



**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 .....                                                     | 6  |
| I. Увод .....                                                             | 7  |
| II. Постановка на дипломната работа, цели и задачи.....                   | 8  |
| 2.1 Мотивация .....                                                       | 8  |
| 2.2 Цели и задачи .....                                                   | 8  |
| 2.3 Ограничения .....                                                     | 8  |
| 2.4 Кратко описание на технологиите .....                                 | 9  |
| 2.4.1 Python.....                                                         | 9  |
| 2.4.2 C# - .NET MAUI .....                                                | 9  |
| 2.4.3 PostgreSQL .....                                                    | 9  |
| 2.4.4 SQLAlchemy.....                                                     | 10 |
| 2.4.5 FastAPI.....                                                        | 10 |
| 2.4.6 Uvicorn.....                                                        | 10 |
| 2.4.7 Разпознаване на обекти .....                                        | 10 |
| 2.4.8 YOLOv8.....                                                         | 11 |
| 2.4.9 CVAT.AI.....                                                        | 11 |
| 2.4.10 Оптично Разпознаване на Символи.....                               | 11 |
| 2.4.11 Aiven .....                                                        | 11 |
| 2.4.12 Docker .....                                                       | 11 |
| 2.4.13 Kubernetes.....                                                    | 12 |
| 2.4.14 Google Cloud .....                                                 | 12 |
| 2.5 Разпознаване на обекти и машинно самообучение .....                   | 12 |
| III. Функционално описание.....                                           | 15 |
| 3.1 Общ Преглед на системата .....                                        | 15 |
| 3.2 Описание на обслужващия сървър.....                                   | 16 |
| 3.3 Описание на базата данни.....                                         | 17 |
| 3.4 Описание на потребителско приложение.....                             | 17 |
| 3.5 Описание на приложение за разпознаване на регистрационни номера ..... | 22 |
| IV. Програмна реализация. ....                                            | 25 |
| 4.1 Реализация на сървър, обслужващ базата данни .....                    | 25 |
| 4.1.1 Входни точки .....                                                  | 25 |
| 4.1.2 Сервизни услуги .....                                               | 30 |

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| 4.2 Реализация на мобилно приложение .....                                                      | 37 |
| 4.2.1 Страница за вход .....                                                                    | 38 |
| 4.2.2 Страница за регистрация .....                                                             | 39 |
| 4.2.3 Страница за общ преглед на регистрационни номера .....                                    | 39 |
| 4.2.4 Страница за преглед на данни за конкретен регистрационен номер и плащане на престой ..... | 39 |
| 4.3 Реализация на приложение за разпознаване на регистрационни номера.....                      | 39 |
| 4.3.1 Подготовка на входни данни за модела .....                                                | 41 |
| 4.3.2 Трениране на модел .....                                                                  | 41 |
| 4.3.4 Визуализация на резултат.....                                                             | 44 |
| V. Ръководство за използване и примери за употреба. ....                                        | 45 |
| 5.1 Регистрация и създаване на първи регистрационен номер.....                                  | 45 |
| 5.2 Преглед на данни за вход на автомобил, чрез потребителско приложение.....                   | 45 |
| 5.3 Вход на автомобил, който е регистриран в системата и няма задължения.....                   | 46 |
| 5.4 Вход на автомобил, който е регистриран в системата, но има задължения .....                 | 46 |
| 5.5 Вход на автомобил, който не е регистриран в системата .....                                 | 46 |
| VI. Заключение .....                                                                            | 47 |
| Съкращения.....                                                                                 | 48 |
| Източници .....                                                                                 | 49 |

## СПИСЪК С ФИГУРИ

|           |                                                                                                                        |    |
|-----------|------------------------------------------------------------------------------------------------------------------------|----|
| Фигура 1  | Общ преглед на системата .....                                                                                         | 15 |
| Фигура 2  | Интерфейс за създаване на потребител .....                                                                             | 16 |
| Фигура 3  | Клъстър на Kubernetes .....                                                                                            | 16 |
| Фигура 4  | Три инстанции на контейнера със сървър .....                                                                           | 17 |
| Фигура 5  | База данни в Aiven .....                                                                                               | 17 |
| Фигура 6  | Примерна страница за регистрация на потребител .....                                                                   | 18 |
| Фигура 7  | Примерен екран за вход .....                                                                                           | 19 |
| Фигура 8  | Основен потребителски екран .....                                                                                      | 20 |
| Фигура 9  | Екран с детайли за регистрационен номер .....                                                                          | 21 |
| Фигура 10 | Интеракция на потребител с приложение и на приложение със сървър .....                                                 | 22 |
| Фигура 11 | Анотиран регистрационен номер [15] .....                                                                               | 23 |
| Фигура 12 | Процес по разпознаване на регистрационен номер .....                                                                   | 24 |
| Фигура 13 | Входна точка за създаване на нов потребител .....                                                                      | 26 |
| Фигура 14 | Входна точка за вземане на данни на конкретен потребител .....                                                         | 26 |
| Фигура 15 | Входна точка за създаване на нов регистрационен номер .....                                                            | 27 |
| Фигура 16 | Входна точка за вход в паркинга .....                                                                                  | 27 |
| Фигура 17 | Входна точка за изход от паркинга .....                                                                                | 28 |
| Фигура 18 | Входна точка за заплащане на влизане .....                                                                             | 29 |
| Фигура 19 | Входна точка за вземане на данни за вход .....                                                                         | 29 |
| Фигура 20 | Диаграма на потока на сервизна услуга за създаване на потребител .....                                                 | 30 |
| Фигура 21 | Диаграма на потока за сервизна услуга за вземане на данни за потребител .....                                          | 31 |
| Фигура 22 | Диаграма на потока на сервизна услуга за създаване на регистрационен номер .....                                       | 32 |
| Фигура 23 | Диаграма на потока на сервизна услуга за вход в паркинга .....                                                         | 33 |
| Фигура 24 | Диаграма на потока на сервизна услуга за изход от паркинга .....                                                       | 35 |
| Фигура 25 | Диаграма на потока на сервизна услуга за плащане .....                                                                 | 36 |
| Фигура 26 | Диаграма на потока на сервизна услуга за получаване на данни за вход в паркинг на конкретен регистрационен номер ..... | 37 |
| Фигура 27 | Общ изглед на страниците на мобилното приложение .....                                                                 | 38 |
| Фигура 28 | Начин на работа на приложението .....                                                                                  | 40 |
| Фигура 29 | Матрица на обръкването на модела .....                                                                                 | 42 |
| Фигура 30 | Отношение на прецизност към увереност .....                                                                            | 43 |
| Фигура 31 | Валидационни резултати на модела .....                                                                                 | 43 |
| Фигура 32 | Визуализация на резултат от приложението .....                                                                         | 44 |

## I. УВОД

Паркингът е част от нашето ежедневие, която е позната на всеки собственик на моторно превозно средство (МПС). Това е място, предназначено за временно пребиваване на МПС. Паркингите, обикновено, не представляват особено технологична структура. Нормалният паркинг представлява голямо, отворено място, където да могат да се паркират превозни средства удобно. Обикновено има охранител, който се грижи МПС да са защитени от недоброжелатели. Охранителят също следи влизащите и излизащите автомобили, като също играе ролята на касиер, при който се плаща престоят на превозното средство 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. Заради това, 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 добър вариант да се изграждат големи количества малки 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 нуждата 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 може да стане лесно, тъй като всяко превозно средство има уникален регистрационен номер, който е поставен на видно място върху него. Интелигентната паркинг система трябва да използва система от камери, която да наблюдава автомобилите и да проверява регистрационните им номера спрямо база данни от регистрирани потребители и така да ограничава достъпа на неоторизирани лица. Използвайки тази система за достъп, е възможно да разберем точния момент на вход и на изход на дадено МПС и да ги запишем в база данни. Така можем да позволим на 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 се адресират проблемите със забавянето при паркингите
- Да се предостави лесен и удобен начин на потребителите да взаимодействат с паркингите.

Тези цели могат да бъдат постигнати чрез приложение за всяка широко използвана операционна система, което да предоставя потребителски интерфейс за интеракция с паркинг системата. Това приложение ще позволява да бъде регистриран потребител и след това, потребителят да може сам да добави автомобила си. Приложението трябва да предоставя на потребителите лесен начин за плащане на паркинга, така че всеки да може да заплати в удобно за него време.

Освен мобилното приложение, трябва да има паркинг система, която ще използва система от камери, които да определят кога доближава автомобил и дали този автомобил е в списъкът на регистрираните автомобили. След като автомобил е добавен, автоматизираното управление на паркингът трябва да започне да отваря входът и изходът, когато регистрираният автомобил се доближи достатъчно.

### **2.3 Ограничения**

За разработката на тази система е необходимо да се съобразим с някои ограничения. Един от най-големите проблеми, който изниква при разпознаване на текст, чрез камера, е свързан с яснотата на изображението, заснето от камерата. Това изображение трябва да бъде ясно, за да може системата да определи текста, записан върху регистрационния 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 потребителя 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 съоръжението.

## **2.4 Кратко описание на технологиите**

### **2.4.1 Python**

Избраният език за реализация на голяма част от тази разработка е Python. Езикът ни дава възможност за лесна работа с голямо количество библиотеки и структури от данни. Това позволява 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. Python е предпочитан за използване при работа с изкуствен интелект, заради простия си синтаксис. Това прави програми, написани на този език, лесни за четене и поддръжка. Езикът предоставя гъвкави инструменти за работа, голямо количество от библиотеки и голямо общество от разработчици, които го поддържат [1].

### **2.4.2 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 програмен код. MAUI 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 [2].

### **2.4.3 PostgreSQL**

PostgreSQL е система за управление на релационни бази данни с отворен код. Системата използва и разширява езика 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 [3].

#### **2.4.4 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 сървър на базата, порт на сървър 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 [4].

#### **2.4.5 FastAPI**

FastAPI представлява библиотека на Python, която позволява изграждане на интерфейси за достъп до сървър. Библиотеката работи директно с езика Python, което значи, че потребителят просто трябва да посочи функция, която иска да използва като интерфейс за достъп, и библиотеката FastAPI сама се грижи за изграждането на интерфейса, като взема параметрите от прототипа на функцията.

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 [5].

#### **2.4.6 Uvicorn**

Uvicorn е уеб сървър за Python. Представлява имплементация на Asynchronous Server Gateway Interface (ASGI) спецификацията. Тази спецификация описва интерфейси за изграждане на асинхронни и синхронни сървъри, използвайки 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 сървър, който да обслужва входните точки [6].

#### **2.4.7 Разпознаване на обекти**

Разпознаване на обекти или Object Detection представлява технология, при която се тренира невронна мрежа, която да разпознава дадени обекти. Това 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 разработка [7].

#### **2.4.8 YOLOv8**

YOLOv8 или You Only Look Once представлява модел за разпознаване на обекти, създаден от Ultralytics, който е бърз, точен и лесен за употреба. Моделът може да класифицира, да разпознава, да сегментира, 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 extra. [8]

#### **2.4.9 CVAT.AI**

Това е платформа за аотиране на данни. Аотацията представлява процес, при който потребител минава през сет от данни с цел да им даде някакво определение. В тази дипломна работа, са аотирани повече от 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 аотиране на данни. Платформата е безплатна за ползване при по-малки проекти [9].

#### **2.4.10 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 низ, 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 [10].

#### **2.4.11 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 [11].

#### **2.4.12 Docker**

Docker е инструмент, 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, върху която работи Docker. Инструментът е безплатен и е широко използван в индустрията [12].

#### **2.4.13 Kubernetes**

Kubernetes е система за оркестрация на контейнери. Тази система се грижи за работата на контейнерите, които са пуснати в нея. Управлява ресурсите на компютъра, 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 [13].

#### **2.4.14 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 [14].

### **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 този тип 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. Скрытите 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. Тренировката на мрежата се състои от две части: същинското трениране и валидацията.

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 неколkokратно 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 тренировъчния 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 принципа проба-грешка.

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 обработи данните, трябва да сравним изхода на алгоритъма с очаквания от нас изход. Ако резултатът, който сме получили е задоволителен, значи мрежата е готова за използване. Ако видим разлики между очаквания резултат и получения резултат, следва да тренираме мрежата отново. При последващи тренирания на мрежата, трябва да бъдат направени промени, за да не получим същия 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 броя 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. Друг 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 случва 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 нагледени, за да получим оптимален резултат с нашите данни.

При разпознаването на обекти се използва много подобен подход. Трябва да създадем пакет от данни, който да представя обектите, които търсим. Колкото е по-голям пакетът, толкова по-добре ще бъде обучена мрежата. Важно е пакетът да е не-само голям, а и да бъде разнообразен. Трябва да предоставим ясни изображения, които да са добре

анотирани. По този начин, чрез разнообразни и добре анотирани изображения, ще получим по-генерализиран модел, който ще може да разпознава обекти по-добре. След това трябва да тренираме невронната мрежа, използвайки тези данни, подобно на примера с цифрите.

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 характерни черти на обектите и чрез тях да може да разпознае самият обект. Това работи добре, когато обектите са ясно изразени и видими добре. При замъглени обекти, където характеристиките им не са ясно отличени, може да има проблем, при който не може да бъде разпознат обектът. Заради начина, 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 които 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

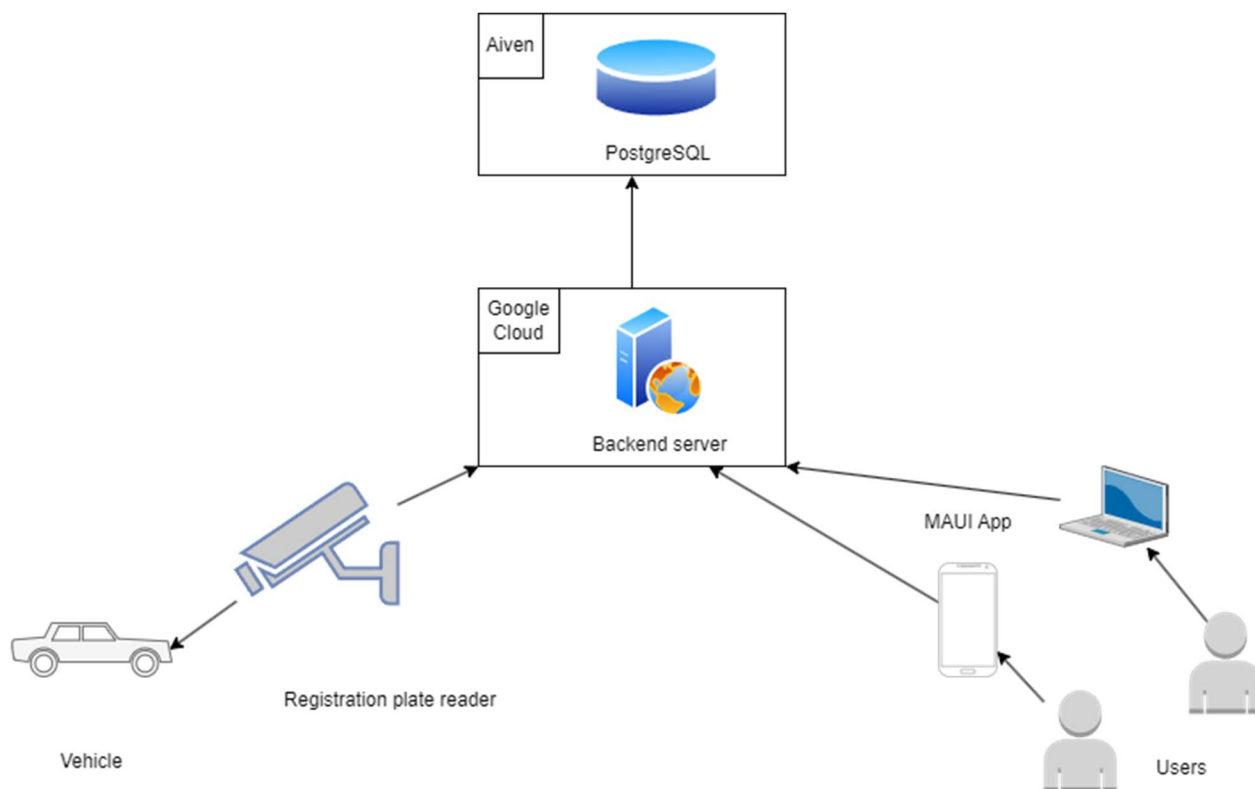
Системата се състои от 4 основни части: потребителският интерфейс, написан на .NET MAUI, софтуерът за разпознаване на регистрационни номера, написан на Python, сървър, обслужващ базата данни, написан на Python, и база данни PostgreSQL.

Комуникацията между обслужващия сървър и двете приложения се случва чрез POST заявки по 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 директно 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 обслужващия сървър

Реализацията на обслужващия 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 използва 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 нашия интерфейс.



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.

| <input type="checkbox"/> Status     | Name ↑                               | Location          | Number of nodes | Total vCPUs | Total memory | Notifications | Labels |
|-------------------------------------|--------------------------------------|-------------------|-----------------|-------------|--------------|---------------|--------|
| <input checked="" type="checkbox"/> | <a href="#">parkingbuddy-cluster</a> | europe-central2-c | 3               | 6           | 12 GB        | —             | ⋮      |

ФИГУРА 3 КЛЪСТЪР НА KUBERNETES

В този клъстер, ще бъде пуснат Docker контейнер, който съдържа изображението на програмата на сървъра, заедно с всички необходими зависимости, 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.



#### Managed pods

| Revision          | Name ↑                                        | Status    | Restarts | Created on ↑             |
|-------------------|-----------------------------------------------|-----------|----------|--------------------------|
| <a href="#">7</a> | <a href="#">parkingbuddy-78c89474c7-sw9cp</a> | ✓ Running | 0        | May 31, 2024, 2:41:01 PM |
| <a href="#">7</a> | <a href="#">parkingbuddy-78c89474c7-ff679</a> | ✓ Running | 0        | May 31, 2024, 2:45:29 PM |
| <a href="#">7</a> | <a href="#">parkingbuddy-78c89474c7-j74fx</a> | ✓ Running | 0        | May 31, 2024, 2:48:36 PM |

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.

| Service                                                                                                                    | Nodes                                                                                       | Plan                                          | Cloud                                |
|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|-----------------------------------------------|--------------------------------------|
|  <b>parking</b><br>PostgreSQL • Running |  Nodes 1 | Free-1-5gb<br>1 CPU / 1 GB RAM / 5 GB storage | DigitalOcean: fra<br>Europe, Germany |

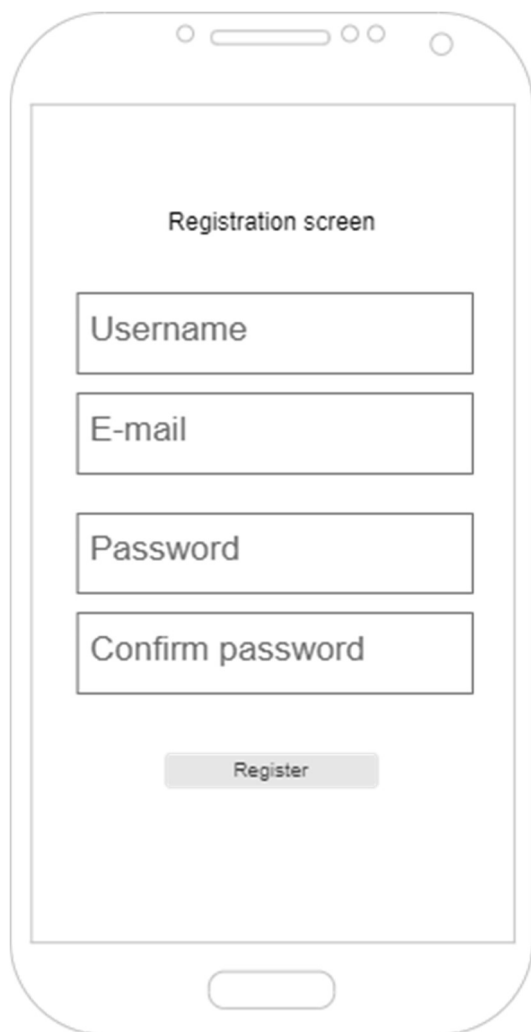
ФИГУРА 5 БАЗА ДАННИ В AIVEN

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

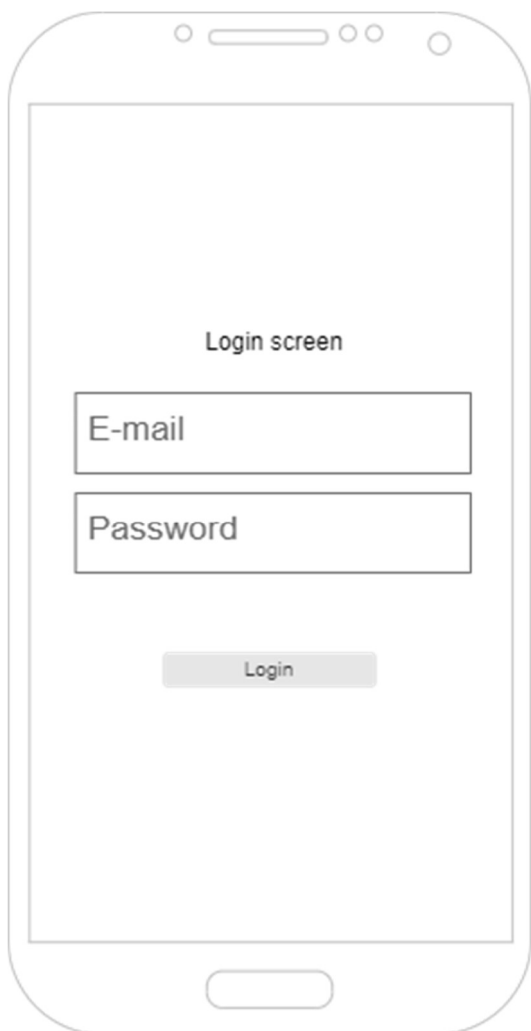
Приложението, предоставящо потребителският интерфейс, ще бъде изготвено на .NET MAUI. Реализацията, чрез MAUI ни позволява да изградим приложение, което може да работи на всички често използвани операционни системи. Приложение, изградено на MAUI, може да работи под операционни системи като Android, Windows и macOS. По този начин, ние не ограничаваме нашите потребители до определени операционни системи. Не се и налага да поддържаеме 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. Страницата на приложението, която се занимава със създаване на потребителски профил, трябва да изисква потребителско име, e-mail адрес, парола и да изисква потвърждение 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 асоциирана с него. Примерен екран за вход е показан на Фигура 7.



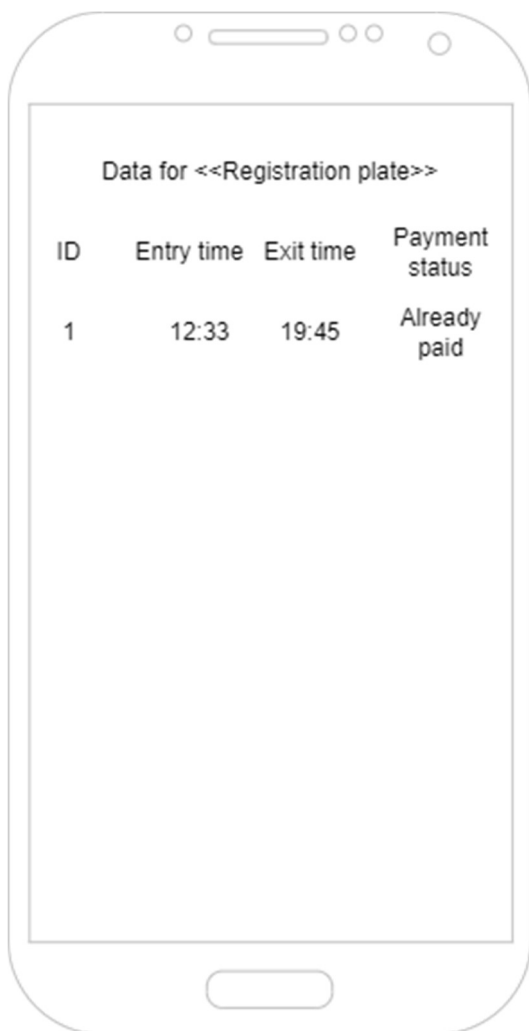
**ФИГУРА 7 ПРИМЕРЕН ЕКРАН ЗА ВХОД**

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**

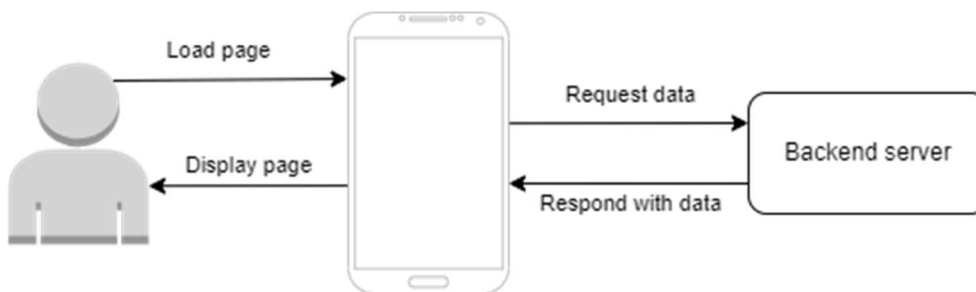
След като потребителят е избрал даден регистрационен номер, приложението ще отвори страница, която ще покаже всички данни, с които разполага паркинг системата, за съответния регистрационен номер. Системата трябва да покаже номера на входа, дата и час на вход, дата и час на изход и да даде възможност на потребителят да заплати своя престой на паркинга. Ако престоят вече е платен, приложението трябва да информира потребителя, че не е необходимо допълнително плащане. 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 модела за разпознаване на регистрационни номера.

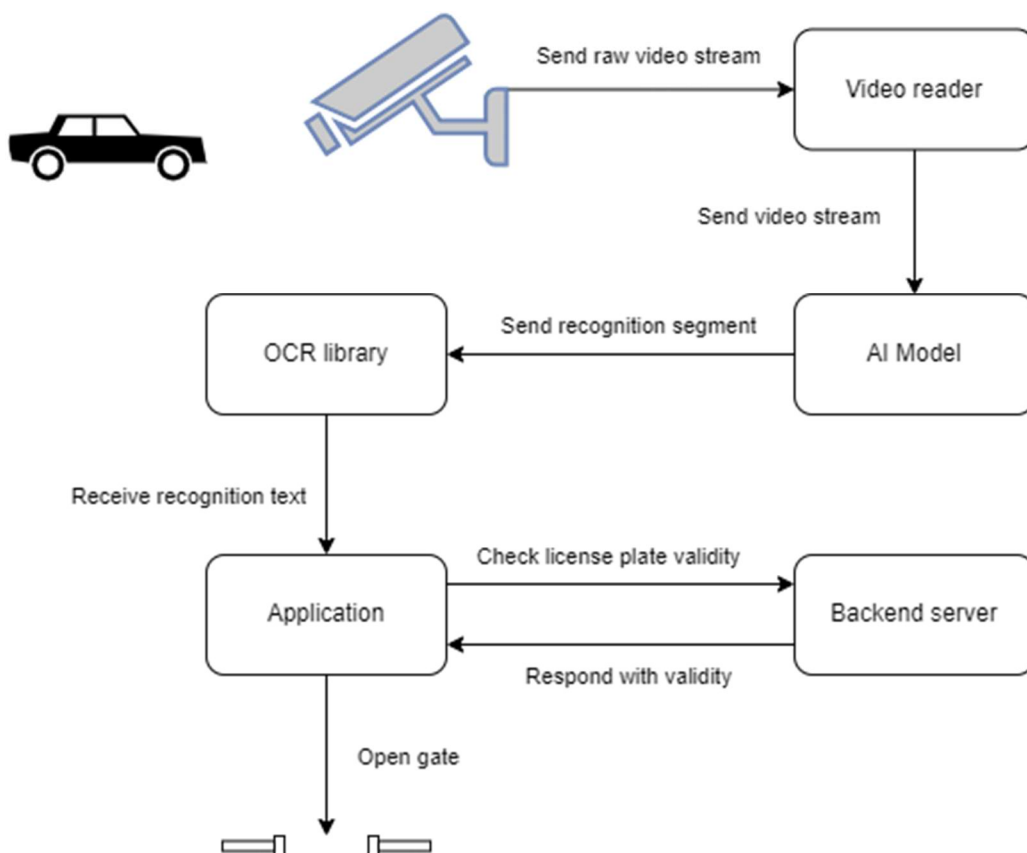
За тренирането на модела, ще използваме библиотеката Ultralytics, с помощта на която ще тренираме модел YOLOv8. Тези модели ни позволяват да засичаме, проследяваме и класифицираме обекти от видеа и снимки. Моделите идват с различни размери, като колкото по-голям е моделът, толкова по-добре се справя с разпознаването. Важно е да се отбележи, 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 за български автомобили, който да е публично достъпен, се налага да бъде изграден нов, използвайки снимки, налични в интернет. 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.



**ФИГУРА 11 АНОТИРАН РЕГИСТРАЦИОНЕН НОМЕР [15]**

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 тренирания 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 сървър, 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 престоя 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 две основни части:

- Входни точки
- Сервизни услуги

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.

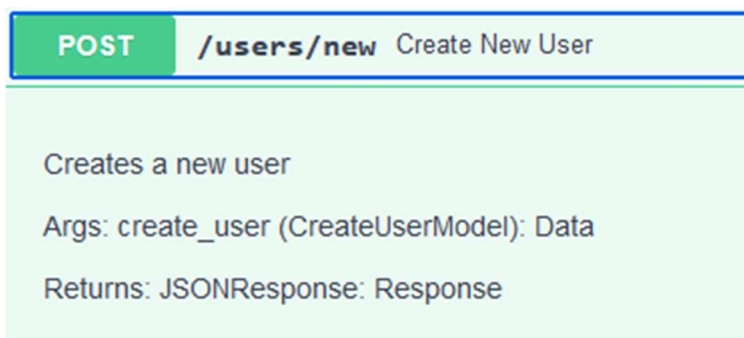
Сервизните услуги представляват основната функционалност. Всяка сервизна услуга отговаря на дадена входна точка на програмата. Сервизната услуга борава с данните, които е получила от входната точка, като също така има директен достъп до базата данни, за да може да борава и с информацията от там.

Приложението работи с типове данни, 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 адреса 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:

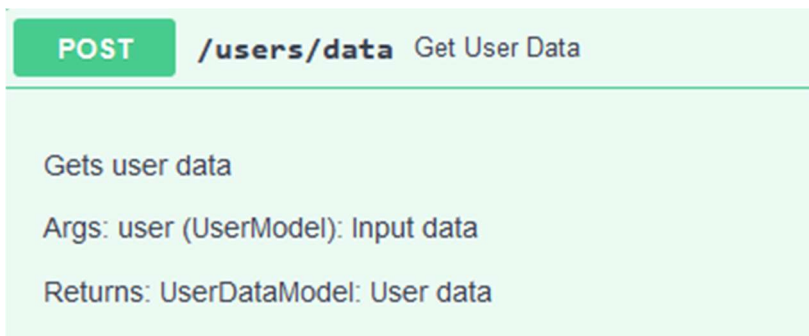
- 201 - Created new user – При успешно създаден нов потребител
- 400 - Invalid data provided for user – При невалидни подадени данни
- 409 - User with provided data already exists – При дублиране на потребител



**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 адреса 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:

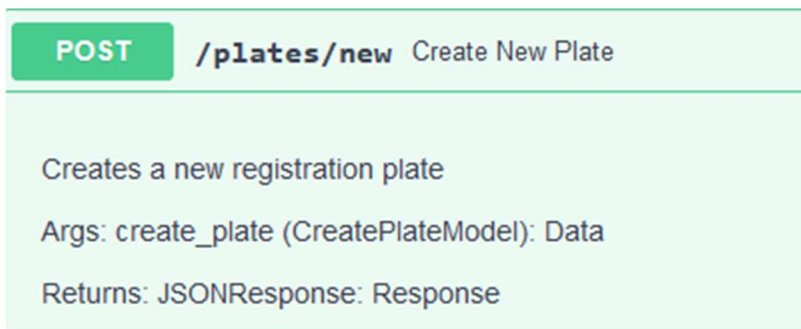
- 200 - Got user data – При успешно извлечена информация
- 400 - Invalid data provided for user – При невалидни подадени данни



**ФИГУРА 14 ВХОДНА ТОЧКА ЗА ВЗЕМАНЕ НА ДАННИ НА КОНКРЕТЕН ПОТРЕБИТЕЛ**

На Фигура 15 Фигура 14 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 адреса 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 създаване на регистрационен номер. 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:

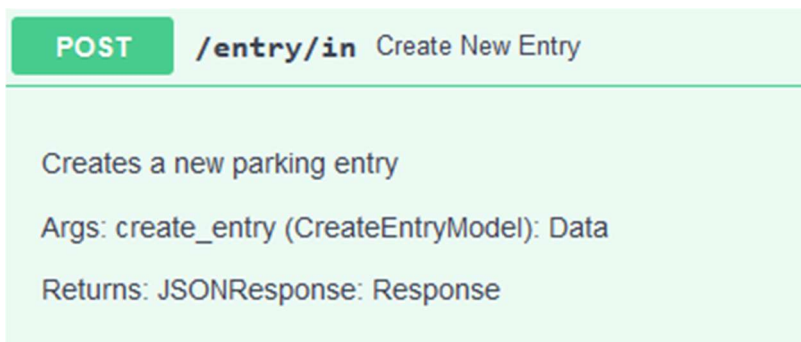
- 201 - Created new registration plate – Създаден е нов регистрационен номер
- 400 - Invalid data provided – Невалидни данни са подадени
- 409 - Plate with provided data already exists – Регистрационният номер вече съществува



**ФИГУРА 15 ВХОДНА ТОЧКА ЗА СЪЗДАВАНЕ НА НОВ РЕГИСТРАЦИОНЕН НОМЕР**

На Фигура 16 Фигура 14 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 адреса 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 създаване на вход. 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

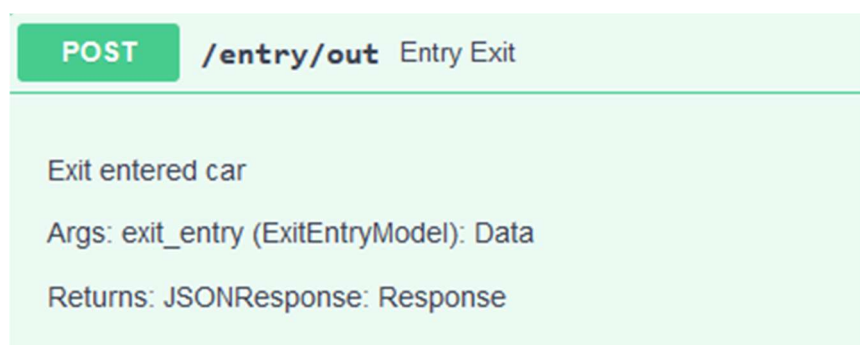


**ФИГУРА 16 ВХОДНА ТОЧКА ЗА ВХОД В ПАРКИНГА**

На Фигура 17 Фигура 14 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 адреса 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 изход от паркинга. 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:

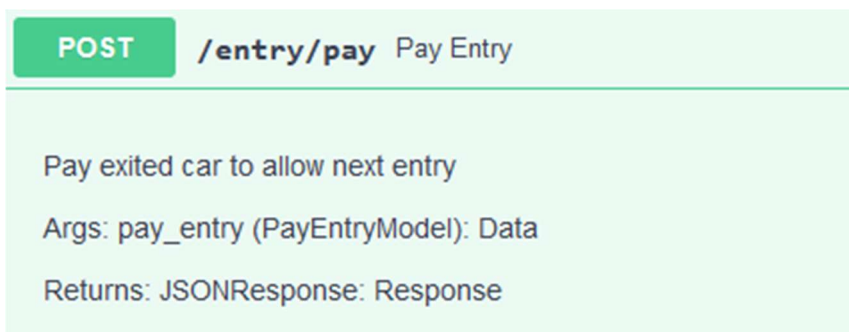
- 200 - Exited successfully – Успешен изход от паркинга
- 400 - Invalid data – Невалидни данни са подадени
- 409 - Plate with provided data never parked – Регистрационният номер не е паркиран



ФИГУРА 17 ВХОДНА ТОЧКА ЗА ИЗХОД ОТ ПАРКИНГА

На Фигура 18 Фигура 14 е показана входната точка за заплащане, както е визуализирана от OpenAPI документа. От фигурата можем да видим, че входната точка е достъпна на адреса 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 плащане на преден вход. 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:

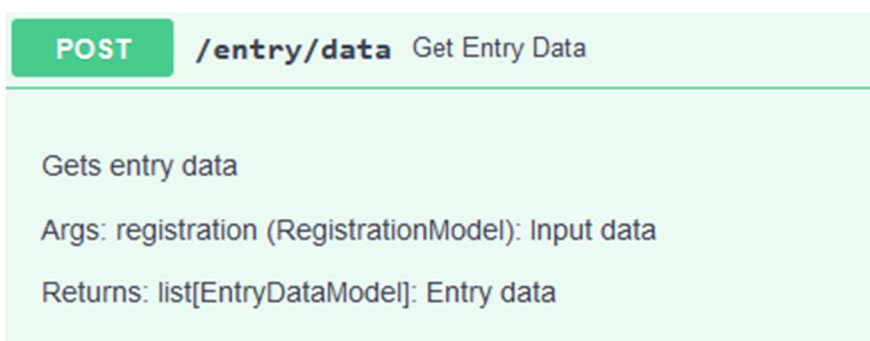
- 200 - Paid successfully – Успешно заплащане на вход
- 202 - Entry already paid – Следващо влизане вече е платено
- 400 - Invalid data – Невалидни данни са подадени



ФИГУРА 18 ВХОДНА ТОЧКА ЗА ЗАПЛАЩАНЕ НА ВЛИЗАНЕ

На Фигура 19 Фигура 14 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 адреса 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 вход. 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:

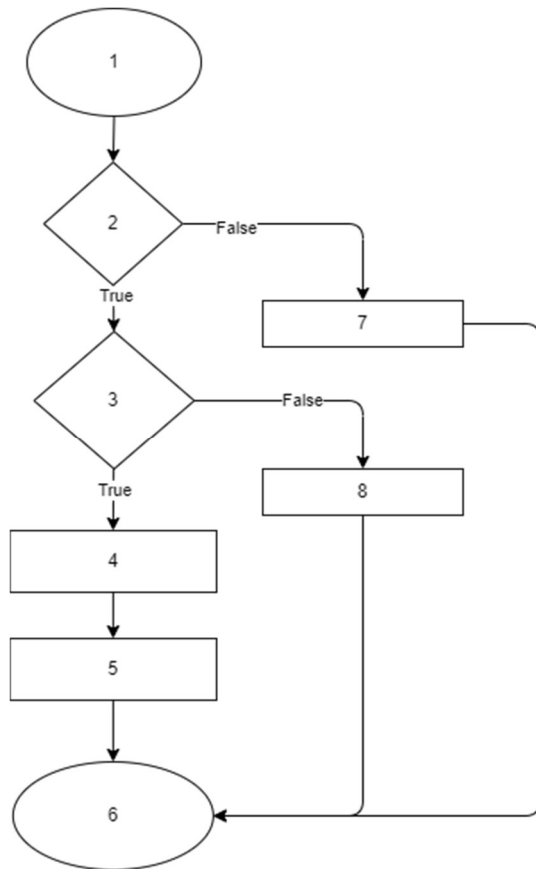
- 200 - Got entry data – При успешно взети данни за вход
- 400 - Invalid data provided for entry – Когато не е открит регистрационен номер в базата



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.



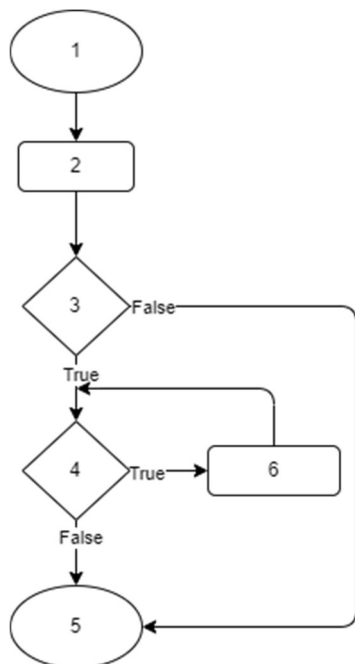
**ФИГУРА 20** ДИАГРАМА НА ПОТОКА НА СЕРВИЗНА УСЛУГА ЗА СЪЗДАВАНЕ НА ПОТРЕБИТЕЛ

В тази диаграма е описана следната функционалност:

1. Вход във функция: `create_new_user(create_user: CreateUserModel)`
2. Проверка на условие дали потребителският e-mail адрес съвпада с регулярен израз `r"[^@]+\@[^\@]+\.[^\@]+"`
  - a. Ако да, премини към 3
  - b. Ако не, премини към 7
3. Проверка на условие дали потребител с такъв e-mail адрес вече съществува
  - a. Ако не, премини към 4
  - b. Ако да, премини към 8
4. Създаване на нов потребител в базата данни
5. Изготвяне на отговор за успешно създаване на потребител
6. Изход от функция

7. Изготвяне на отговор за неуспешно създаване 400, поради невалиден e-mail адрес
8. Изготвяне на отговор за неуспешно създаване 409, поради e-mail адрес, който вече принадлежи 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.

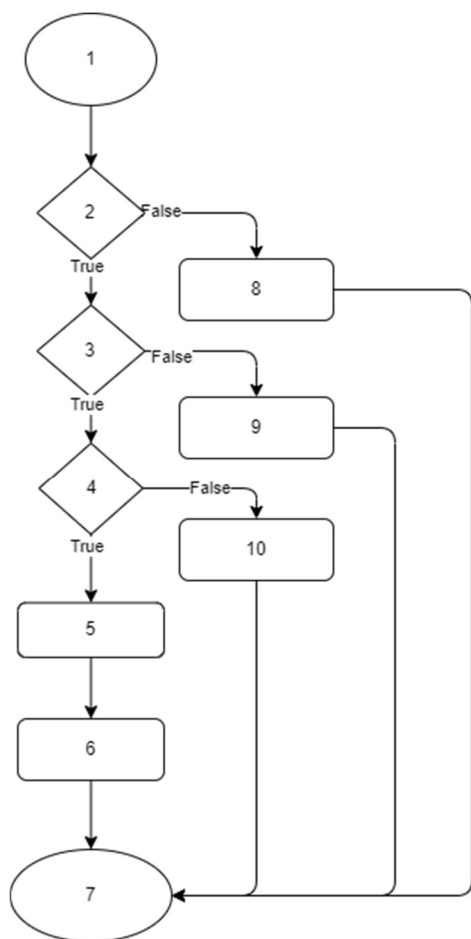


**ФИГУРА 21** ДИАГРАМА НА ПОТОКА ЗА СЕРВИЗНА УСЛИГА ЗА ВЗЕМАНЕ НА ДАННИ ЗА ПОТРЕБИТЕЛ

В тази диаграма е описана следната функционалност:

1. Вход във функция: `get_user_data(user: UserModel)`
2. Създаване на празен списък с данни, които да бъдат върнати
3. Проверка на условие дали потребителският e-mail съществува в базата данни
  - a. Ако да, премини към 4
  - b. Ако не, премини към 5
4. Проверка на условие дали в базата данни има регистрационни номера за потребителя
  - a. Ако да, премини към 6
  - b. Ако не, премини към 5
5. Изход от функция и връщане на данни
6. Допълване на изходните данни с регистрационния номер и връщане обратно в 4

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

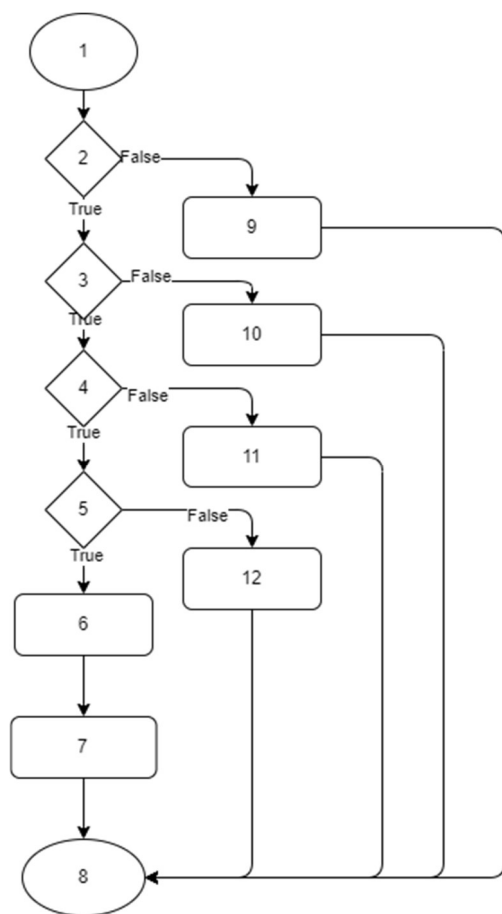
В тази диаграма е описана следната функционалност:

1. Вход във функция: `create_new_plate(create_plate: CreatePlateModel)`
2. Проверка дали регистрационният номер отговаря на регулярният израз `r"[A-Z]{2}\d{4}[A-Z]{2}"`
  - a. Ако да, премини към 3
  - b. Ако не, премини към 8
3. Проверка на условие дали потребителският e-mail съществува в базата данни
  - a. Ако да, премини към 4
  - b. Ако не, премини към 9
4. Проверка на условие дали в базата данни не е вече добавен този регистрационен номер



- a. Ако да, премини към 5
  - b. Ако не, премини към 10
5. Създаване на регистрационен номер в базата данни
6. Изготвяне на отговор за успешно създаване 201
7. Изход от функция
8. Изготвяне на отговор за неуспешно създаване 400, поради невалиден регистрационен номер
9. Изготвяне на отговор за неуспешно създаване 400, поради неоткрит e-mail адрес
10. Изготвяне на отговор за неуспешно създаване 409, поради вече съществуващ 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 предходният паркинг. Ако всичко е успешно, преминаването е позволено.

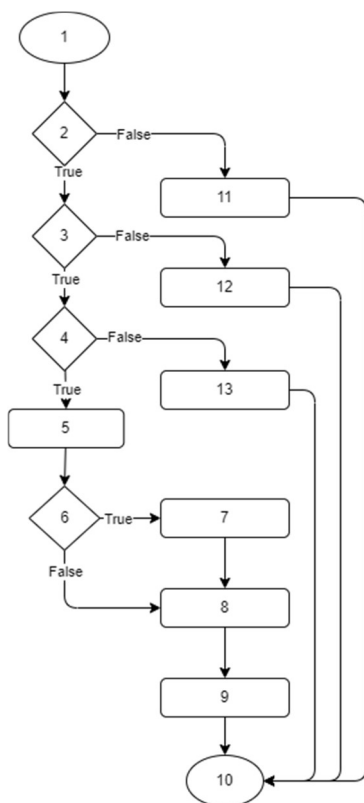


**ФИГУРА 23** ДИАГРАМА НА ПОТОКА НА СЕРВИЗНА УСЛУГА ЗА ВХОД В ПАРКИНГА

В тази диаграма е описана следната функционалност:

1. Вход във функция: `create_new_entry(create_entry: CreateEntryModel)`
2. Проверка дали регистрационният номер отговаря на регулярният израз `r"[A-Z]{2}\d{4}[A-Z]{2}"`
  - a. Ако да, премини към 3
  - b. Ако не, премини към 9
3. Проверка на условие дали регистрационният номер съществува в базата данни
  - a. Ако да, премини към 4
  - b. Ако не, премини към 10
4. Проверка на условие дали регистрационният номер е излязъл преди това
  - a. Ако да, премини към 5
  - b. Ако не, премини към 11
5. Проверка на условие, дали регистрационният номер е заплатил предния си престой
  - a. Ако да, премини към 6
  - b. Ако не, премини към 12
6. Създаване на нов запис за вход в базата данни
7. Изготвяне на отговор за успешно влизане 201
8. Изход от функция
9. Изготвяне на отговор за неуспешно създаване 400, поради невалиден регистрационен номер
10. Изготвяне на отговор за неуспешно създаване 400, поради нерегистриран номер
11. Изготвяне на отговор за неуспешно създаване 409, поради вече паркиран автомобил
12. Изготвяне на отговор за неуспешно създаване 402, поради неплатен престой

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 допълнителна проверка, която проверява дали престоят е под 2 часа. Ако е, то престоят е безплатен и автоматично се маркира като платен.



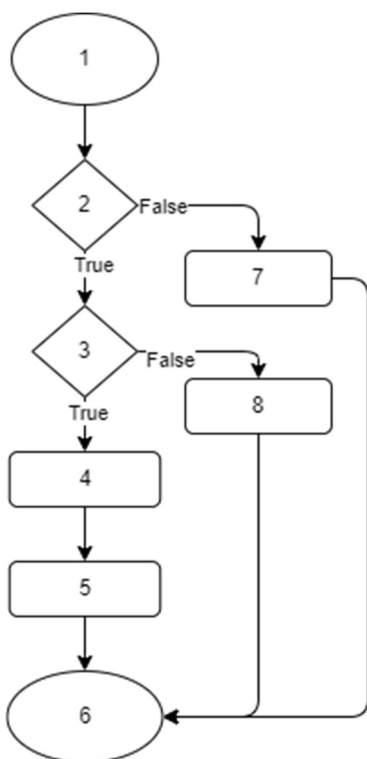
**ФИГУРА 24** ДИАГРАМА НА ПОТОКА НА СЕРВИЗНА УСЛУГА ЗА ИЗХОД ОТ ПАРКИНГА

В тази диаграма е описана следната функционалност:

1. Вход във функция: entry\_exit(exit\_entry: ExitEntryModel)
2. Проверка дали регистрационният номер отговаря на регулярният израз `r"[A-Z]{2}\d{4}[A-Z]{2}"`
  - a. Ако да, премини към 3
  - b. Ако не, премини към 11
3. Проверка на условие дали регистрационният номер съществува в базата данни
  - a. Ако да, премини към 4
  - b. Ако не, премини към 12
4. Проверка на условие дали регистрационният номер е излязъл преди това
  - a. Ако да, премини към 5
  - b. Ако не, премини към 13
5. Пресметни колко време е бил паркиран автомобилът
6. Проверка на условие, дали времето на паркиране е под 2 часа
  - a. Ако да, премини към 7
  - b. Ако не, премини към 8
7. Маркирай текущото паркиране като платено
8. Запиши данните за паркинг (време на изход, статус на плащане)
9. Изготвяне на отговор за успешно излизане 200
10. Изход от функция

11. Изготвяне на отговор за неуспешно излизане 400, поради невалиден регистрационен номер
12. Изготвяне на отговор за неуспешно излизане 400, поради нерегистриран номер
13. Изготвяне на отговор за неуспешно излизане 409, поради вече напуснал автомобил

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“.



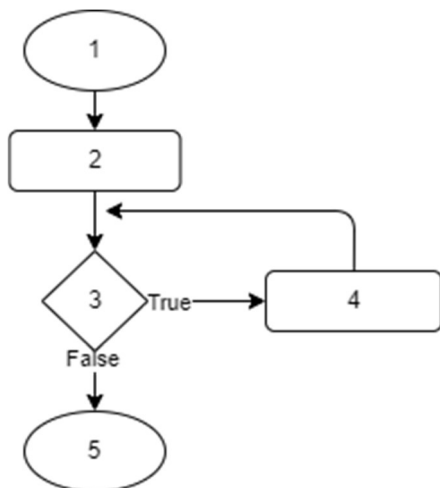
**ФИГУРА 25** ДИАГРАМА НА ПОТОКА НА СЕРВИЗНА УСЛУГА ЗА ПЛАЩАНЕ

В тази диаграма е описана следната функционалност:

1. Вход във функция: `pay_entry(pay_entry:PayEntryModel)`
2. Проверка дали паркинг записът с дадения идентификатор съществува в базата данни
  - a. Ако да, премини към 3
  - b. Ако не, премини към 7
3. Проверка на условие дали записът не е платен
  - a. Ако да, премини към 4
  - b. Ако не, премини към 8
4. Маркирай записа като платен
5. Изготвяне на отговор за успешно плащане 200

6. Изход от функция
7. Изготвяне на отговор за неуспешно плащане 400, поради невалиден идентификатор
8. Изготвяне на отговор за неуспешно плащане 202, поради вече платен запис

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 регистрационен 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

В тази диаграма е описана следната функционалност:

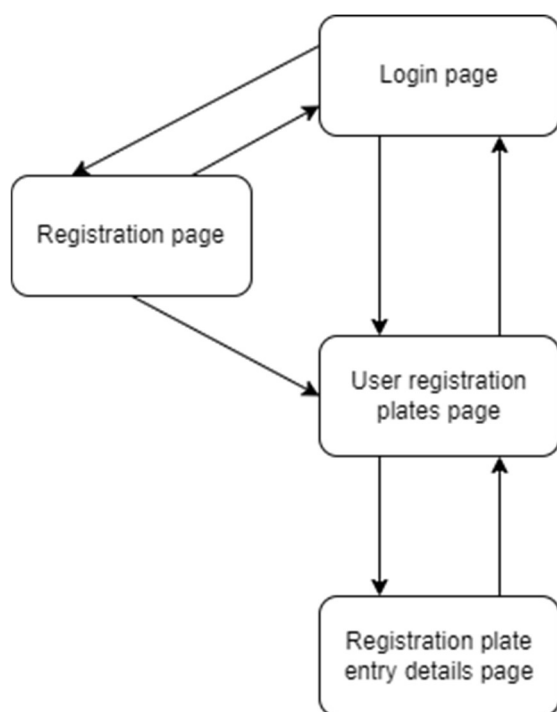
1. Вход във функция: `get_entry_data(registration: RegistrationModel)`
2. Създаване на празен списък с данни, които да бъдат върнати
3. Проверка на условие дали в базата данни има още данни за паркирания
  - a. Ако да, премини към 4
  - b. Ако не, премини към 5
4. Добави запис към изходния списък, премини на следващия елемент и се върни на 3
5. Изход от функция и връщане на списък с данни

## 4.2 Реализация на мобилно приложение

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

Приложението има четири страници. При зареждане, се отваря страницата за вход на потребителя 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.

На Фигура 27 е показан общият изглед на страниците на мобилното приложение и връзките между тях. Всяка 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**

Страницата за вход на съществуващи потребители предоставя начин на потребител да влезе в системата. Това е първата страница, която се визуализира, при зареждане на приложението. На тази страница се изисква да бъдат въведени e-mail и парола на регистриран потребител. При натискане на бутона за вход се изпраща заявка до сървър, обслужващ базата данни, който проверява дали потребителят може да влезе в системата. Ако потребителят е въвел коректни данни, заявката ще върне списък с регистрационни номера на потребителя. Ако въведените данни не са коректни, ще бъде върната грешка, която да бъде визуализирана.

#### **4.2.2 Страница за регистрация**

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. Ако данните не са коректни, като например невалиден e-mail адрес, сървърът връща грешка, със съответно съобщение. Това съобщение се визуализира, за да може потребителят да коригира данните си.

#### **4.2.3 Страница за общ преглед на регистрационни номера**

На тази страница има изобразен списък с всички регистрационни номера, които всъщност са бутони, които водят към следващата страница. Най-отдолу на страницата има бутон за добавяне на нов регистрационен номер, заедно с поле за въвеждане на низа на номера. Последната страница се достъпва, след като потребител избере регистрационен номер. Прави се заявка към сървъра, обслужващ базата данни, който валидира дали избраният регистрационен номер е действителен. Тази проверка не би трябвало да се провали, тъй като списъкът, който се използва за визуализиране на регистрационните номера, е взет от базата данни за съответния потребител.

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

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 уникалния идентификационен номер на входът в паркинга. Това число започва от 1 и се увеличава с всеки вход в паркинга, независимо кой потребител е влязъл.

### **4.3 Реализация на приложение за разпознаване на регистрационни номера**

Приложението за разпознаване на регистрационни номера е имплементирано на Python, използвайки библиотеките `ultralytics`, `easyocr` и `opencv`. Приложението чете входно видео, което подава на предварително трениран модел за разпознаване на регистрационни номера.

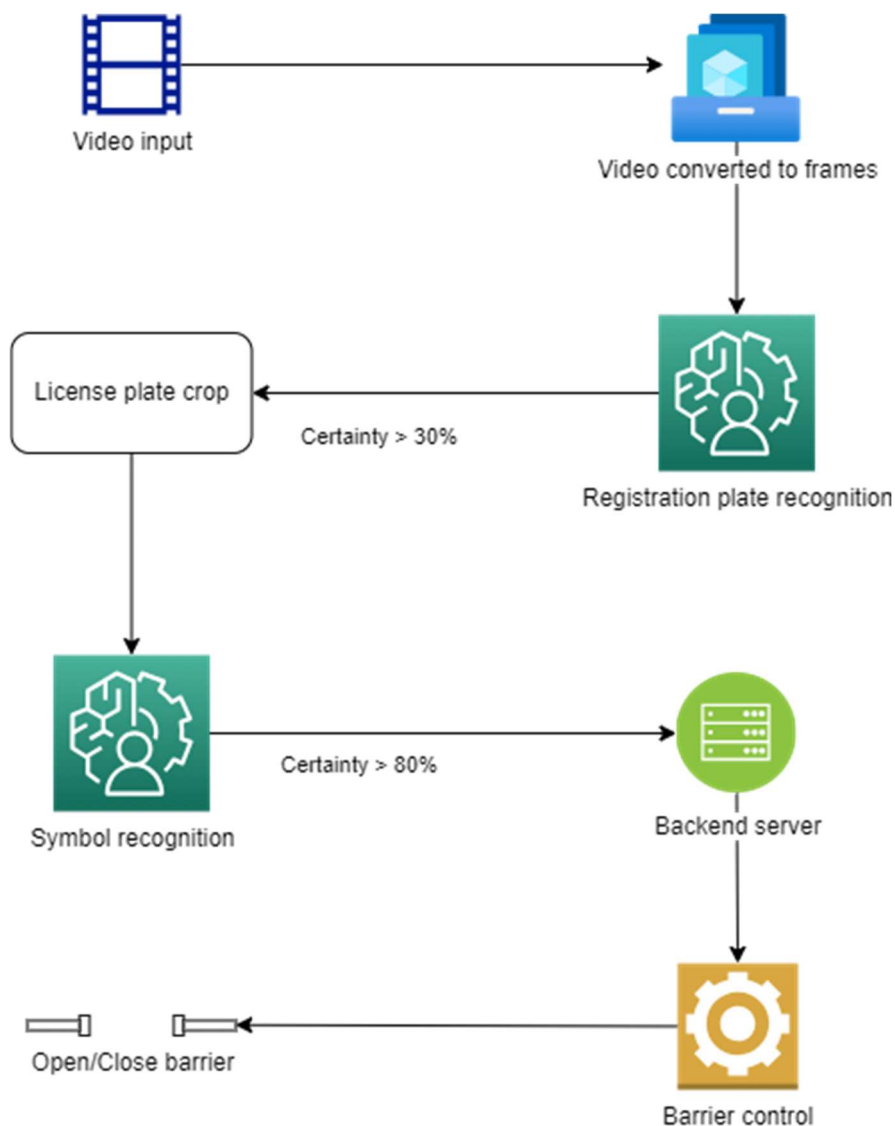
Видеото се разбива на кадри и всеки кадър се обработва отделно. Моделът взема даден кадър и се опитва да открие обект, който прилича на регистрационен номер. Ако е открит такъв, се проверява сигурността на модела, в дадения обект. Ако моделът смята, с

поне 30% точност, че обектът е регистрационен номер, то този обект се подава на друг модел, който се занимава с разпознаване на символи.

Вторият модел е предоставен от библиотеката *easyocr*. Този модел взема изображението, проверява го за познати символи и връща резултат под формата на символен низ и процент сигурност.

Когато получим резултат от този модел с поне 80% сигурност, се прави заявка към сървърното приложение, което обслужва базата данни за да бъде направен опит за влизане или излизане от паркинга.

На Фигура 28 е показан начинът на работа на приложението, описан по-горе.



ФИГУРА 28 НАЧИН НА РАБОТА НА ПРИЛОЖЕНИЕТО



#### 4.3.1 Подготовка на входни данни за модела

За да бъде трениран моделът, са използвани 156 изображения на автомобили с български регистрационни номера. Изображенията са взети от страницата на популярен онлайн каталог за употребявани автомобили. Всяко от изображенията съдържа ясно видим автомобил и неговия регистрационен номер. Избрани са изображения, където е видим или задният или предният номер, за да може моделът да разпознава регистрационни номера независимо от посоката на движение на автомобила.

След като са избрани изображенията, те трябва да бъдат анотирани. Тази задача е изцяло ръчен процес, който е изпълнен използвайки CVAT.AI платформата. Тази платформа позволява да бъдат създадени множество класове за класификация, но за нашите цели имаме само един клас „registration\_plate“. След като са качени изображенията и е създаден класификационният клас, следва да бъдат обходени всички изображения и да бъдат анотирани. Това става чрез избиране на класификационния клас и рисуване на правоъгълник около обекта, който искаме да анотираме.

След като е приключена анотацията, трябва да експортираме готовите данни във формат YOLO и да съхраним файловете в директорията на проекта.

#### 4.3.2 Трениране на модел

За тренирането на модела е използвана библиотеката ultralytics за Python. Чрез тази библиотека, тренирането на модел се случва само с няколко реда код.

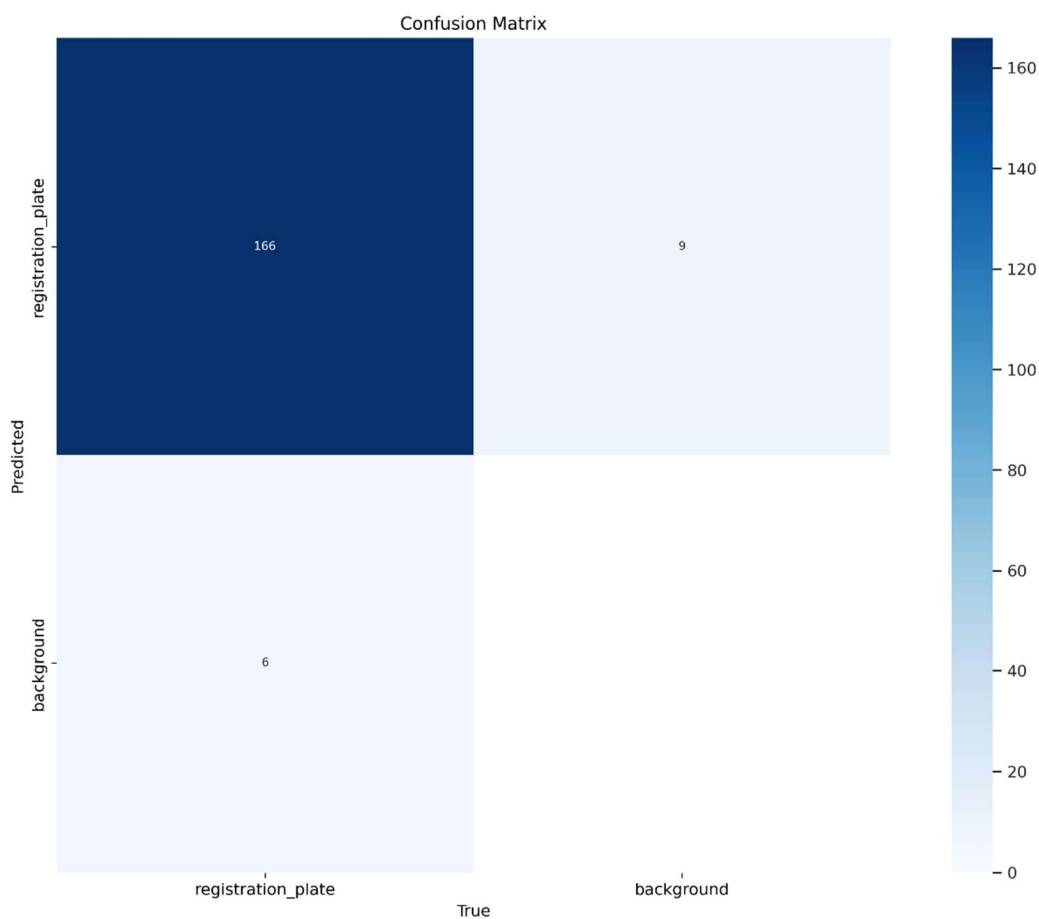
Първо трябва да изберем какъв вид модел ще тренираме. За нашите цели е избран модел YOLO версия 8 със среден размер. Този модел предлага добри резултати, като не е прекалено тежък за изпълнение.

След това, трябва да изберем колко епохи искаме да тренираме нашия модел. Епохата представлява едно обхождане на всичките ни данни. При изпълнението на трениране за 100 епохи върху 156 изображения, моделът вижда поне 15600 регистрационни номера. Реално моделът вижда малко повече номера, тъй като на някои изображения се виждат повече от един автомобил.

Процесът на трениране може да отнеме много време, ако е изпълнен използвайки централния процесор на компютъра. Заради това, библиотеката ultralytics позволява използване на графичния процесор на компютъра за трениране на модела. По този начин времето за трениране пада драстично. За сравнение, най-малкият модел, трениран за 100 епохи на централен процесор, върху 50 изображения, отне около 1 час за тренировка. Този модел не предлагаше задоволителни резултати и не успяваше да засече регистрационни номера. Следващият модел, беше трениран на две видео карти, използвайки среден размер модел и 156 тренировъчни изображения. Този модел успява да засече регистрационни номера с висока точност. Времето за трениране на новия, по-голям модел, е под 3 минути!

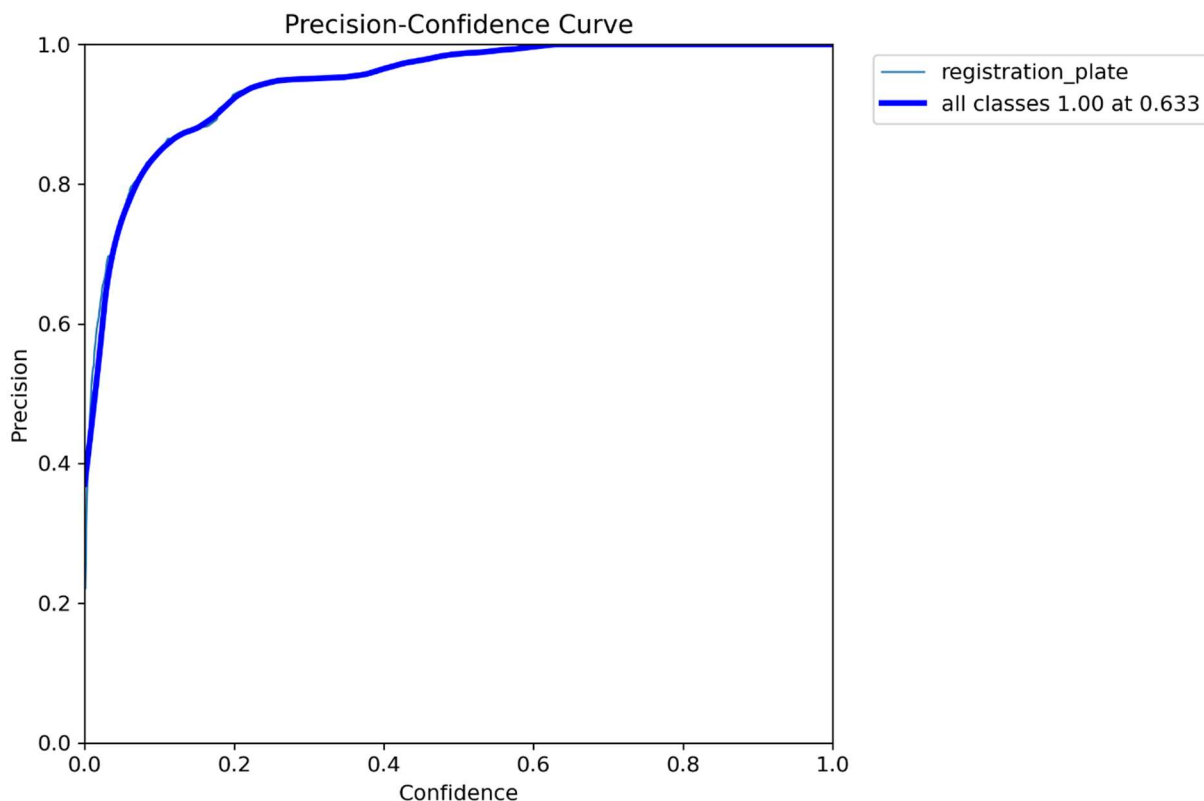
Финалният модел е трениран на 2 RTX4090 видео карти, върху 156 изображения, за 100 епохи. На Фигура 29 е показана матрицата на обръкванията на модела. Моделът е

установил коректно 166 регистрационни номера и е имал няколко обърквания с фонові елементи.



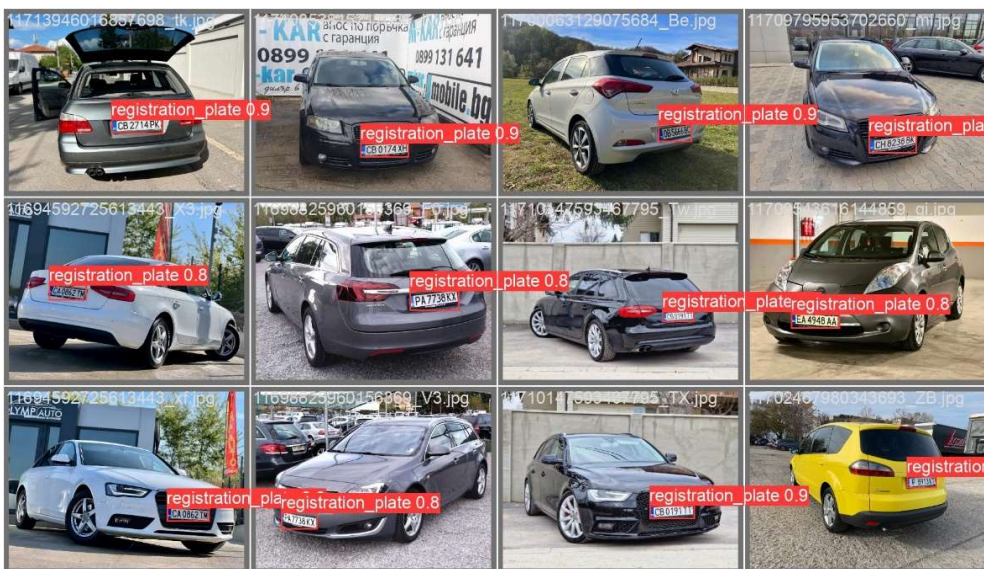
**ФИГУРА 29** МАТРИЦА НА ОБЪРКВАНЕТО НА МОДЕЛА

На Фигура 30 е показано отношението между прецизност и точност. Това отношение ни показва колко точен е моделът при различните нива на увереност. За този модел, моделът не е напълно точен, докато не стигне увереност от 63%. Всякакъв резултат над 63% е гарантирано валиден регистрационен номер. Задоволителните резултати започват от около 20%.



ФИГУРА 30 ОТНОШЕНИЕ НА ПРЕЦИЗНОСТ КЪМ УВЕРЕНОСТ

На Фигура 31 са показани част от валидационните резултати на модела. Валидацията представлява част от тренировъчния процес, при която моделът е тестван върху изображения, които не е виждал до дадения момент. При валидирането на този модел, имаме точно открити регистрационни номера с висока увереност от страна на модела.



ФИГУРА 31 ВАЛИДАЦИОННИ РЕЗУЛТАТИ НА МОДЕЛА

#### 4.3.4 Визуализация на резултат

Визуализацията на получените резултати се случва чрез библиотеката, използвана за обработване на видео потока: `opencv`. Всеки прочетен кадър се обработва, като на визуализирания кадър се очертава откритият регистрационен номер. След това, в горен ляв ъгъл на визуализацията, се показват данни от моделите. Първият елемент е изрязаният регистрационен номер, който е обработен допълнително за да се виждат символите лесно. Това е изображението, което се подава на модела за разпознаване на символи. След това са визуализирани резултатът от изхода на модела за разпознаване на символи и, ако има достатъчно висока точност, се изписва и отговорът на сървъра на базата данни, който е получил данните на регистрационния номер.

На Фигура 32 е показан кадър от визуализацията на резултат от обработване на видеоклип. Както се вижда на фигурата, регистрационният номер е ограден в правоъгълник и са изведени данни за съдържанието на номера, както и отговорът на сървъра.



ФИГУРА 32 ВИЗУАЛИЗАЦИЯ НА РЕЗУЛТАТ ОТ ПРИЛОЖЕНИЕТО

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

### **5.1 Регистрация и създаване на първи регистрационен номер**

Този пример изисква да бъде използвано приложението за потребителски достъп до системата. При началния екран се избира опцията за регистрация на нов потребител. Отваря се екран, който изисква потребителят да въведе валиден e-mail адрес, потребителско име и парола. Паролата трябва да бъде повторена, за да бъде сигурно, че потребителят не я е написал погрешно.

След натискането на бутона за създаване на регистрация, се прави връзка със сървъра за управление на базата данни. Той обработва подадените данни и връща отговор. Този отговор може да обявява успешното извършване на операцията, което ще доведе до зареждане на следващата страница или да върне грешка като отговор. Тази грешка може да бъде заради вече регистриран e-mail или заради невалидно подаден e-mail. При наличие на грешка, тя ще бъде визуализирана и потребителят няма да може да продължи към следващата страница, докато не я коригира.

След като се регистрира успешно, потребителят ще премине към главната страница на приложението, където той може да види вече регистрираните си номера. Тъй като, в този пример, потребителят е новосъздаден, тази страница ще има само текстово поле за въвеждане на регистрационен номер и бутон за потвърждение. При натискане на бутона, ще бъде изпратена заявка до сървъра, обслужващ базата данни, който ще провери дали регистрационният номер е коректно написан или дали не е бил вече регистриран от друг потребител. Ако всички проверки са успешни, регистрационният номер ще бъде асоцииран с потребителя и ще бъде върнат отговор за успешна проверка. Потребителят ще види своя новодобавен регистрационен номер над полето за добавяне на нов. При неуспешно добавяне, ще бъде върната грешка, която ще бъде визуализирана, за да може потребителят да я коригира.

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

Този пример изисква да бъде използвано приложението за потребителски достъп до системата. При началния екран се въвеждат данните на съществуващ потребител. След натискане на бутона за вход, се прави заявка към сървъра, обслужващ базата данни, който трябва да провери дали данните са коректни и потребителят съществува. Ако това е така, сървърът отговаря със списък с всички регистрационни номера, асоциирани с дадения потребител. Ако данните не са верни, сървърът връща грешка, която се визуализира, за да може потребителят да коригира данните си.

След коректно въведени данни, се зарежда главната страница на приложението. На тази страница се визуализират всички регистрирани номера на дадения потребител, които са върнати от сървъра, при натискане на бутона за вход. От тук, потребителят може да натисне върху даден регистрационен номер и така да изпрати нова заявка до сървъра, с която да изиска всички данни за престой на паркинга за дадения регистрационен номер. Заявката би трябвало винаги да е успешна, тъй като всички визуализирани данни са предоставени от

базата данни. Въпреки това, съществува проверка дали регистрационният номер е валиден, за да се предпазим от потенциално достъпване на входната точка на сървъра с грешни данни.

При успешна заявка и позитивен отговор от сървъра, се визуализира следващата страница, която дава на потребителя детайлна информация за всеки негов престой в паркинга, както и опцията да заплати предходния си престой. На този екран се вижда уникалният номер на престоя, часът и датата на влизане, часът и датата на излизане и статусът на плащането.

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

За този пример, трябва да имаме автомобил, който е асоцииран с потребителски профил. В тази ситуация, автомобилът ще бъде допуснат до паркинга и ще бъде върнат отговор, че входът е успешен.

За да бъде тестван този сценарий е необходимо приложението за разпознаване на регистрационни номера да получи видео, в което да бъде видим автомобил, който е добавен към профил от системата. Този автомобил трябва да бъде без парични задължения за предния си престой. В тази ситуация, при успешно прочитане на регистрационния номер, ще бъде направена заявка към сървъра, обслужващ базата данни, която позволява вход. Барьерата на паркинга ще бъде вдигната и автомобилът ще може да премине.

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

За този пример, трябва да имаме автомобил, който е асоцииран с потребителски профил. В тази ситуация, автомобилът няма да бъде допуснат до паркинга и ще бъде върнат отговор, че автомобилът не е заплатил преден престой.

За да бъде тестван този сценарий е необходимо приложението за разпознаване на регистрационни номера да получи видео, в което да бъде видим автомобил, който е добавен към профил от системата, но има незавършени плащания в историята на престойте си. В тази ситуация, при успешно прочитане на регистрационния номер, ще бъде направена заявка към сървъра на базата данни, която ще покаже, че има неплатен престой на паркинг. Барьерата няма да се вдигне и автомобилът ще остане отвън.

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

За този пример, трябва да имаме автомобил, който не е асоцииран с потребителски профил. В тази ситуация, автомобилът няма да бъде допуснат до паркинга и ще бъде върнат отговор, че автомобилът не е регистриран в системата.

За да бъде тестван този сценарий е необходимо приложението за разпознаване на регистрационни номера да получи видео, в което да бъде видим автомобил, който не е добавен към профил от системата. При прочитане на регистрационния номер, ще бъде пусната заявка към сървъра на базата данни, който ще отговори, че автомобилът не е регистриран. Барьерата няма да бъде вдигната и автомобилът няма да премине.

## VI. ЗАКЛЮЧЕНИЕ

В заключение, тази разработка предлага иновативно решение за интелигентна паркинг система, която е насочена към улесняване на процесите на паркиране в съвременната градска среда. Тя адресира успешно проблемите със задръстванията на входно-изходните точки на паркингите и забавянията при плащане, като внедрява модерни технологии за разпознаване на регистрационни номера, онлайн плащания и управлението на паркинг пространствата.

Основният принос на разработката включва внедряването на сложни алгоритми за разпознаване на обекти, използвайки модели като YOLOv8 за идентифициране на регистрационни номера и библиотеката EasyOCR за разпознаване на символи. Интеграцията на тези технологии с потребителски интерфейс, реализиран чрез .NET MAUI, и сървър, който обслужва база данни, реализирана чрез PostgreSQL, базиран на FastAPI, осигурява ефективна комуникация между компонентите на системата и улеснява управлението на данните.

Предложената система предлага значителни предимства като елиминирането на необходимостта от човешка намеса при вход-изход процедурите и плащанията, което ускорява процеса на паркиране и намалява потенциалните грешки и злоупотреби. Същевременно, използването на облачни услуги като Google Cloud и Aiven осигурява висока надеждност и скалируемост на решението.

Въпреки че разработката показва обещаващи резултати, съществуват определени ограничения, свързани с качеството на заснемане и капацитетът на системата да обработва по-сложни случаи. Необходимостта от висока точност на камерите и поддръжката на добре осветени зони около тях остават ключов фактор за успеха на системата.

Интеграцията на съвременни технологии и автоматизация в управлението на паркинг системи, както е предвидено в тази разработка, представлява значителен напредък в подобряването на градската инфраструктура. В бъдеще, по-нататъшното усъвършенстване на алгоритмите за разпознаване, оптимизацията на потребителския интерфейс и инфраструктурата, могат да допринесат за още по-голяма ефективност и приложимост на тази система в реални условия.

## СЪКРАЩЕНИЯ

|                                                   |    |
|---------------------------------------------------|----|
| AI - Artificial Intelligence .....                | 11 |
| ASGI - Asynchronous Server Gateway Interface..... | 10 |
| CVAT - Computer Vision Annotation Tool.....       | 11 |
| DOS - Denial Of Service .....                     | 17 |
| HTTP - Hyper Text Transfer Protocol .....         | 15 |
| JSON - JavaScript Object Notation.....            | 25 |
| MAUI - Multi-platform App UI .....                | 9  |
| OCR - Optical Character Recognition.....          | 11 |
| SQL - Structured Query Language.....              | 9  |
| YOLO - You Only Look Once .....                   | 11 |
| МПС - Моторни Превозни Средства .....             | 7  |



## ИЗТОЧНИЦИ

- [1] Turing, „Why Is Python Best Adapted to AI and Machine Learning?“, [Онлайн]. Available: <https://www.turing.com/kb/python-best-adapted-to-ai-and-machine-learning>.
- [2] „What is MAUI,“ Microsoft, [Онлайн]. Available: <https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0>.
- [3] „About PostgreSQL,“ PostgreSQL. [Онлайн].
- [4] S. Team, „SQLAlchemy 2.0 Documentation,“ [Онлайн]. Available: <https://docs.sqlalchemy.org/en/20/>.
- [5] Tiangolo, „FastAPI,“ [Онлайн]. Available: <https://fastapi.tiangolo.com/>.
- [6] Encode, „Uvicorn,“ [Онлайн]. Available: <https://www.uvicorn.org/>.
- [7] MatLab, „What Is Object Detection?,“ [Онлайн]. Available: <https://www.mathworks.com/discovery/object-detection.html>.
- [8] „YOLOv8,“ Ultralytics, [Онлайн]. Available: <https://github.com/ultralytics/ultralytics>.
- [9] CVAT, „Documentation of Computer Vision Annotation Tool,“ [Онлайн]. Available: <https://docs.cvat.ai/docs/>.
- [10] Amazon, „What is OCR (Optical Character Recognition)?,“ [Онлайн]. Available: <https://aws.amazon.com/what-is/ocr/>.
- [11] Aiven, „Aiven docs,“ [Онлайн]. Available: <https://aiven.io/docs>.
- [12] Docker, „Get started guide,“ [Онлайн]. Available: <https://docs.docker.com/get-started/>.
- [13] Kubernetes, „Kubernetes Documentation,“ [Онлайн]. Available: <https://kubernetes.io/docs/home/>.
- [14] Google, „Google Cloud Documentation,“ [Онлайн]. Available: <https://cloud.google.com/docs>.
- [15] Mobile.bg, „Mobile,“ [Онлайн]. Available: <https://www.mobile.bg/>.