  
**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**Факултет компютърни системи и технологии**

**Дипломна работа на тема**:

**СИСТЕМА ЗА АВТОМАТИЗИРАНО УПРАВЛЕНИЕ НА ПАРКИНГ**

|  |  |
| --- | --- |
| **София 2024** | **Дипломант:**  **Калоян Каменов Борисов КСИ, 121220159**  **Дипломен ръководител:**  **Ас. Иво Гергов** |

Съдържание

[Списък с фигури 5](#_Toc168421023)

[I. Увод 6](#_Toc168421024)

[II. Постановка на дипломната работа, цели и задачи. 7](#_Toc168421025)

[2.1 Мотивация 7](#_Toc168421026)

[2.2 Цели и задачи 7](#_Toc168421027)

[2.3 Ограничения 7](#_Toc168421028)

[2.4 Кратко описание на технологиите 8](#_Toc168421029)

[1.4.1 Python 8](#_Toc168421030)

[1.4.2 C# - .NET MAUI 8](#_Toc168421031)

[1.4.3 PostgreSQL 8](#_Toc168421032)

[1.4.4 SQLAlchemy 8](#_Toc168421033)

[1.4.5 FastAPI 9](#_Toc168421034)

[1.4.6 Uvicorn 9](#_Toc168421035)

[1.4.7 Разпознаване на обекти 9](#_Toc168421036)

[1.4.8 YOLOv8 9](#_Toc168421037)

[1.4.9 CVAT.AI 10](#_Toc168421038)

[1.4.10 Оптично Разпознаване на Символи 10](#_Toc168421039)

[1.4.11 Aiven 10](#_Toc168421040)

[1.4.12 Docker 10](#_Toc168421041)

[1.4.13 Kubernetes 10](#_Toc168421042)

[1.4.14 Google Cloud 11](#_Toc168421043)

[2.5 Разпознаване на обекти и машинно самообучение 11](#_Toc168421044)

[III. Функционално описание 14](#_Toc168421045)

[3.1 Общ Преглед на системата 14](#_Toc168421046)

[3.2 Описание на обслужващият сървър 15](#_Toc168421047)

[3.3 Описание на базата данни 17](#_Toc168421048)

[3.4 Описание на потребителско приложение 17](#_Toc168421049)

[3.5 Описание на приложение за разпознаване на регистрационни номера 22](#_Toc168421050)

[IV. Програмна реализация. 25](#_Toc168421051)

[4.1 Реализация на сървър, обслужващ базата данни 25](#_Toc168421052)

[4.1.1 Входни точки 25](#_Toc168421053)

[4.1.2 Сервизни услуги 30](#_Toc168421054)

[4.2 Реализация на мобилно приложение 36](#_Toc168421055)

[4.2.1 Страница за вход 38](#_Toc168421056)

[4.2.2 Страница за регистрация 38](#_Toc168421057)

[4.2.3 Страница за общ преглед на регистрационни номера 38](#_Toc168421058)

[4.2.4 Страница за преглед на данни за конкретен регистрационен номер и плащане на престой 38](#_Toc168421059)

[4.3 Реализация на приложение за разпознаване на регистрационни номера 38](#_Toc168421060)

[4.3.1 Подготовка на входни данни 38](#_Toc168421061)

[4.3.2 Трениране на модел 38](#_Toc168421062)

[4.3.3 Реализация на разпознаване на регистрационен номер 38](#_Toc168421063)

[4.3.4 Визуализация на резултат 38](#_Toc168421064)

[V. Ръководство за използване и примери за употреба.  39](#_Toc168421065)

[5.1 Регистрация и създаване на първи регистрационен номер 39](#_Toc168421066)

[5.2 Преглед на данни за вход на автомобил, чрез потребителско приложение 39](#_Toc168421067)

[5.3 Вход на автомобил, който е регистриран в системата и няма задължения 39](#_Toc168421068)

[5.4 Вход на автомобил, който е регистриран в системата, но има задължения 39](#_Toc168421069)

[5.5 Вход на автомобил, който не е регистриран в системата 39](#_Toc168421070)

[VI. Заключение 40](#_Toc168421071)

[Съкращения 41](#_Toc168421072)

[Източници 42](#_Toc168421073)

# Списък с фигури

[Фигура 1 Общ преглед на системата 15](#_Toc168421074)

[Фигура 2 Интерфейс за създаване на потребител 16](#_Toc168421075)

[Фигура 3 Клъстър на Kubernetes 16](#_Toc168421076)

[Фигура 4 Три инстанции на контейнера със сървъра 17](#_Toc168421077)

[Фигура 5 База данни в Aiven 17](#_Toc168421078)

[Фигура 6 Примерна страница за регистрация на потребител 18](#_Toc168421079)

[Фигура 7 Примерен екран за вход 19](#_Toc168421080)

[Фигура 8 Основен потребителски екран 20](#_Toc168421081)

[Фигура 9 Екран с детайли за регистрационен номер 21](#_Toc168421082)

[Фигура 10 Интеракция на потребител с приложение и на приложение със сървър 22](#_Toc168421083)

[Фигура 11 Анотиран регистрационен номер 23](#_Toc168421084)

[Фигура 12 Процес по разпознаване на регистрационен номер 24](#_Toc168421085)

[Фигура 13 Входна точка за създаване на нов потребител 26](#_Toc168421086)

[Фигура 14 Входна точка за вземане на данни на конкретен потребител 26](#_Toc168421087)

[Фигура 15 Входна точка за създаване на нов регистрационен номер 27](#_Toc168421088)

[Фигура 16 Входна точка за вход в паркинга 27](#_Toc168421089)

[Фигура 17 Входна точка за изход от паркинга 28](#_Toc168421090)

[Фигура 18 Входна точка за заплащане на влизане 29](#_Toc168421091)

[Фигура 19 Входна точка за вземане на данни за вход 29](#_Toc168421092)

[Фигура 20 Диаграма на потока на сервизна услуга за създаване на потребител 30](#_Toc168421093)

[Фигура 21 Диаграма на потока за сервизна услига за вземане на данни за потребител 31](#_Toc168421094)

[Фигура 22 Диаграма на потока на сервизна услуга за създаване на регистрационен номер 32](#_Toc168421095)

[Фигура 23 Диаграма на потока на сервизна услуга за вход в паркинга 33](#_Toc168421096)

[Фигура 24 Диаграма на потока на сервизна услуга за изход от паркинга 34](#_Toc168421097)

[Фигура 25 Диаграма на потока на сервизна услуга за плащане 35](#_Toc168421098)

[Фигура 26 Диаграма на потока на сервизна услуга за получаване на данни за вход в паркинг на конкретен регистрационен номер 36](#_Toc168421099)

[Фигура 27 Общ изглед на страниците на мобилното приложение 37](#_Toc168421100)

# I. Увод

Паркингът е част от нашето ежедневие, която е позната на всеки собственик на моторно превозно средство. Това е място, предназначено за временно пребиваване на моторното превозно средство (МПС). Паркингите обикновено не представляват особено технологична структура. Нормалният паркинг представлява голямо, отворено място, където да могат да се паркират МПС удобно. Обикновено има охранител, който се грижи превозните средства да са защитени от недоброжелатели. Охранителят също следи влизащите и излизащите автомобили, като също играе ролята на касиер, при когото се плаща престоят на МПС-то в паркинга.

Тази система може да работи при по-малки паркинги, където е лесно да се следят влизащите и излизащите автомобили. При тези по-малки паркинги, също не се създава натоварено движение на входа или изхода на паркинга. Не се случва често да е необходимо да се изчакват коли. В днешно време, много хора предпочитат да използват лични превозни средства, като автомобили, за да се придвижват от едно място до друго. Това създава растяща нужда от повече паркинги. Заради ограниченото място, не е добра опоция да се изграждат големи количества паркинги, а вместо това да има по-малко, но по-големи паркинги. Тези по-големи паркинги създават трудности при плащане и организация на вход и изход. Тук идва нужда за изграждане на система, която да позволява лесен достъп до паркинга и лесен изход от него. Забавянето при влизане или излизане трябва да бъде сведено до минимум.

За да се наблюдават превозните средства, които влизат и излизат от паркинга, е необходимо да се следят уникалните МПС. Това става лесно, тъй като всяко МПС има уникален регистрационен номер, който е поставен на видно място върху превозното средство. Интелигентната паркинг система може да използва система от камери, която да наблюдава автомобилите и да проверявва регистрационните им номера в база данни от регистрирани потребители и така да ограничава достъпът на неоторизирани лица. Използвайки тази система за достъп е възможно да разберем точният момент на вход и на изход на дадено МПС и да ги запишем в сървър. Така можем да позволим потребителите да плащат сметките си онлайн, без да имаме нужда от касиер на паркинга, който да бави процеса на изход от паркинга.

# II. Постановка на дипломната работа, цели и задачи.

## 2.1 Мотивация

В днешното забързано ежедневие, не е допустимо да съществуват толкова остарели и неудобни системи като паркингите, в това им състояние. Необходимо е да съществува система, която да улеснява потребителите, за да може всичко да се случва бързо и удобно. Не трябва да се налага да има опашки на входовете и изходите на паркингите, докато се чака да се отвори бариера или докато се чака да се прочете бар код, който да отвори изходът. Всички тези забавяния довеждат до трафик и изнервят всички участници в движението.

## 2.2 Цели и задачи

Основните цели и задачи на тази разработка са да адресират проблемите със забавянето при паркингите и да предоставят лесен и удобен начин на потребителите да взаимодействат с паркингите. Това ще се случи, чрез приложение за всяка широко използвана операционна система, което да предоставя потребителски интерфейс. Това приложение ще позволява да бъде регистриран потребител и след това, потребителят да може сам да добави автомобилът си. След като автомобил е добавен, автоматизираното управление на паркингът трябва да започне да отваря входът и изходът, когато регистрираният автомобил се доближи. Приложението трябва да предоставя на потребителите лесен начин за плащане на паркинга, така че всеки да може да заплати в удобно за него време. Паркингът ще използва система от камери, които да определят кога доближава автомобил и дали този автомобил е в списъкът на регистрираните автомобили.

## 2.3 Ограничения

За разработката на тази система е необоходимо да се съобразим с някои ограничения. Един от най-големите проблеми, който изниква при разпознаване на текст, чрез камера, е яснотата на изображението, заснето от камерата. Това изображение трябва да бъде ясно, за да може системата да определи текстът, записан върху регистрационният номер. Необходимо е камерите да бъдат оборудвани с поляризиращ филтър, който да елиминира отблясъци от светлина, както и зоната около камерата да бъде добре осветена, за да може автомобилите да бъдат разпознавани дори и нощем.

Друго ограничение е самият потребител. Тази система разчита на способността на потребителят да използва мобилно устройство или компютър. Ако потребителят няма възможност да използва такова устройство, той не би могъл да се регистрира в паркинг системата и следователно, той не би могъл да използва паркинг съоражението.

## 2.4 Кратко описание на технологиите

### 1.4.1 Python

Избраният език за реализация на голяма част от тази дипломна работа е Python. Езикът позволява лесна работа с голямо количество библиотеки и структури от данни. Това помага с бърза реализация на сложни системи, използвайки само няколко реда програмен код.

Езикът за програмиране има употреба при статистически проучвания, заради лесната работа със структури от данни. Също е предпочитан за използване при работа с изкуствен интелект, заради простият си синтаксис. Това прави програми, написани на този език, лесни за четене и поддръжка. Езикът предоставя гъвгави инструменти за работа, голямо количество от библиотеки и голямо общество от разработчици, които го поддържат.

### 1.4.2 C# - .NET MAUI

Езикът C# е използван за реализация на мобилно приложение, което да позволява достъп до данните от системата за паркиране. Използвана е структурата за програмиране Multi-platform App UI (MAUI), която от своя страна стъпва върху структурата .NET.

Това е популярна структура, която позволява разработка на приложения, които могат да работят върху различни операционни системи, използвайки един и същи код. Структурата позволява дадено приложение да работи върху Android, iOS, Windows и macOS, без да са необходими промени в кода. Тази структура е наследникът на Xamarin.Forms. Тя разширява функционалността на старата структура, като добавя увеличена производителност и по-лесни за разширяване приложения.

Структурата MAUI работи като унифицира интерфейсите за програмиране на приложения на Android, iOS, Windows и macOS и предоставя собствени интерфейси, които сами определят коя платформа трябва да използват. [1]

### 1.4.3 PostgreSQL

PostgreSQL е система за управление на релационни бази данни с отворен код. Системата използва и разширява езикът Structured Query Language (SQL) и предоставя функционалност за сигурно съхранение на данни. Системата за управление може да работи под всички често използвани операционни системи, предоставя голямо количество инструменти за улеснено управление или разширяване на функционалност и има широк кръг от потребители, което прави решаването на проблеми лесно. [2]

### 1.4.4 SQLAlchemy

SQLAlchemy е библиотека за Python, която позволява лесна работа с голямо количество системи за управление на релационни бази данни. Библиотеката предоставя унифициран начин за достъпване на данните в базите данни, което ни позволява лесно да сменим типа на релационната база данни, без да се налага да се променя кодът, който използва базата данни. Това прави тази библиотека много ценна при създаване на каквото и да е приложение, което работи с база данни.

За да се създаде връзка с база данни, е необходимо да има известни адрес на сървъра на базата, порт на сървъра на базата, потребителско име, парола и име на базата данни. След това библиотеката сама се грижи за тази връзка и ние просто трябва да създаваме сесии, с които да работим.

### 1.4.5 FastAPI

FastAPI представлява библиотека на Python, която позволява изграждане на интерфейси за достъп до сървър. Библиотеката се работи директно с езика Python, което значи, че потребителят просто трябва да посочи парче код, което иска да използва като интерфейс за достъп, и библиотеката FastAPI сама се грижи за изграждането на интерфейсa.

След като се опишат входните точки на интерфейса, FastAPI може да генерира стандартизиран документ, който описва всички интерфейси и данни, които приема приложението, познат като OpenAPI файл. Този файл се използва от клиентски приложения за да се генерира код, който да позволи връзка с нашето приложение.

### 1.4.6 Uvicorn

Uvicorn e уеб сървър за Python. Представлява имплементация на Asynchronous Server Gateway Interface (ASGI) спецификацията. Тази спецификация описва интерфейси за изграждане на асинхронни и синхронни сървъри на Python.

Uvicorn е напълно съвместим с OpenAPI спецификацията и следователно е съвместим с FastAPI. Така в едно приложение можем да използваме FastAPI за да създадем OpenAPI съвместими входни точки за сървърът ни и след това да пуснем Uvicorn, който да обслужва входните точки.

### 1.4.7 Разпознаване на обекти

Разпознаване на обекти или Object Detection представлява технология, при която се тренира невронна мрежа, която да разпознава дадени обекти. Тази технология е техника за ‚компютърно зрение‘, което буквано представлява способността на дадена машина да ‚вижда‘. Ще навлезем по-дълбоко в невронни мрежи и разпознаване по-късно в тази глава.

### 1.4.8 YOLOv8

YOLOv8 или You Only Look Once представлява модел за разпознаване на обекти на Ultralytics, който е бърз, точен и лесен за употреба. Моделът може да класифицира обекти, да разпознава обекти, да сегментира обекти, да проследява обекти и да определя пози. Моделът има няколко различни размери, като по-големите модели са по-точни, но и изискват повече ресурси за да работят. Най-лекият модел е nano, последван от small, последван от medium, след това large и най-големият модел x. [3]

### 1.4.9 CVAT.AI

Това е платформа за анотиране на данни. Анотацията представлява процес, при който потребител минава през сет от данни с цел да им даде някакво определение. В тази дипломна работа, са анотирани около 150 различни изображения на автомобили с регистрационни номера, за да бъдат използвани за обучение на модел за разпознаване. Платформата предоставя уеб интерфейс, който улеснява работата и множество инструменти, които покриват всякакви сценарии, с които можем да се сблъскаме при анотиране на данни. Платформата е безплатна за ползване при по-малки проекти.

### 1.4.10 Оптично Разпознаване на Символи

OCR или Оптично Разпознаване на Символи представлява технология, при която снимка на текст, било той ръчно написан или принтиран, се подава на конвертор, който превръща снимката в текст, разбираем от машина. Тази технология позволява автоматично въвеждане на данни от документи, което елиминира ръчното въвеждане и така увеличава скоростта на обработка на данни. Проблемът е, че снимката трябва да бъде ясна, за да може алгоритъмът да прочете данните коректно.

### 1.4.11 Aiven

Aiven е платформа за управление на данни. Платформата предоставя възможност данни да бъдат следени, оптимизирани и съхранявани. Предоставя безплатен план, който можем да използваме за да съхраняваме данните си в база данни, работеща с PostgreSQL. Необходимо е да бъде настроен сървър, което става лесно, чрез предоставен потребителски интерфейс.

### 1.4.12 Docker

Docker е инстртумент, използван за създаване на изображение на приложението, което се използва за управление на базата данни. Чрез това изображение, можем да създадем контейнер, който съдържа нашето приложение и необходимите му зависимости. Това ни позволява да изпълняваме нашето приложение независимо от операционната система, върхо която работи Docker. Инструментът е безплатен и е широко използван в индустрията.

### 1.4.13 Kubernetes

Kubernetes е система за оркестрация на контейнери. Тази система се грижи за работата на контейнерите, които са пуснати в нея. Управлява ресусрсите на компютърът, на който работи и осигурява непрекъсната работа на контейнери. Ако даден контейнер падне, Kubernetes трябва да го върне в работно състояние. Този инструмент е доказал полезността си и се използва в най-големите компании.

### 1.4.14 Google Cloud

Това е решението на Google за лесно създаване и управление на проекти в облака. Платформата предоставя потребителски интерфейс за управление на приложение и централизирано място, от където да се наблюдават всички включени системи. Има голямо количество разширения, които можем да използваме за да развием нашето приложение. Платформата предоставя безплатен план, който може да бъде използван за малки приложения. За нашите цели, ще използваме Google Cloud за да управляваме Kubernetes cluster, в който да работи нашият контейнер, който съдържа софтуерът за връзка с базата данни.

## 2.5 Разпознаване на обекти и машинно самообучение

Машинното самообучение представлява разработка, при която на даден алгоритъм се предоставят познати данни, от които да се самообучи, за да може да разпознава тези данни, при непознати ситуации.

Алгоритъмът за самообучение представлява мрежа от параметри, които много наподобяват неврони. Тези невронни мрежи имат променливо количество параметри, като те са разположени на нива. Входното ниво представлява данните, които подаваме. След това съществуват даден брой скрити нива, където се случва обработката на данните. Скритите нива са последвани от изходно ниво, където виждаме какъв е изходът от невронната мрежа. Невроните на различните нива са свързани с невроните от предишното и следващото ниво, като силата на връзката определя как би се активирал невронът. При слаба връзка между два неврона, силна активация на един неврон ще доведе до слаба активация в друг неврон. Тренирането на невронната мрежа представлява точно определянето на силата на връзките между нивата. Подават се данни на входа и се описва желаният изход. Задачата на невронната мрежа е да обработи входните данни и да се доближи максимално до желаният изход.

При трениране на алгоритъм е необходимо голямо количество разнообразни входни данни, за да може алгоритъмът да бъде генерализиран за типа данни. Ако на даден алгоритъм се подадат малко количество данни, той ще се научи само на няколко конкретни случая и няма да бъде добър, когато му бъдат подадени данни, които не е виждал до сега. Също, заради начинът по който невронната мрежа определя теглата между нивата, е необходимо неколкократно изпълнение на тренировъчния процес. При първо преминаване през данните, е възможно връзките между нивата да не бъдат достатъчно ясно изразени, което да доведе до несигурност в изходните данни. Изпълнението на тренировъчният процес се нарича епоха. Препоръчва се да бъдат използвани няколко тренировъчни епохи, като техният брой не може да бъде определен универсално, заради всички разлики при обучението на различни модели. Оптималните епохи обикновено се определят на прицнипът проба-грешка.

Тренировката на мрежата се състои от две части: същинското трениране и валидацията. Вече описахме какво представлява тренирането на мрежата. След като приключи етапът на трениране, следва етап на валидиране. В този етап невронната мрежа трябва да премине през данни, които ние сме подготвили предварително, но този път без да знае какъв изход трябва да получи. Ние, разбира се, знаем какъв трябва да бъде изходът и след като мрежата премине данните, трябва да сравним изходът на алгоритъма с очакваният от нас изход. Ако резултатът, който сме получили е задоволителен, значи мрежата е готова за използване. Ако видим разлики между очакваният резултат и полученият резултат, следва да тренираме мрежата отново. При последващи тренирания на мрежата, трябва да бъдат направени промени, за да не получим същият изход отново. Какви ще бъдат тези промени зависи изцяло от какви проблеми сме видели при изходните данни. Ако, например, много от тестовите ни данни не са получили резултат изобщо, е възможно мрежата да не е била тренирана достатъчно пъти. Следва да увеличим броят епохи за следващо трениране. Съществува друг вариант, при който мрежата ни връща резултат, но той е погрешен. Този проблем е по-сложен за решаване, тъй като той може да бъде породен от много различни причини. Една от най-често срещаните причини е проблем с входните данни. Възможно е да не е определен коректно изходът от алгоритъма или данните да не са напълно ясни, което може да доведе до изкривяване на очакванията. В тази ситуация алгоритъмът работи правилно, но данните, които предоставяме, не са верни, поради човешка грешка. Другият вариант е, когато данните ни са коректни, но изходът все още да не е верен, е когато невроните в мрежата не са достатъчно или не са разпределени между нивата коректно. Неправилното разположение на невроните и връзките между тях може да доведе до сериозни изкривявания в изхода, тъй като всяко следващо ниво представлява филтриран резултат от предишно. За нещастие, няма готова рецепта, която да определи колко неврони трябва да имаме на всяко от вътрешните нива или колко вътрешни нива трябва да имаме. Всяка невронна мрежа е различна и изисква да бъде настроена според конкретните си нужди. Това, обикновено, се случчва на принципа проба-грешка. От добрата страна, е доста лесно да настроим входните и изходните неврони, тъй като те се определят директно от нашите данни. Ако, например, имаме алгоритъм, на който подаваме изображение на цифра, и искаме да разпознае каква е тази цифра, то входът ни би имал по един неврон за всеки пиксел от изображението. Този вход, трябва да ни даде цифра от 0 до 9, следователно изходните неврони трябва да бъдат 10 на брой, по един за всеки възможен изход. Вътрешните неврони трябва да бъдат нагодени, за да получим оптимален резултат с нашите данни.

При разпознаването на обекти се ползва много подобен подход. Трябва да създадем сет от данни, който да представя обектите, които търсим. Колкото е по-голям сетът, толкова по-добре ще бъде обучена мрежата. Важно е сетът да е не-само голям, а и да бъде разнообразен. Трябва да предоставим ясни изображения, които да са добре анотирани. По този начин, чрез разнообрази и анотирани изображения, ще получим по-генерализиран модел, който ще може да разпознава обекти по-добре. След това трябва да тренираме невронната мрежа, чрез тези данни, подобно на примера с цифрите, но този път по-сложно.

При разпознаването на обекти имаме конкретен очакван изход, но входът не е толкова изчистен, както би бил при разпознаване на единични цифри. Разпознаването на обекти предполага, че обектът се намира в дадена среда и не е единственото нещо в изображението. За да може нашият алгоритъм да разпознава обектите ефективно, независимо от средата, той трябва да може да разпознава единични характерни черти на обектите и чрез тях да може да разпознае самият обект. Това работи добре, когато обектите са ясно изразени и видиими добре. При замъглени обекти, където характеристиките им не са ясно отличени, може да има проблем, при който не може да бъде разпознат обекта. Заради начинът, по който работят моделите за разпознаване на обекти, те могат да бъдат обучени да разпознават много различни обекти, чрез индивидуалните им черти. В едно голямо изображение, можем да имаме, например, три различни обекта, върху които да е обучен нашият модел и той да успее да ги разпознае по характерните им черти. След като обектите са разпознати, следващата задача на модела за разпознаване на обекти е да определи къде в изображението се намират разпознатите обекти. При по-простите модели, това се случва с намиране на координатите на изображението, в който е разположен обектът, и определяне на правоъгълник, който обхваща дадения обект. При по-сложните модели, може да бъде определен конкретният силует на обекта, точните му граници и позицията му в изображението.

# III. Функционално описание

## 3.1 Общ Преглед на системата

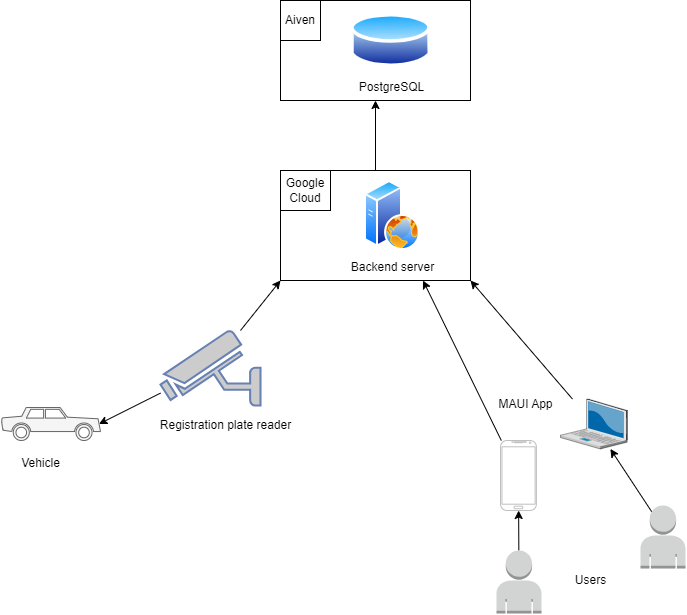
Системата се състои от 4 основни части: потребителският интерфейс, написан на .NET MAUI, софтуерът за разпознаване на регистрационни номера, написан на Python, сървър, обслужващ базата данни, написан на Python и база данни PostgreSQL.

Комуникацията между обслужващият сървър и двете приложения, се случва, чрез POST заявки на HTTP протокол. Това позволява лесен обмен на данни, чрез унифициран интерфейс, което улеснява имплементацията на междусистемните връзки.

Комуникацията между базата данни и системата, която я управлява, се случва посредством SQL заявки, които се изпращат към адреса на базата данни.

Системата за управление на базата данни позволява конкурентен достъп до базата, използвайки няколко обслужващи процеса. По този начин имаме непрекъсната работа с потребителски приложения.

Фигура 1 показва опростен изглед на системата. Показан е обслужващият сървър, работещ в Google Cloud, който комуникира с базата данни в Aiven. Приложенията за потребителски интерфейс и разпознаване на регистрационни номера се свързват с тази система, като по този начин избягват директна връзка с базата данни.



Фигура 1 Общ преглед на системата

## ****3.2**** Описание на обслужващият сървър

Реализацията на обслужващият сървър ще бъде изпълнена на Python. Основната причина за използване на езика е, че пускането на такъв сървър се случва много бързо и лесно. Python ни дава достъп до библиотеката SQLAlchemy. Тази библиотека позволява лесна работа с бази данни, като от нас, като потребители, се изисква само да знаем адреса на сървъра, с който се свързваме, и данните за вход. Друга библиотека, която позволява лесно създаване на сървъра е FastAPI. Тази библиотека улеснява създаването на интерфейс за комуникация с приложението, като изпозлва функциите, които създаваме, за да създаде връзките. На Фигура 2 е показан примерен интерфейс, който позволява създаване на нов потребител. Интерфейсът е описан от FastAPI, чрез OpenAPI, което е универсален стандарт за описване на такива интерфейси. Това ни позволява лесно да генерираме клиентско приложение, което да се свързва с нашият интерфейс.

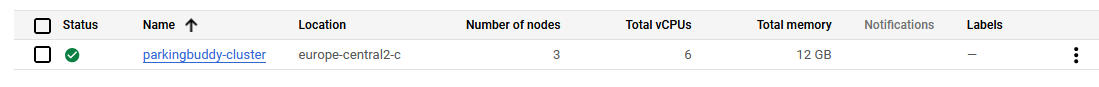
A screenshot of a video game

Description automatically generated

Фигура 2 Интерфейс за създаване на потребител

FastAPI предоставя сървър Uvicorn, работещ по ASGI стандартът за Python. Това е решение, което идва заедно с библиотеката и ни позволява да стартираме приложението си само с няколко реда код.

Обслужващият сървър ще работи в Google Cloud, като за да бъде стартиран в облака, е направен Kubernetes Cluster за проекта. Показан е примерен клъстер на Фигура 3.



Фигура 3 Клъстър на Kubernetes

В този клъстер, ще бъде пуснат Docker контейнер, който съдържа изображението на програмата на сървъра, заедно с всички необходими програми, за да се осигури постоянна работа. Kubernetes се грижи приложението да бъде постоянно активно. При фатална грешка, ще бъде записано съобщението на грешката и контейнерът на приложението ще бъде рестартиран, за да продължи работата. Приложението работи върху три отделни процеса, за да се осигури постоянна възможност за обслужване на заявки, както е показано в примера на Фигура 4.

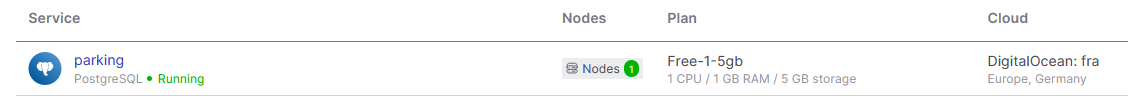
A screenshot of a computer

Description automatically generated

Фигура 4 Три инстанции на контейнера със сървъра

## 3.3 Описание на базата данни

Базата данни ще работи на сървър на Aiven, за да бъде по-лесно управлението на базата. Aiven предоставя инструменти, които правят лесно създаването и достъпването на база данни, като предлагат безплатен план, който е подходящ за целите на тази разработка. На Фигура 5 е показана примерна база данни, разположена на сървърите на Aiven.



Фигура 5 База данни в Aiven

Aiven позволява да управляваме връзките в базата, като можем да терминираме заявки, които отнемат прекалено много време и така да се защитим от потенциални DOS атаки. Платформата също позволява да създаваме резервни копия на базата данни, което е много важно по време на разработка на приложение, тъй като е много лесно да загубим или повредим записани данни, ако не сме внимателни.

## 3.4 Описание на потребителско приложение

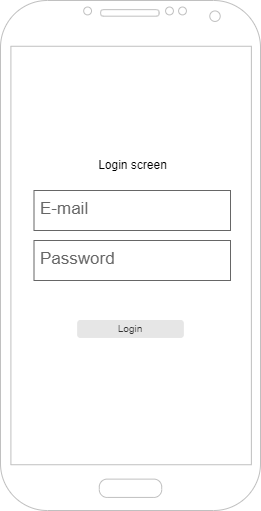
Приложението, предоставящо потребителският интерфейс, е ще бъде изготвено на .NET MAUI. Реализацията, чрез MAUI ни позволява да изградим приложение, което може да работи на всички често използвани операционни системи. Приложение, изградено на MAUI, може да работи под операционни системи като Android, Windows и macOS. По този начин, ние не ограничаваме нашите потребители до определени операционни системи. Не се и налага да поддържаме няколко различни приложения, тъй като MAUI унифицира програмният код до едно приложение, което може да бъде компилирано за различни платформи. По този начин, нашата работа намалява значително.

Потребителският интерфейс, трябва да позволява на потребителят да създаде собствен профил, без да има нужда да се свързва със собственика на паркинга. Странициата на приложението, която се занимава със създаване на потребителски профил, трябва да изисква потребителско име, e-mail адрес, парола и да изисква потвържение за паролата, въведена от потребителя. На Фигура 6 е показана примерна страница за регистрация на нов потребител.



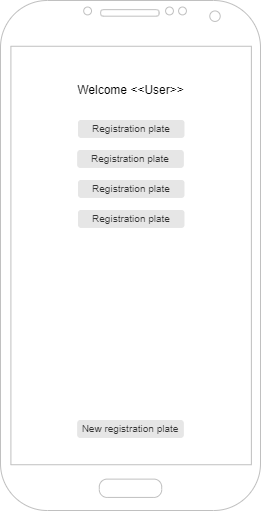
Фигура 6 Примерна страница за регистрация на потребител

Приложението трябва да предоставя екран за вход, тъй като веднъж регистриран, потребителят не би трябвало да въвежда всичките си данни отново. Този екран за вход, ще изисква по-малко данни да бъдат въведени, тъй като можем да считаме e-mail адресите за уникални. За вход е достатъчно да се подаде e-mail адрес, който вече е регистриран и паролата, която е асосциирана с него. Примерен екран за вход е показан на Фигура 7.



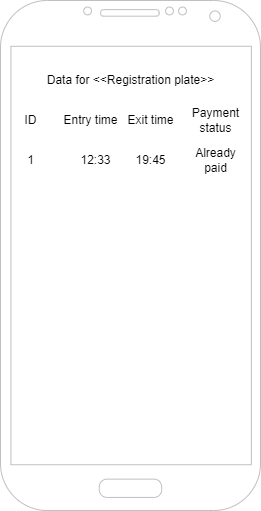
Фигура 7 Примерен екран за вход

След като потребителят влезе в приложението, дали с регистрация, като нов потребител, или след вход, той трябва да бъде посрещнат от екран, където да може да види добавените си регистрационни номера, както и да добави нов номер. От този екран ще се случва основната навигация в приложението. На Фигура 8 е показано как би изглеждал подобен екран.



Фигура 8 Основен потребителски екран

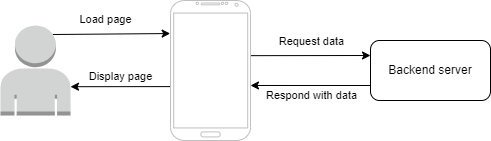
След като потребителят е избрал даден регистрационен номер, приложението ще отвори страница, която ще покаже всички данни, с които разполага паркинг системата, за съответният регистрационен номер. Системата трябва да покаже номерът на входа, дата и час на вход, дата и час на изход и да даде възможност на потребителят да заплати своят престой на паркинга. Ако престоят вече е платен, приложението трябва да информира потребителя, че не е необходимо допълнително заплащане. На Фигура 9 е показан примерен екран за показване на данните за регистрационен номер.



Фигура 9 Екран с детайли за регистрационен номер

За показаните примери е използвано мобилно устройство, но е важно да се отбележи, че програмата би изглеждала по същият начин и би работила по същият начин, дори и ако е заредена на друга операционна система или върху друг тип устройство, като например персонален компютър.

При зареждане на всеки един от тези екрани, приложението на MAUI прави HTTP заявка към сървъра, за да получи данните, необходими за визуализиране на следващия екран. Възможно е да има забавяне при зареждане на следващ екран, което е предизвикано от скоростта и латентността на интернет връзката на потребителското устройство. На Фигура 10 е показано как потребителят управлява приложението и как приложението прави заявки към сървъра, когато трябва да зареди нова страница.



Фигура 10 Интеракция на потребител с приложение и на приложение със сървър

## 3.5 Описание на приложение за разпознаване на регистрационни номера

Приложението, което разпознава регистрационни номера, ще бъде написано на Python, тъй като езикът е широко използван при приложения с компютърно зрение и има готови библиотеки, които ще използваме за трениране на моделът за разпознаване на регистрационни номера.

За тренирането на модела, ще използваме библиотеката Ultralytics, с помощта на която ще тренираме модел YOLOv8. Тези модели ни позволяват да засичаме, проследяваме и класифицираме обекти от видеа и снимки. Моделите идват в различни размери, като колкото по-голям е моделът, толкова по-добре се справя с разпознаването. Проблемът идва от там, че колкото е по-голям моделът, толкова повече време отнема тренирането.

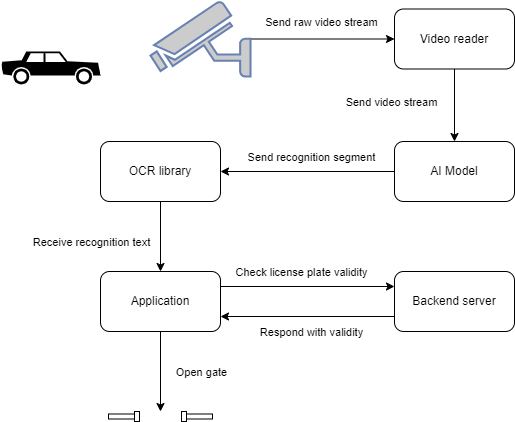
За тренирането на модела, трябва да бъде подаден комплект от данни, който да показва регистрационни номера върху автомобили. Поради липса на такъв комплект от данни, който да е публично достъпен, се налага да бъде изграден нов, използвайки снимки, налични в интернет на български автомобилни регистрационни номера. За изграждане на новия сет от данни, ще бъде използван онлайн инструментът CVAT.AI. Този инструмент ще помогне с анотацията на изображенията, които ще бъдат използвани за трениране и валидиране на модела. На Фигура 11 е показан анотиран регистрационен номер на автомобил. Съдържанието на регистрационният номер е прикрито, за да бъде запазено конфиденциално.

A close up of a car

Description automatically generated

Фигура 11 Анотиран регистрационен номер

След като вече е създаден модел, който умее да разпознава регистрационни номера, с висока точност, следва да се премине към етап на имплементация на функционалност, за четене на видеозапис и пускане на този видеозапис към вече тренираният модел. След това, моделът ще обработи записа и ще даде като отговор координати на регистрационен номер, ако той е успял да прочете такъв. При успешно прочитане, отрязъкът, където е засечен регистрационният номер, ще бъде подаден на екранен четец, който ще извлече символите. След това, приложението ще използва извлечените символи, за да се обърне към сървърът, който обслужва базата данни, за да провери дали автомобилът има право да достъпи паркинга, където работи системата. На Фигура 12 е показан процесът за разпознаване на регистрационен номер, в случай, когато номерът е записан в базата данни и автомобилът има право да влезе на паркинга.



Фигура 12 Процес по разпознаване на регистрационен номер

След като автомобилът е влязъл в паркинга, моментът на неговото влизане се записва в базата данни, за да бъде използвано при проверката за плащане. След като автомобилът приключи престоят си, той може да напусне паркинга свободно. При напускане на паркинга се записва часът на излизане. Ако автомобилът е прекарал по-малко от дадено, предварително зададено, време, то престоят му автоматично се маркира като платен. Ако автомобилът, обаче, е прекарал повече от позволеното време, се появява бутон, който кара потребителят да заплати престоя си. Ако потребителят не заплати престоя си, той не може да прави последващи посещения на паркинга. Това означава, че потребител има право на едно безплатно посещение на паркинга, което ако желае, може да не заплати. За всяко следващо посещение, ще бъде изискано предното посещение да бъде заплатено.

# IV. Програмна реализация.

## 4.1 Реализация на сървър, обслужващ базата данни

Реализацията на сървърното приложение може да се раздели на три основни части:

* Входни точки
* Сервизни услуги
* Типове данни

Входните точки представляват всички места, където FastAPI, посредством OpenAPI, предоставя интерфейси за получаване или подаване на данни към приложението. В тази имплементация са използвани само HTTP POST заявки, но съществуват и други видове, които могат да бъдат по-подходящи, според разработваната система. Всяка входна точка има сервизна услуга, която използва, за да извърши определен тип дейност. Входните точки предават данните, получени отвън към сервизните услуги и обратното.

Сервизните услуги представляват основната функционалност. Всяка сервизна услуга отговаря на дадена входна точка на програмата. Сервизната услуга борави с данните, които е получила от входната точка, като също така има директен достъп до базата данни, за да може да борави с информацията от там.

Типовете данни са унифицирани типове, които са описани в OpenAPI документа, за да може да бъдат използвани за лесна комуникация с външни системи. Всяко приложение, което имплементира клиент, използващ това OpenAPI, трябва да използва същите типове данни, както е описано в това сървърно приложение.

### 4.1.1 Входни точки

На Фигура 13 е показана входната точка за създаване на нов потребител, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /users/new за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за създаване на нов потребител. Ще бъде върнат отговор от тип JSON, който в себе си съдържа HTTP код на операцията и съобщение, върнато от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 201 - Created new user – При успешно създаден нов потребител
* 400 - Invalid data provided for user – При невалидни подадени данни
* 409 - User with provided data already exists – При дублиране на потребител

A screenshot of a computer

Description automatically generated

Фигура 13 Входна точка за създаване на нов потребител

На Фигура 14 е показана входната точка за извличане на данни за потребител, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /users/data за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип UserDataModel, който в себе си съдържа HTTP код на операцията и данни, върнати от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 200 - Got user data – При успешно извлечена информация
* 400 - Invalid data provided for user – При невалидни подадени данни

A screenshot of a computer

Description automatically generated

Фигура 14 Входна точка за вземане на данни на конкретен потребител

На Фигура 15 е показана входната точка за създаване на нов регистрационен номер, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /plates/new за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип JSON, който в себе си съдържа HTTP код на операцията и съобщение, върнато от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 201 - Created new registration plate – Създаден е нов регистрационен номер
* 400 - Invalid data provided – Невалидни данни са подадени
* 409 - Plate with provided data already exists – Регистрационният номер вече съществува

A screenshot of a computer

Description automatically generated

Фигура 15 Входна точка за създаване на нов регистрационен номер

На Фигура 16 е показана входната точка за вход в паркинга, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /entry/in за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип JSON, който в себе си съдържа HTTP код на операцията и съобщение, върнато от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 201 - Created new parking entry – Създаден е нов вход в паркинга
* 400 - Invalid data – Невалидни данни са подадени
* 402 - Previous parking has not been paid – Предно паркиране не е платено
* 409 - Plate with provided data already parked – Регистрационният номер вече е паркиран

A screenshot of a computer

Description automatically generated

Фигура 16 Входна точка за вход в паркинга

На Фигура 17 е показана входната точка за изход от паркинга, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /entry/out за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип JSON, който в себе си съдържа HTTP код на операцията и съобщение, върнато от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 200 - Exited successfully – Успешен изход от паркинга
* 400 - Invalid data – Невалидни данни са подадени
* 409 - Plate with provided data never parked – Регистрационният номер не е паркиран

A screenshot of a computer

Description automatically generated

Фигура 17 Входна точка за изход от паркинга

На Фигура 18 е показана входната точка за заплащане, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /entry/pay за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип JSON, който в себе си съдържа HTTP код на операцията и съобщение, върнато от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 200 - Paid successfully – Успешно заплащане на вход
* 202 - Entry already paid – Следващо влизане вече е платено
* 400 - Invalid data – Невалидни данни са подадени

A screenshot of a computer

Description automatically generated

Фигура 18 Входна точка за заплащане на влизане

На Фигура 19 е показана входната точка за вземане на данни за влизания на даден регистрационен номер, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адресът на сървъра, който трябва да бъде последван от /entry/data за да достъпим дадената точка. Входната точка приема HTTP заявка от тип POST, която трябва да има тяло, което да съдържа информацията, необходима за откриване на конкретен потребител. Ще бъде върнат отговор от тип списък от EntryDataModel, който в себе си съдържа HTTP код на операцията и списък от данни, върнат от програмата. Възможно е да бъдат върнати няколко различни отговора:

* 200 - Got entry data – При успешно взети данни за вход
* 400 - Invalid data provided for entry – Когато не е открит регистрационен номер в базата

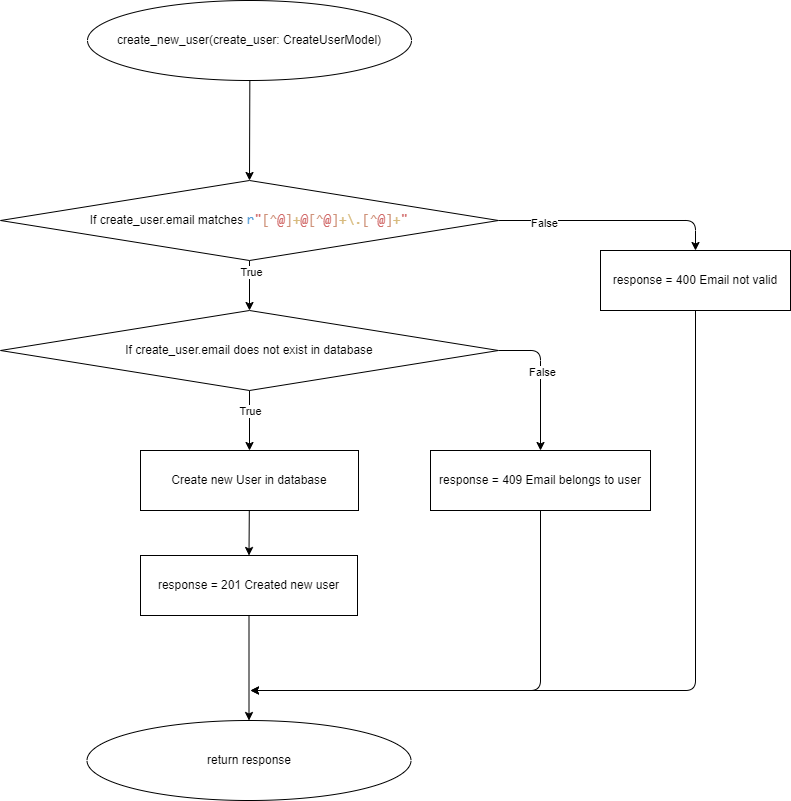
A screen shot of a computer

Description automatically generated

Фигура 19 Входна точка за вземане на данни за вход

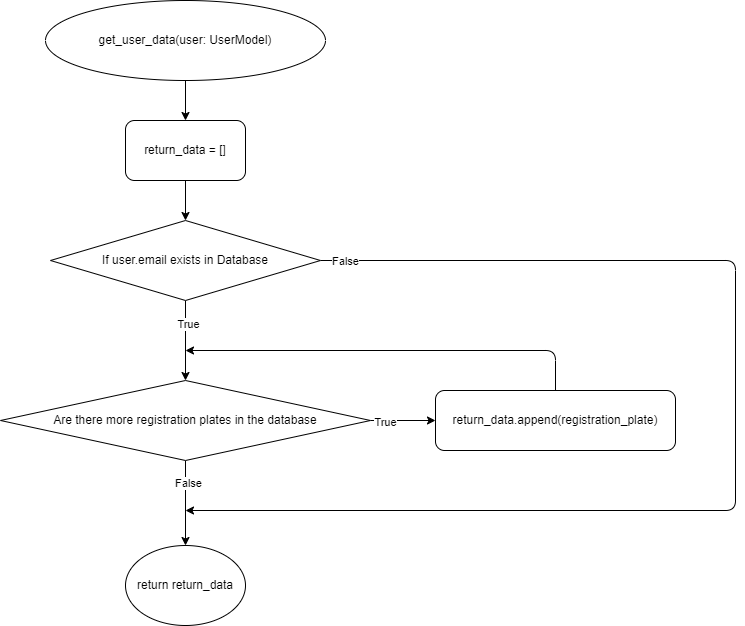
### 4.1.2 Сервизни услуги

На Фигура 20 е показана диаграмата на потока на логиката за сервизната услуга, която създава нов потребител. Функцията изпълнява две проверки, които валидират e-mail адресът на потребителя. Ако проверките са успешни, се създава нов потребител. Ако някоя проверка се провали, се връща грешка и съответно съобщение.



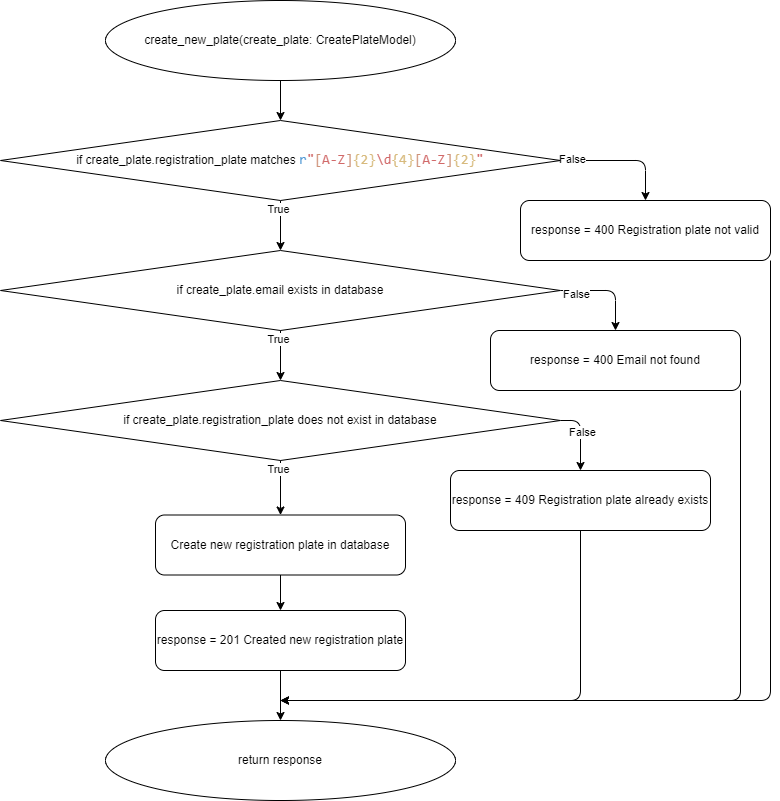
Фигура 20 Диаграма на потока на сервизна услуга за създаване на потребител

На Фигура 21 е показана диаграмата на потока на сервизната услуга за извличане на данните за конкретен потребител. Услугата проверява дали потребителят съществува в базата данни и ако това е така, то услугата връща регистрационните номера, асоциирани с потребителския профил.



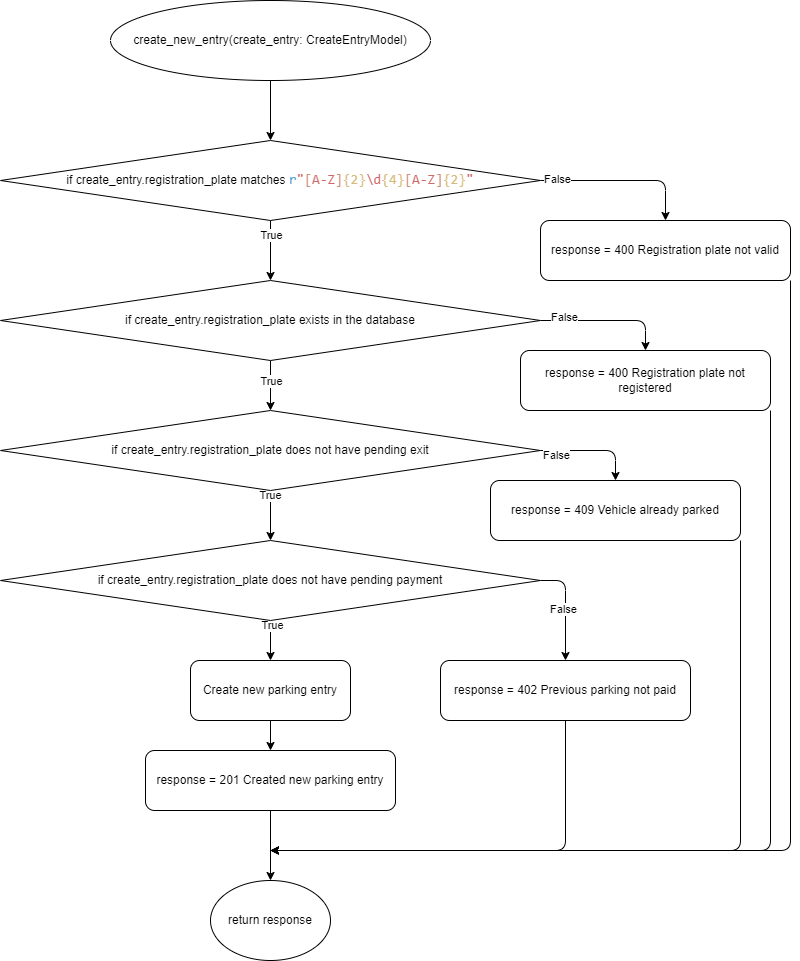
Фигура 21 Диаграма на потока за сервизна услига за вземане на данни за потребител

На Фигура 22 е показана диаграмата на потока на сервизната услуга за създаване на нов регистрационен номер. Услугата изпълнява три проверки, с които цели да се увери, че регистрационният номер е валиден, че потребителят е регистриран в приложението и, че номерът не е вече въведен в базата данни. Ако проверките минат успешно, регистрационният номер се добавя в базата данни и се асоциира с конкретния потребител.



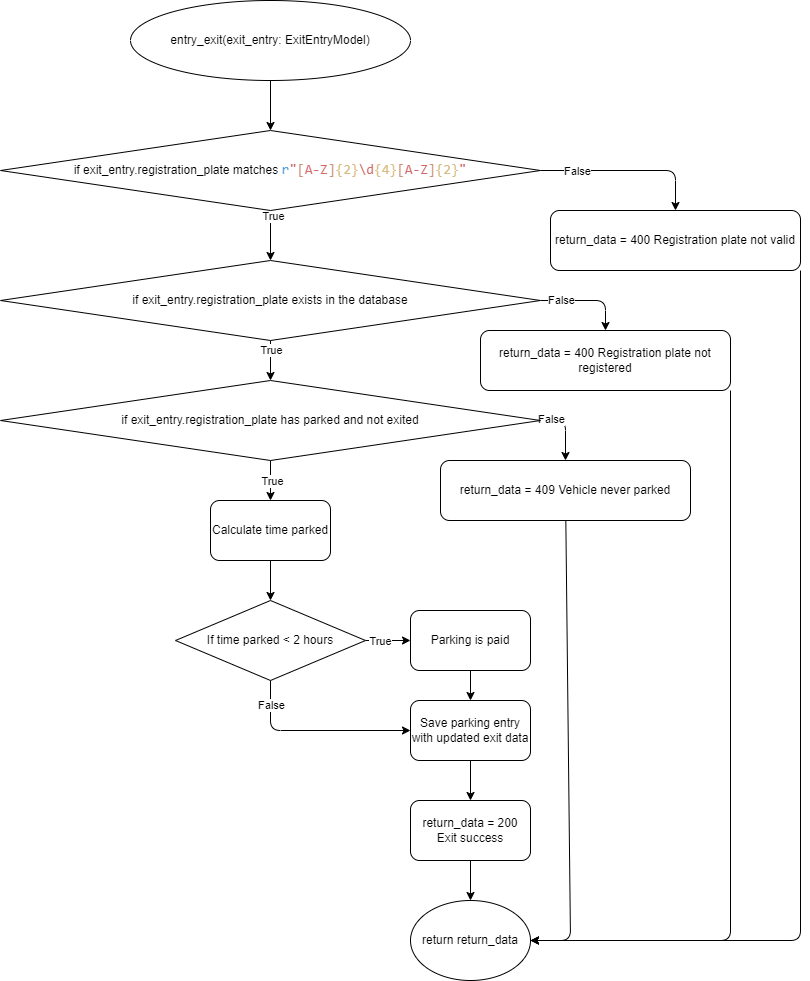
Фигура 22 Диаграма на потока на сервизна услуга за създаване на регистрационен номер

На Фигура 23 е показана диаграмата на потока на сервизната услуга за вход в паркинга. Услугата изпълнява четири проверки, преди да създаде нов вход в паркинга и да позволи преминаване. Услугата проверява дали регистрационният номер е валиден, дали е регистриран в системата, дали вече не е влязъл и дали е платен предходния паркинг. Ако всичко е успешно, преминаването е позволено.



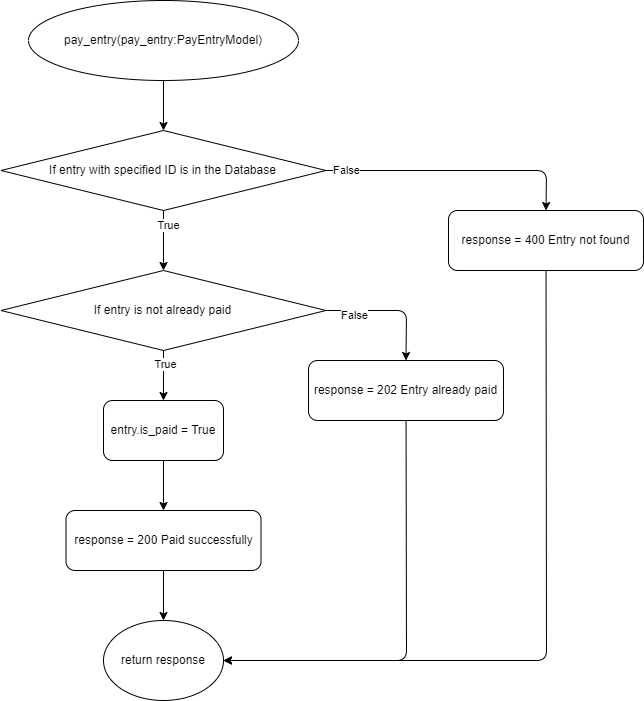
Фигура 23 Диаграма на потока на сервизна услуга за вход в паркинга

На Фигура 24 е показана диаграмата на потока на сервизната услуга за изход от паркинга. Услугата изпълнява 3 основни проверки и една допълнителна, която определя дали паркингът е платен. Първо се проверява дали регистрационният номер е валиден. След това проверяваме дали регистрационният номер е регистриран в базата данни. Следва проверка дали автомобилът е влязъл и не е излязъл. Ако тези проверки са успешни, автомобилът може да излезе. Следва доппълнителна проверка, която проверява дали престоят е под 2 часа. Ако е, то престоят е безплатен и автоматично се маркира като платен.



Фигура 24 Диаграма на потока на сервизна услуга за изход от паркинга

На Фигура 25 е показана диаграма на потока на сервизна услуга за плащане. Услугата изпълнява две проверки. Първо проверява дали записът, който се опитваме да платим, наистина съществува в базата данни. Ако той съществува, услугата проверява дали вече не е заплатен паркингът. Ако паркингът не е платен, то тогава се променя статусът му на „платен“.



Фигура 25 Диаграма на потока на сервизна услуга за плащане

На Фигура 26 е показана диаграма на потока на сервизна услуга за получаване на данни за вход в паркинг за конкретен регистрационен номер. Услугата изважда от базата данни всички данни за вход в паркинга за указания регистрационнен номер и ги връща като отговор.



Фигура 26 Диаграма на потока на сервизна услуга за получаване на данни за вход в паркинг на конкретен регистрационен номер

## 4.2 Реализация на мобилно приложение

Мобилното приложение е изградено на .NET MAUI. Тази платформа ни позволява да съставяме приложения, които могат да работят на множество платформи, използвайки един и същ програмен код. Приложението няма да променя как изглежда, тъй като MAUI създава потребителски интерфейс, който се променя динамично според размерът на екрана на потребителското устройство.

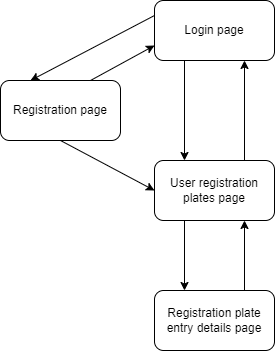
Приложението има четири страници. При зареждазне, се отваря страницата за вход на потребителят в приложението. Там се изисква да бъдат въведени e-mail адрес и парола за вход. Ако потребителят не е регистриран, той може да натисне бутон, който да го отведе на страница, която позволява регистрация на нови потребители.

Страницата за регистрация на нови потребители изисква да бъде въведено потребителско име, e-mail адрес, парола и паролата да бъде потвърдена, чрез повторно въвеждане.

След успешно въведени данни или на екранът за вход, или на екранът за регистрация, потребителят е отведен на страницата за визуализиране на регистрационните номера, които той е регистрирал в приложението. На тази страница има изобразен списък със всички регистрационни номера, които са всъщност са бутони, които водят към следващата страница. Най-отдолу на страницата има бутон за добавяне на нов регистрационен номер.

Последната страница се достъпва след като потребител избере регистрационен номер. На тази страница се визуализират всички записи за вход в паркинга, които са асоциирани с конкретния регистрационен номер. Показани са данните за време и дата на вход, време и дата на изход (ако автомобилът е напуснал паркинга), бутон за плащане, ако паркингът не е платен или текст, който уведомява потребителят, че паркингът вече е платен. Освен това има текст, който показва уникалният идентификационен номер на входът в паркинга. Това число започва от 1 и се увелчава с всеки вход в паркинга, независимо кой потребител е влязъл.

На Фигура 27 е показан общият изглед на страниците на мобилното приложение и връзките между тях. Всвяка страница позволява връщане на предишната. Страницата, показваща данните за всички регистрационни номера на даден потребител, може да бъде достъпена както от страницата за вход, така и от страницата за регистрация. При опит за връщане, обаче, потребителят ще бъде върнат на страницата за вход, независимо от къде е дошъл преди това.



Фигура 27 Общ изглед на страниците на мобилното приложение

### 4.2.1 Страница за вход

### 4.2.2 Страница за регистрация

### 4.2.3 Страница за общ преглед на регистрационни номера

### 4.2.4 Страница за преглед на данни за конкретен регистрационен номер и плащане на престой

## 4.3 Реализация на приложение за разпознаване на регистрационни номера

### 4.3.1 Подготовка на входни данни

### 4.3.2 Трениране на модел

### 4.3.3 Реализация на разпознаване на регистрационен номер

### 4.3.4 Визуализация на резултат

# V. Ръководство за използване и примери за употреба.

## 5.1 Регистрация и създаване на първи регистрационен номер

## 5.2 Преглед на данни за вход на автомобил, чрез потребителско приложение

## 5.3 Вход на автомобил, който е регистриран в системата и няма задължения

## 5.4 Вход на автомобил, който е регистриран в системата, но има задължения

## 5.5 Вход на автомобил, който не е регистриран в системата

# VI. Заключение

# Съкращения

8

ASGI - Asynchronous Server Gateway Interface 9

HTTP - Hyper Text Transfer Protocol 14

JSON - JavaScript Object Notation 25

MAUI - Multi-platform App UI 8

OCR - Optical Character Recognition 10

YOLO - You Only Look Once 9

МПС - Моторни Превозни Средства 6

# Източници

|  |  |
| --- | --- |
| [1] | "What is MAUI," Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0. |
| [2] | "About PostgreSQL," PostgreSQL. [Online]. |
| [3] | "YOLOv8," Ultralytics, [Online]. Available: https://github.com/ultralytics/ultralytics. |