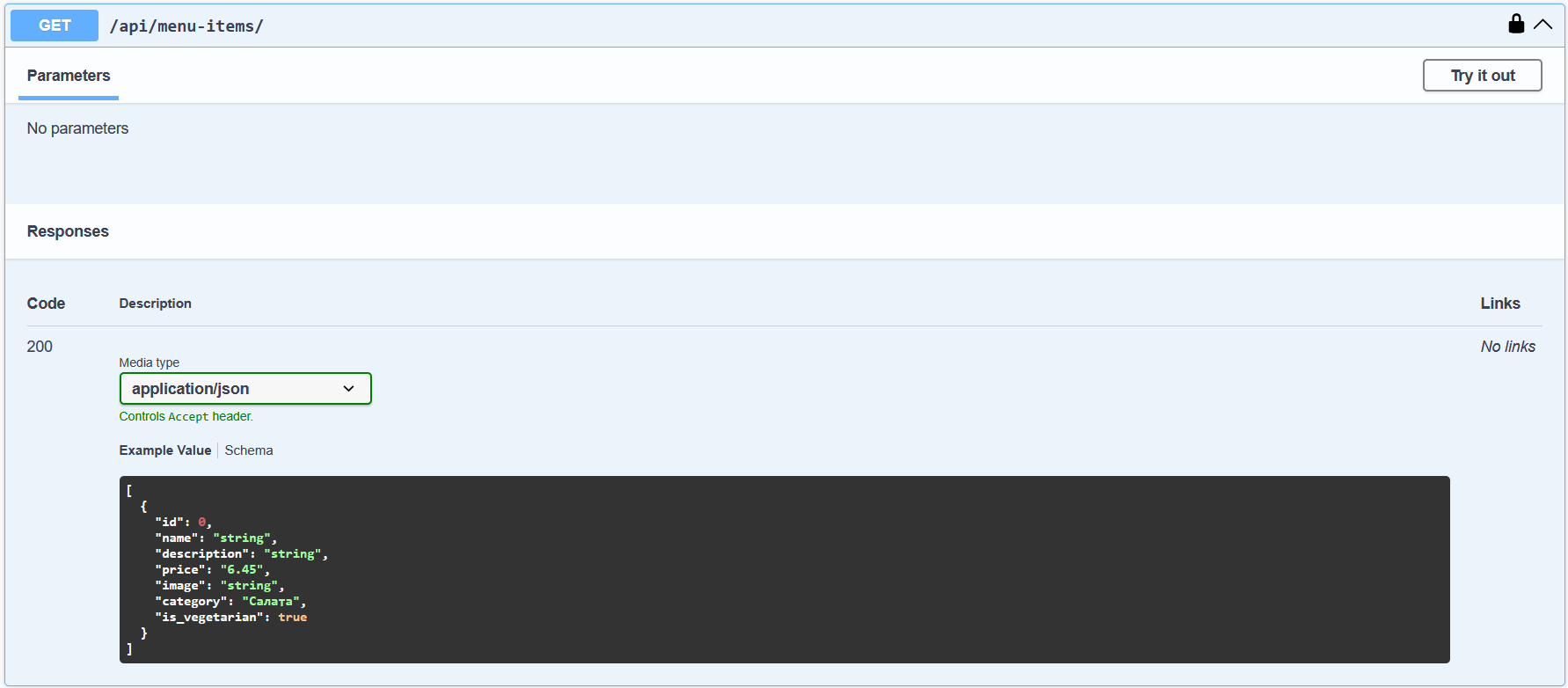
### 4.1.2 Входни точки с OpenAPI спецификация

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 приложенито 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. DRF автоматично. 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, с които не сме доволни.

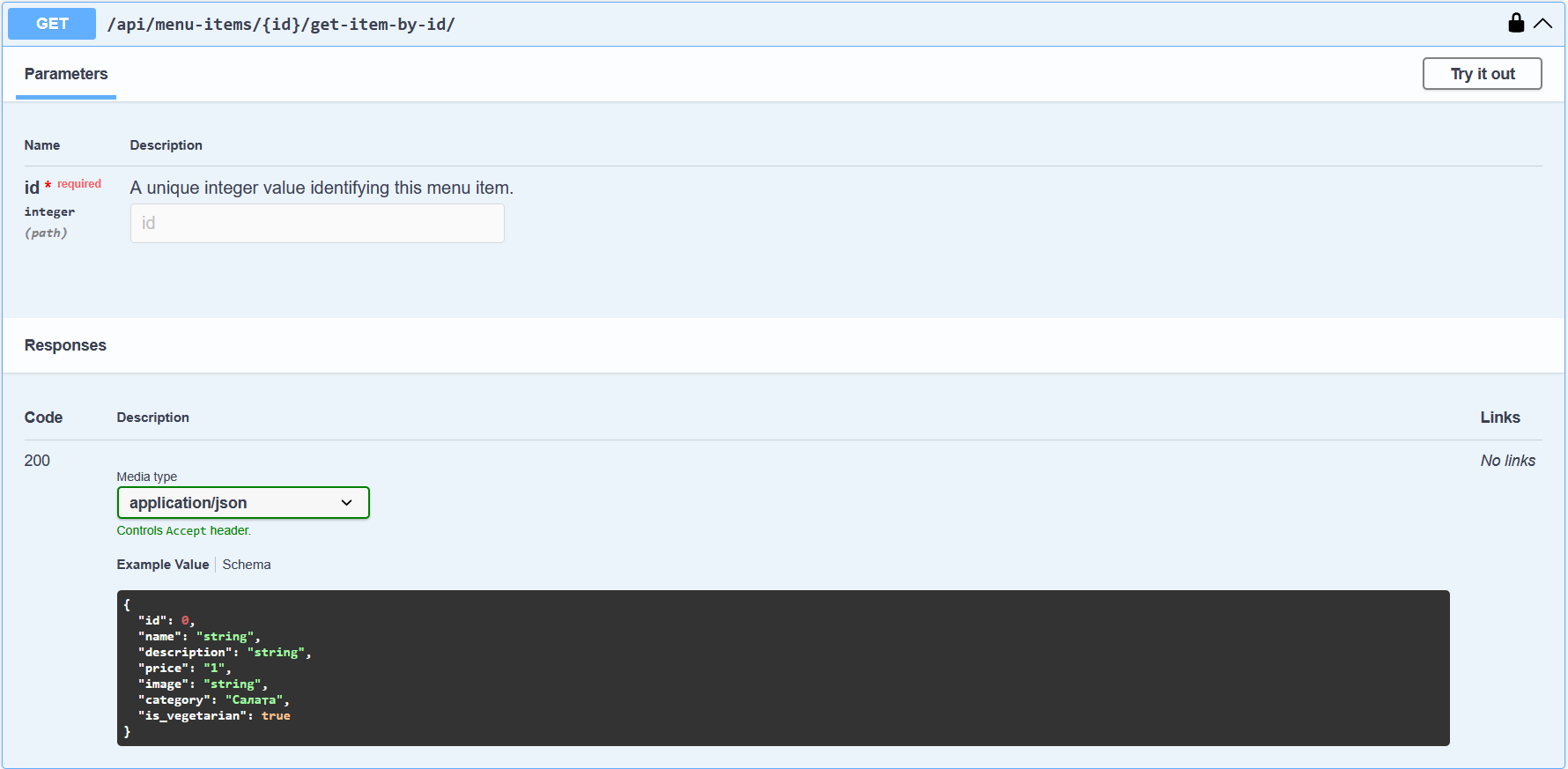
Използвайки DRF, са описани осем входни точки в програмата, които са разделени на няколко подгрупи. Съществуват входни точки в група за работа с потребител, работа с поръчки и работа с елементи от менюто. Всяка категория съдържа една или повече входни точки, които позволяват работа с даден ресурс.

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.



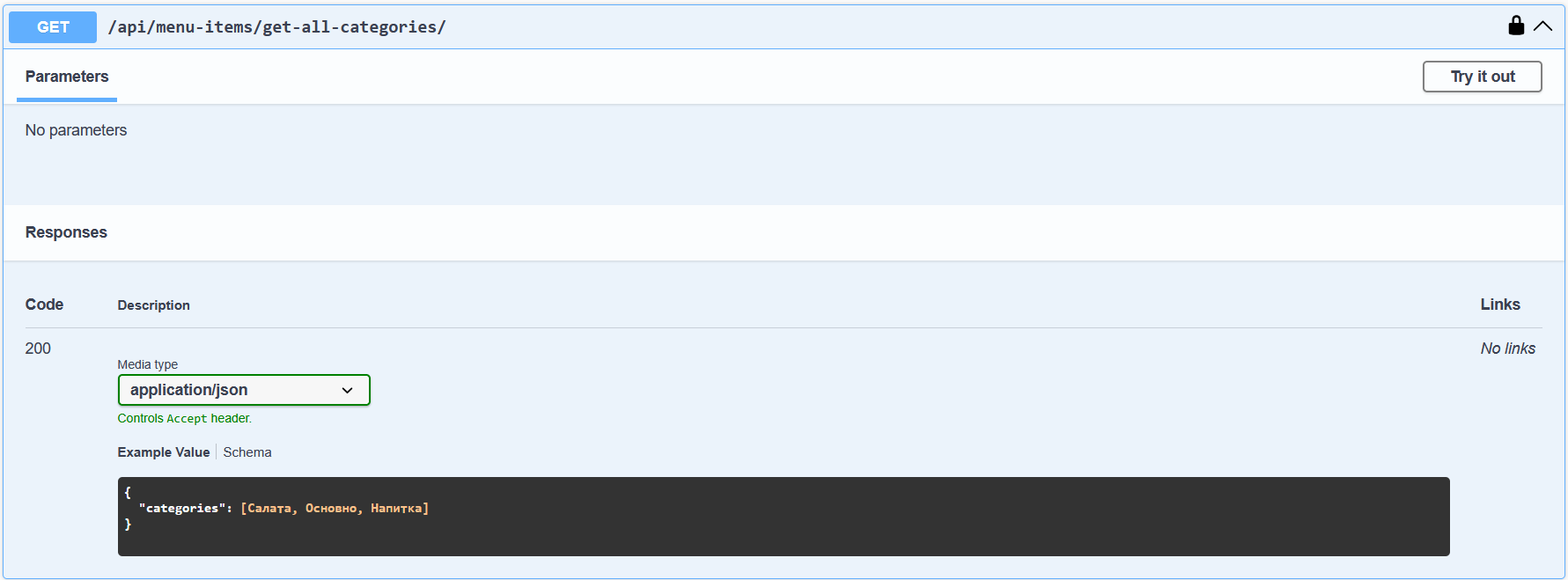
Фигура 11 Входна точка за получаване на всички елементи от менюто

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 индентификационен 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



Фигура 12 Входна точка за получаване на елемент по неговото ID

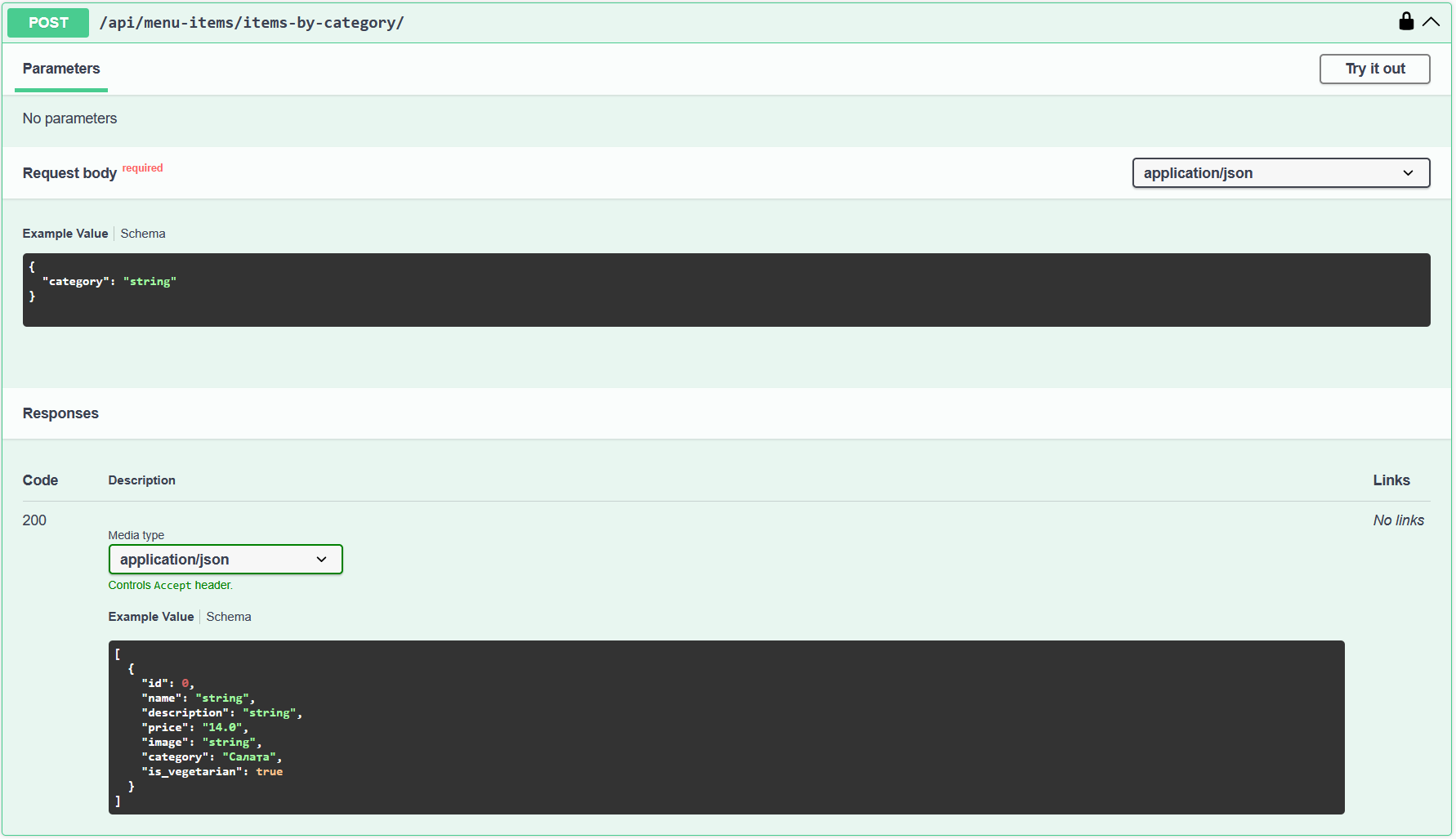
На Фигура 13 е показана още една входна точка от групата menu-items. Тази входна точка предоставя списък от категориите на елементи от менюто. Тази входна точка отново използва HTTP GET. При успешен отговор, връща HTTP\_200\_OK като статус код и JSON обект, състоящ се от ключ categories и списък със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



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.

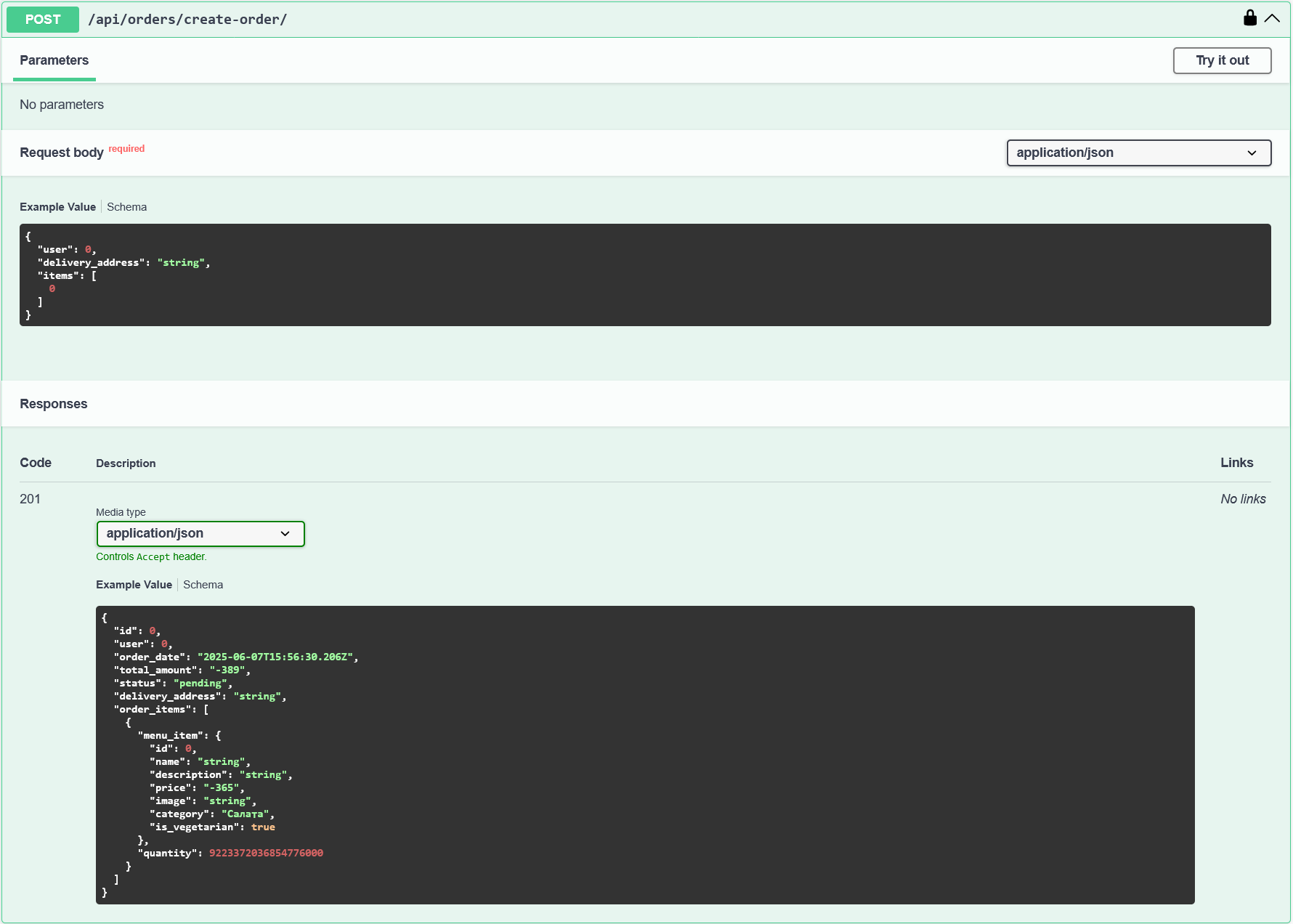


Фигура 14 Входна точка за получаване на всички елементи от дадена категория

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.

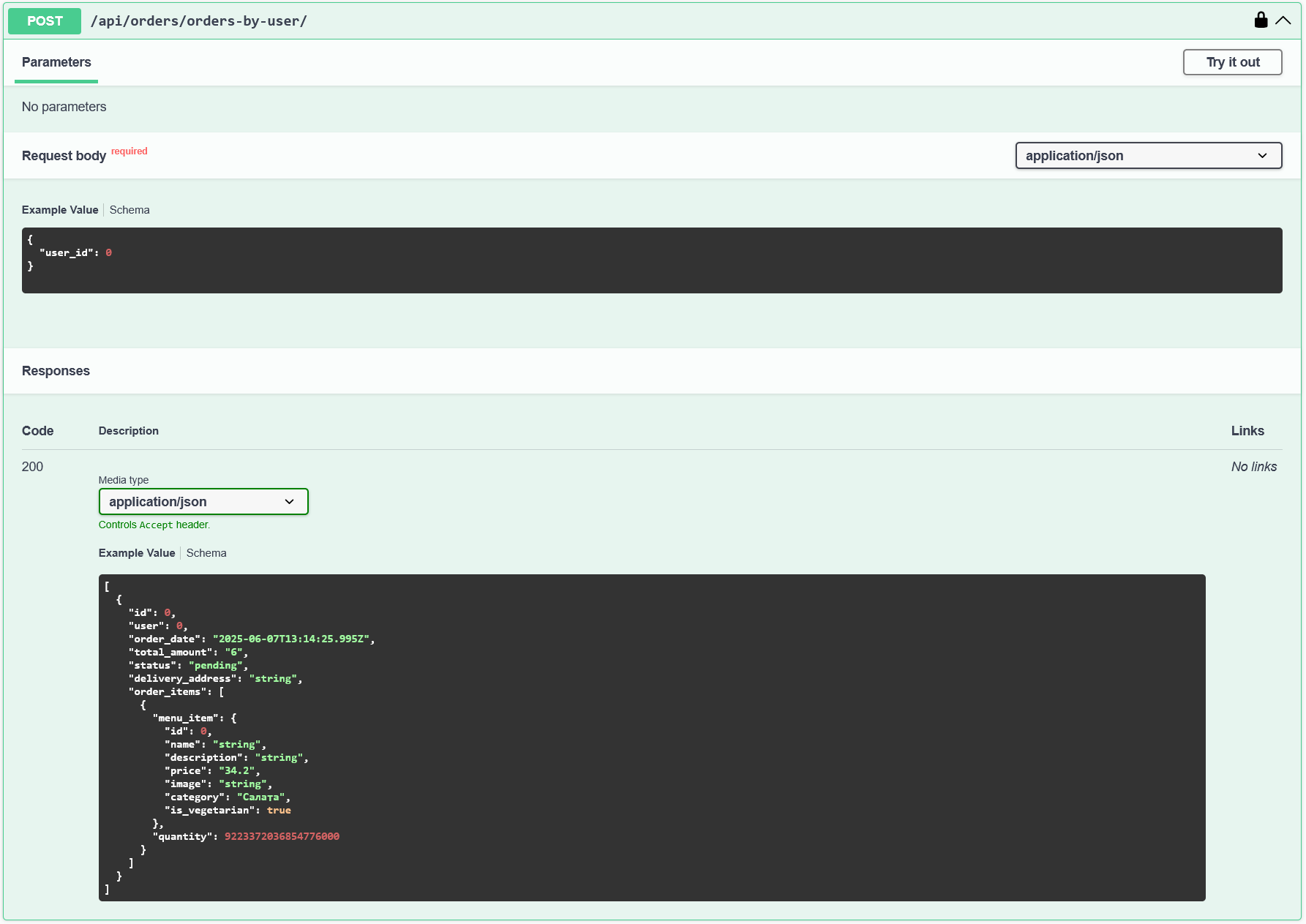
Когато поръчката се обработва, тя трябва да бъде попълнена с останалите данни, необходими за базата данни. В отговорът на системата, потребителят трябва да получи статус код HTTP\_201\_CREATED, както и пълен обект на поръчка, която съдържа елементите от релацията с order\_items и menu\_items. По този 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

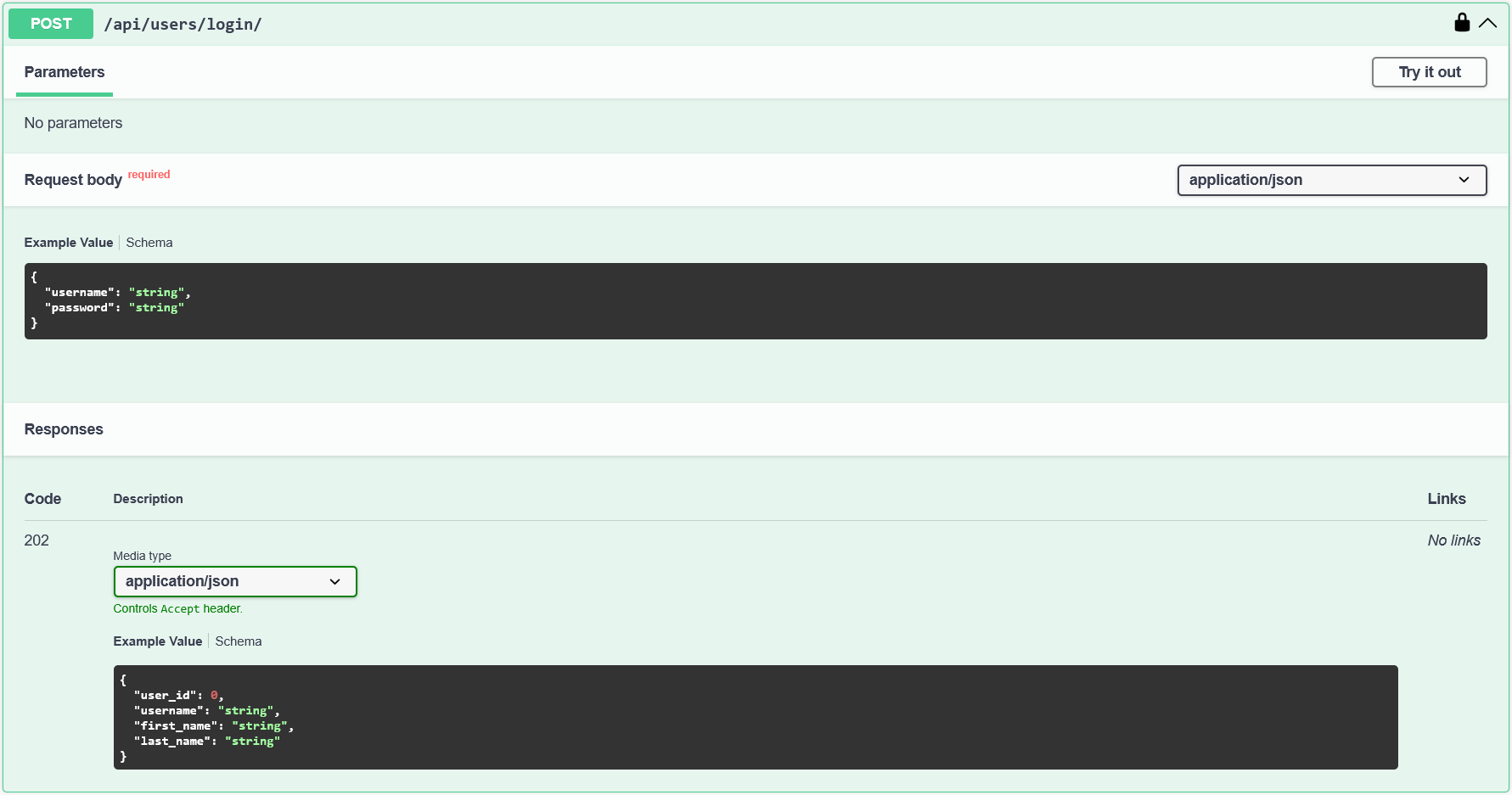
На Фигура 16 е показана входната точка за получаване на всички поръчки, създадени от конкретен потребител. Заявката е последната от групата на orders и е от тип HTTP POST. Имаме задължително тяло на заявката, което трябва да съдържа уникалният идентификатор на потребителя. Спрямо този индикатор, приложението 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

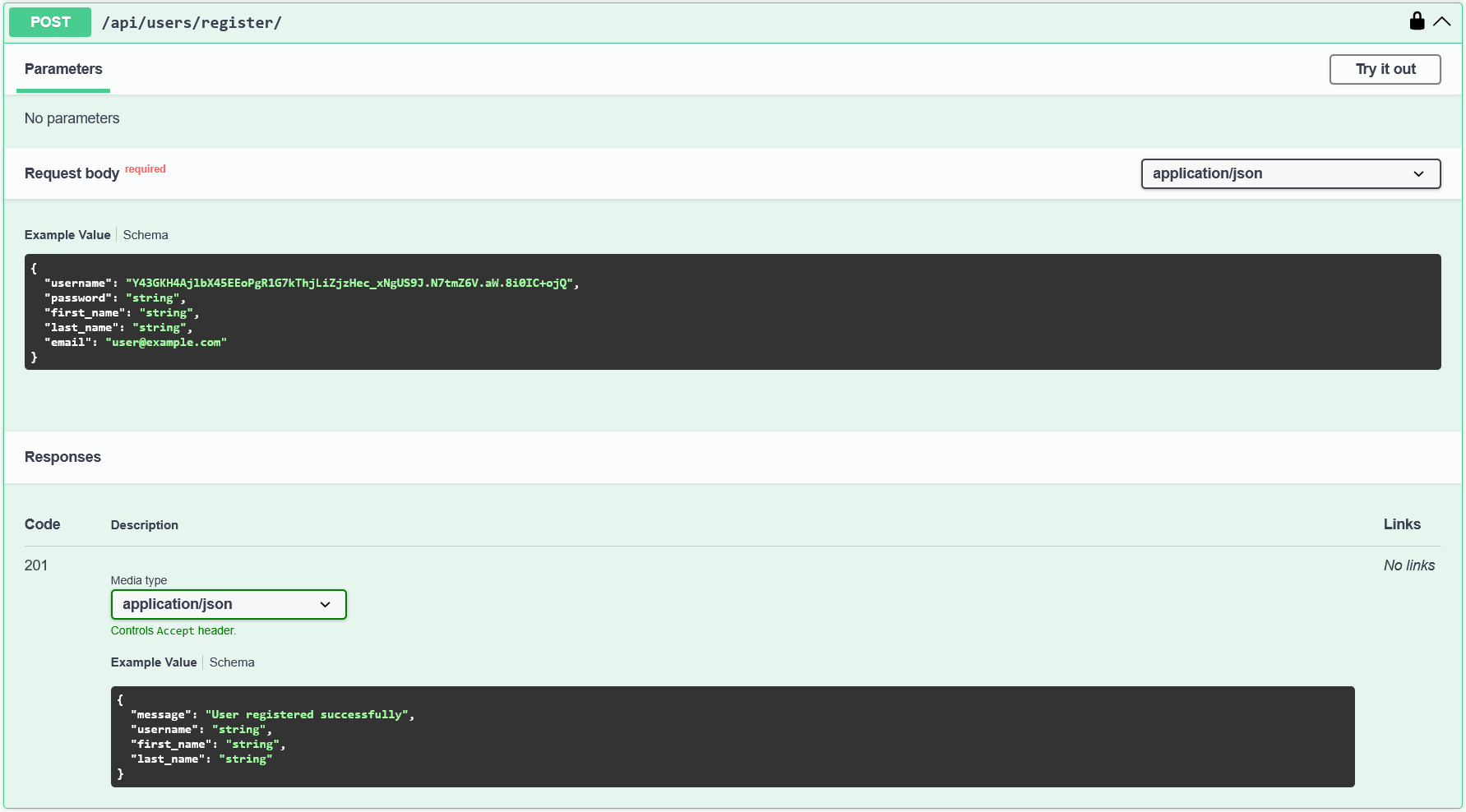
На Фигура 17 е показана първата входна точка от групата users. Тази входна точка представлява функционалността за влизане в системата. На тази входна точка потребителят трябва да изпрати данните си за вход в системата. Това се случва чрез заявка от тип HTTP POST и подаване на тяло към заявката, което 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, тъй като комуникацията се извършва по протокол HTTPS, това дава допълнително ниво на сигурност, за да подсигурим, 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

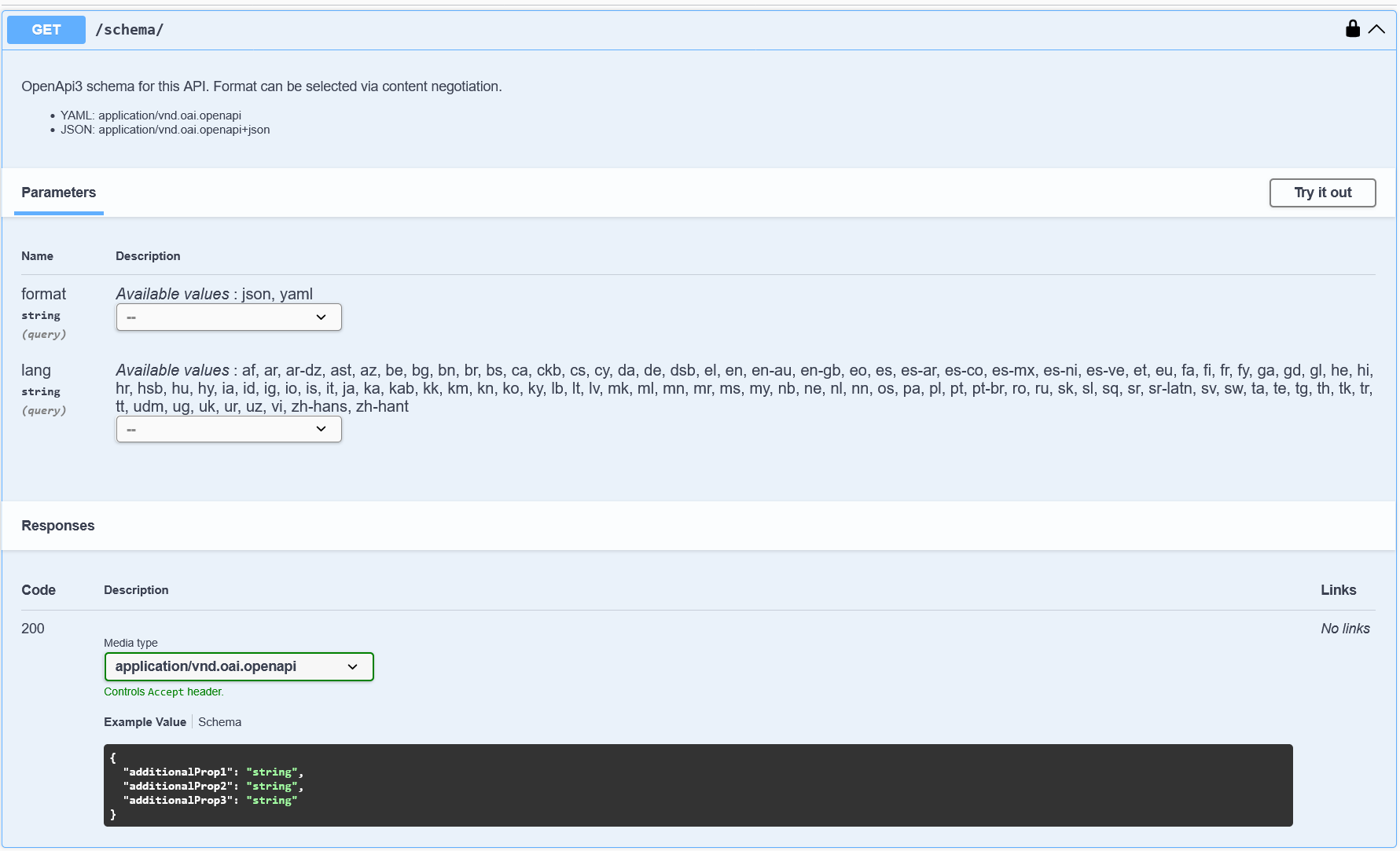
На Фигура 18 е показана входната точка за регистрация на нов потребител. Входната точка е към групата users, като типът ѝ е HTTP POST. Има задължително тяло, което трябва да предостави информацията за потребителя, като потребителско име, парола, email, първо име и фамилно име. При успешно създаване на 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

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].



Фигура 19 Входна точка за получаване на спецификация

### 4.1.3 Изгледи на приложението

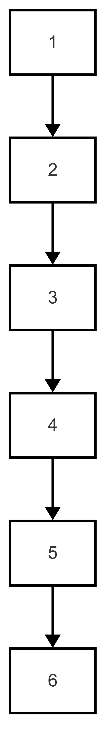
В Django логиката зад входните точки се разполага в така наречените изгледи. Те предоставят удобен начин за групиране на входните точки според какво обслужват. В нашето приложение имаме три изгледа: MenuItemViewSet, OrderViewSet и UserViewSet.

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 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 изтриване на записи.

MenuItemViewSet е изглед, който предоставя операции за меню елементи. Съдържа методите get\_all\_categories, items\_by\_category, get\_аll\_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 извличане 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.

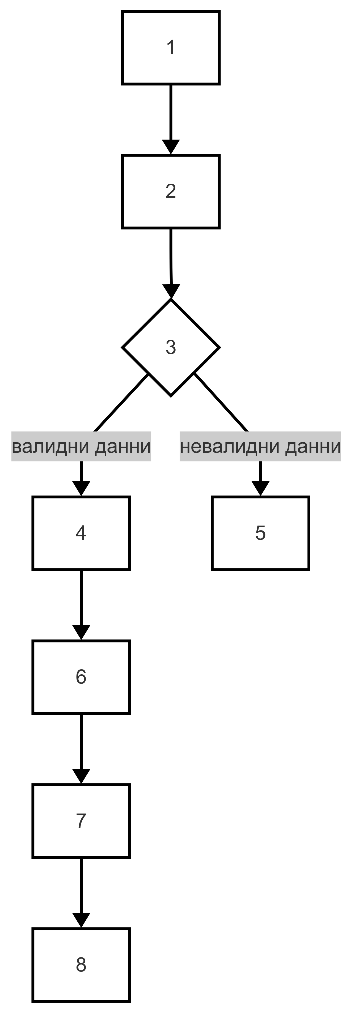


Фигура 20 Поток на работа за метода get\_all\_categories

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.

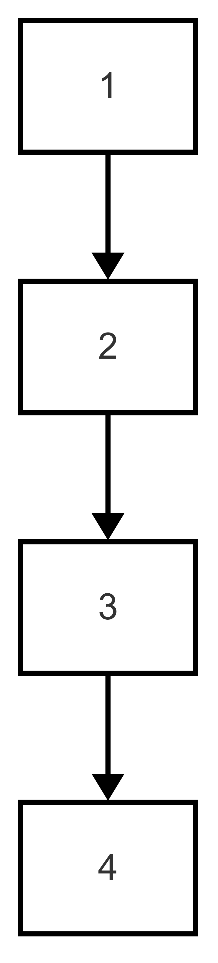


Фигура 21 Поток на работа на метода items\_by\_category

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.
   1. Ако са валидни, премини към 4.
   2. 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\_аll\_items. Този метод не се разклонява, тъй като не проверява данните, а директно връща на потребителя каквото е успял да прочете от базата данни.

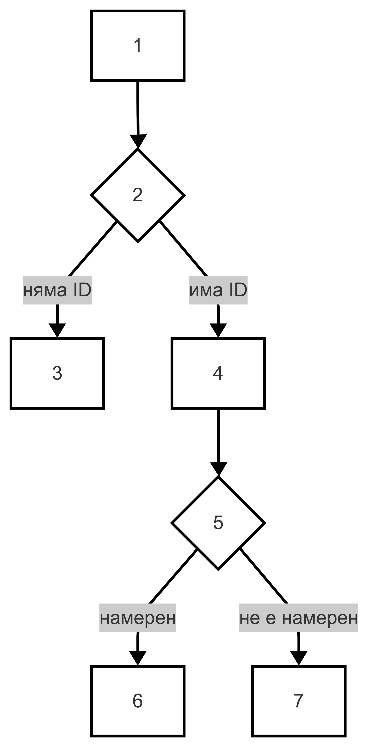


Фигура 22 Поток на работа на метода get\_all\_items

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.



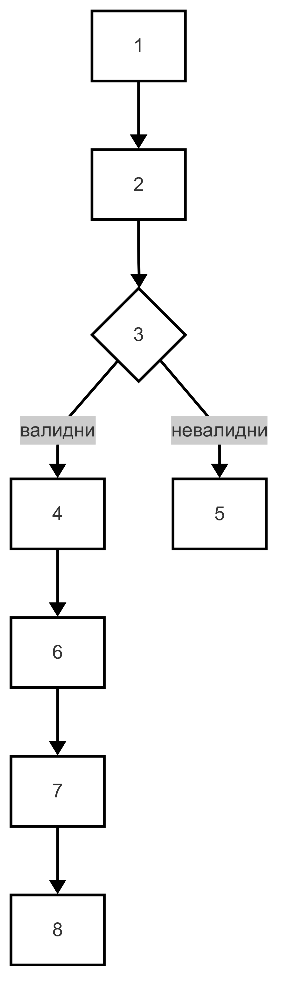
Фигура 23 Поток на метода get\_item\_by\_id

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
   1. Ако няма ID, премини към 3
   2. 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
   1. Ако съществува, премини към 6
   2. 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.

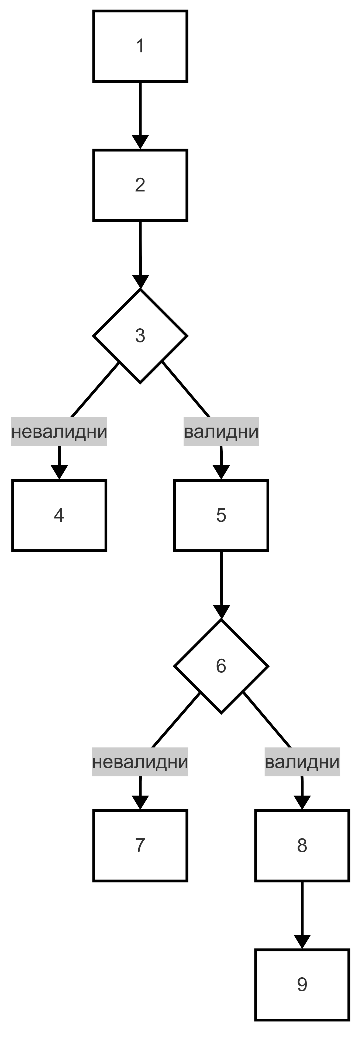


Фигура 24 Поток на работа на метода orders\_by\_user

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.
   1. Ако данните са валидни, преминава към 4.
   2. 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.



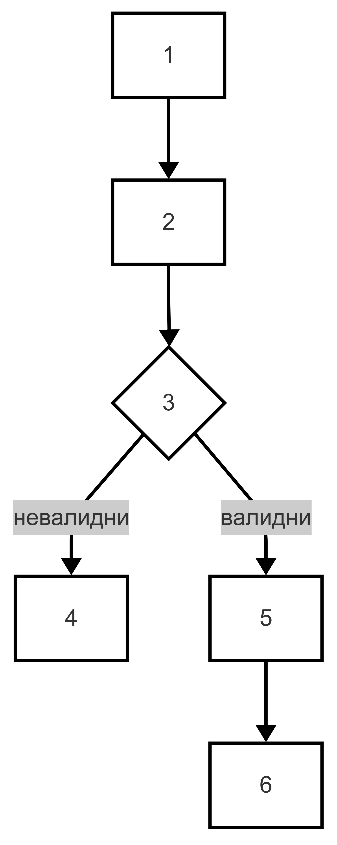
Фигура 25 Поток на изпълнение на метода create\_order

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
   1. Ако са невалидни, премини към 4
   2. 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
   1. Ако не е, премини към 7
   2. 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 потока на работа на метода register, който позволява на потребител да се регистрира в системата. Метода трябва да провери дали данните на потребителя са валидни. Това включва валидация на e-mail, проверка за вече съществуващи потребители и други.

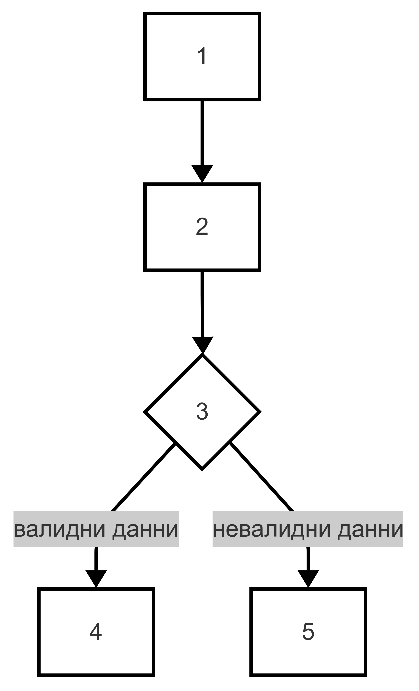


Фигура 26 Поток на метода register

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
   1. Ако не е валиден, премини към 4
   2. 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
   1. Ако данните са валидни, премини към 4
   2. 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

## 4.2 Реализация на мобилно приложение

# Ръководство за използване и примери за употреба.

# Заключение

# Съкращения

8

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

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. |