Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.93

*На правах рукопису*

# ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: “ Методи нейромережевого розпізнавання об’єктів в системі громадського контролю законності паркування на базі хмарних технологій”

Спеціальність – 121 “Інженерія програмного забезпечення”

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

БР.ІПЗ – 30.00.00.000

Студент
ІПЗм-21 _____ /Ілля ПЕКНЕВИЧ/

Науковий керівник
к.т.н., доц. _____ /Тетяна КОВАЛЮК/

Консультант
з питань нормоконтролю
к.т.н., асистент _____ /Анастасія ВЕЧЕРКОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н., проф. _____ /Олексій БИЧКОВ/

Київ – 2022

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

д.т.н., проф. Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Спеціальність 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри програмних систем і технологій

_____ (О.С.Бичков)

“___” _____ 20__ р.

**ЗАВДАННЯ**

**НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ**

**СТУДЕНТУ**

Пекневичу Іллі Ігоровичу

1. Тема випускної кваліфікаційної магістерської роботи «Методи нейромережевого розпізнавання об’єктів у системі громадського контролю законності паркування на базі хмарних технологій» та керівник проекту (роботи) Ковалюк Тетяна Володимирівна, к.т.н., доцент затверджені наказом вищого навчального закладу від «__»____20__ р. №_____

2. Строк здачі студентом закінченої роботи «06» травня 2022 р.

3. Вихідні дані до роботи: підручники, навчальні посібники, статті, Інтернет-ресурси.

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

Аналітична частина:

- обґрунтувати актуальність розробки системи громадського контролю за законністю паркування;
- дослідити та формалізувати бізнес-процеси комунікації вповноваженими службами;

- дослідити процес виявлення та розпізнавання номерних знаків на зображеннях;
- сформувати функціональні вимоги та обрати відповідні технології розробки.

Практична частина:

- спроектувати архітектуру системи відповідно до визначених бізнес-процесів;
- спроектувати базу даних системи;
- розробити програмну систему та відповідну хмарну інфраструктуру.

5. Перелік графічного матеріалу (з точним забезпеченням обов'язкових креслень): блок-схеми, UML-діаграми.

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Аналіз предметної області та теоретичних аспектів | Ковалюк Т.В. | 06.10.21 | 14.12.21 |
| Проектування та вибір технологій програмування | Ковалюк Т.В. | 14.12.21 | 17.01.22 |
| Програмна реалізація | Ковалюк Т.В. | 17.01.22 | 04.05.22 |
| Огляд результату виконання поставленої задачі | Ковалюк Т.В. | 04.05.22 | 06.05.22 |

7. Дата видачі завдання «06» жовтня 2021 р.

Керівник _____ Тетяна КОВАЛЮК

Завдання прийняв до виконання _____ Ілля ПЕКНЕВИЧ

# КАЛЕНДАРНИЙ ПЛАН

| Номер і назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|
| 1: Дослідження та опис бізнес-процесів предметної області | 06.10.21 – 22.10.21 | |
| 2: Дослідження та опис процесу розпізнавання номерних знаків | 25.20.21 – 30.11.21 | |
| 3: Опис функціональних та нефункціональних вимог до системи | 30.11.21 – 14.12.21 | |
| 4: Розробка необхідних архітектурних діаграм | 14.12.21 – 24.12.21 | |
| 5: Аналіз та вибір технологій програмування | 24.12.21 – 17.01.22 | |
| 6: Створення цільової бази даних системи | 18.01.22 – 04.02.22 | |
| 7: Розробка програмних компонент | 07.02.22 – 01.04.22 | |
| 8: Розгортання хмарної інфраструктури | 01.04.22 – 11.04.22 | |
| 9: Оформлення пояснювальної записки | 12.04.22 – 06.05.22 | |
| 10: Підготовка презентаційних матеріалів | 09.05.22 – 16.05.22 | |

Студент – магістр _____ Ілля ПЕКНЕВИЧ

Керівник роботи _____ Тетяна КОВАЛЮК

# АНОТАЦІЯ

**Випускна кваліфікаційна магістерська робота** складається зі вступу, 4 розділів, висновків та списку використаних джерел (11 найменувань). Містить у собі 14 рисунків, 7 таблиць та 5 додатків. Загальний обсяг роботи становить 99 сторінки.

**Темою** роботи є “Методи нейромережевого розпізнавання об’єктів в системі громадського контролю законності паркування на базі хмарних технологій”.

**Об’єктом** роботи є нейромережеві методи та хмарні технології у програмних системах автоматизації. **Предметом** – проектування та розробка комплексної програмної системи «Parking Control» для автоматизації процесів звітування та обробки актів незаконного паркування з використанням технологій нейромережевого розпізнавання та хмарних технологій Microsoft Azure.

**Метою** роботи є налагодження комунікації між громадянами та вповноваженими службами евакуації, а також зменшення кількості актів незаконного паркування шляхом проектування та розробки комплексної програмної системи «Parking Control», що надає можливості для звітування, обробки та реагування щодо актів подібних правопорушень.

**Результати роботи**: Формалізовані процеси контролю за законністю паркування. Досліджені та описані підходи до нейромережевого розпізнавання номерних знаків. Спроектовані та розроблені загальна архітектура програмної системи, сховища даних та програмний продукт «Parking Control».

**Ключові слова**: комплексна програмна система, автоматизація, програмна архітектура, Azure, Blazor, нейромережеве розпізнавання.

# ABSTRACT

**The final qualifying master's thesis** consists of an introduction, 4 chapters, conclusions and a list of sources used (11 items). Contains 14 figures, 7 tables and 5 appendices. The total volume of the work is 99 pages.

The **theme** of the work is "Methods of neural network recognition of objects in the system of public control of the legality of parking based on cloud technologies".

The **object** of the work is neural network methods and cloud technologies in software automation systems. **Subject** - design and development of a comprehensive software system "Parking Control" to automate the reporting and processing of illegal parking with the use of neural network recognition technologies and cloud technologies Microsoft Azure.

The **goal** of the work is to establish communication between citizens and authorized evacuation services, as well as reduce the number of illegal parking acts by designing and developing a comprehensive software system "Parking Control", which provides opportunities for reporting, processing, and responding to such offenses.

**Results of work**: Formalized processes of control over the legality of parking. Methods of neural network license plate recognition have been studied and described. The general architecture of the software system, data warehouses, and the software product "Parking Control" are designed and developed.

**Keywords**: complex software system, automation, software architecture, Azure, Blazor, neural network recognition.

# ЗМІСТ

ВСТУП ..... 10
РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА ..... 13
1.1 Аналіз предметної області та існуючих механізмів звітування ..... 13
1.2 Нейромережеве розпізнавання номерних знаків ..... 14
1.2.1 Локалізація номерного знаку на зображенні ..... 15
1.2.2 Процес нейромережевого розпізнавання текстових зображень 19
1.2.3 Переваги згорткових мереж для задач розпізнавання ..... 23
1.2.4 Запропонована нейронна мережа та її використання ..... 26
РОЗДІЛ 2 ПРИЗНАЧЕННЯ ТА ВИМОГИ ДО СИСТЕМИ ..... 28
2.1 Призначення та мета програмної системи ..... 28
2.2 Функціональні вимоги до системи ..... 28
2.3 Нефункціональні вимоги ..... 31
РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ «PARKING CONTROL» ..... 33
3.1 Розробка бізнес-процесів та архітектури програмної системи ..... 33
3.2 Проектування бази даних системи ..... 35
3.2.1 Аналіз предметної області ..... 36
3.2.2 Інфологічне проектування ..... 37
3.2.3 Даталогічне та фізичне проектування ..... 40
3.3 Розробка веб-частини ..... 41
3.3.1 Вимоги до технології згідно з поставленою задачею ..... 42
3.3.2 Вибір платформи Blazor ..... 42
3.3.3 Аспекти використання фреймворку ..... 44
3.4 Azure Cognitive сервіси для нейромережевого розпізнавання ..... 46
3.5 Azure Functions як інструмент безсерверних обчислень ..... 47

3.5.1 Azure Durable Functions для організації бізнес-процесу..... 48
3.5.2 Аспекти реалізації процесу розпізнавання..... 49
3.6 Розробка мобільного додатку ..... 50
3.6.1 Архітектура роботи Xamarin.Forms..... 50
3.6.2 Підходи та програмні рішення розробки ..... 51
РОЗДІЛ 4 ПРОГРАМНА СИСТЕМА «PARKING CONTROL»..... 53
4.1 Загальний функціонал підсистем веб-додатку ..... 53
4.2 Підсистема адміністратора ..... 54
4.3 Підсистема обробника звітів..... 55
4.3 Підсистема громадянина..... 56
ВИСНОВКИ ..... 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 58
Додаток А – Діаграми прецедентів користувацьких підсистем ..... 60
Додаток Б – Діаграма бази даних системи ..... 61
Додаток В – Фрагменти реалізації програмної системи ..... 62
В.1 Приклад задання інтерфейсу та логіки компонента ..... 62
В.2 Реалізація класу розпізнавання з Azure Cognitive..... 63
В.3 Реалізація класу розпізнавання з Azure Cognitive..... 64
В.4 Реалізація MVVM шаблону з Xamarin ..... 65
Додаток Д – Інтерфейс розробленої системи ..... 67
Д.1 Загальні елементи інтерфейсу веб-додатку..... 67
Д.2 Підсистема адміністратора..... 68
Д.3 Підсистема обробника звітів..... 70
Д.4 Підсистема громадянина..... 73
Додаток Е – Software Architecture Document ..... 75

# ВСТУП

**Оцінка сучасного стану об’єкта розробки.** Швидкий розвиток технологій надає широкий спектр можливостей для підвищення зручності та ефективності великого спектру процесів людської діяльності шляхом їх автоматизації. Одними з основних технологічних рішень, що є невід’ємними рушіями прогресу систем автоматизації, можна назвати хмарні технології та засоби нейромережевого розпізнавання зображень. Перші надають широкі можливості організації комп’ютерних ресурсів, сховищ даних та систем комунікації у вигляді готового веб-сервісу, який легко конфігурується, масштабується та не має необхідності у постійній інфраструктурній підтримці, що покладається на постачальника послуг. Засоби розпізнавання у свою чергу надають можливість автоматизувати ті процеси, що до цього вимагалася безпосередня участь оператора-людини.

Сфера контролю за законністю паркування досі не отримала бажаного рівня автоматизації. Повідомлення про порушення досі надходять до вповноважених служб телефоном чи шляхом письмових заяв, що негативно впливає на ефективність комунікації між громадянами та ефективними службами, адже в такий спосіб неможливо оцінити валідність поточної скарги без виїзду оперативної групи.

Таким чином систем автоматизації контролю за законністю паркування можна охарактеризувати як такі, що мають усе необхідне технологічне підґрунтя у вигляді вищенаведених технологій, проте ще не є розкритими у вигляді конкретних програмних рішень, що вже мають практичне застосування. У подібних системах хмарні технології надають масштабовану та підтримувану інфраструктуру, а засоби нейромережевого розпізнавання – можливості до автоматичного розпізнавання номерних знаків порушника чи безпосередньо самого факту правопорушення як такого.

**Актуальність роботи та підстави до її виконання.** Сучасні шалені темпи урбанізації ведуть за собою одну із основних проблем великих міст –

перенасичення автомобілями. Окрім складного трафіку на часі також питання нестачі пакувальних територій. Кожна новобудова, сотні яких кожного року виникають на першій вільній галявині, покриває потребу у паркомісцях у кращому випадку на 40%. Логічним наслідком подібного є те, що водії часто порушують усі загальноприйняті правила паркування, створюючи незручності для інших, а іноді навіть утворюючи аварійні ситуації. Влада міст заключає контракти зі службами евакуації, проте досі не існує централізованого засобу, за яким громадянин може повідомити про акт незаконного паркування, а оператор евакуаційної служби у свою чергу проаналізувати правопорушення та лише за зафіксованої необхідності викликати екіпаж. Подібна система надала б змогу більш ефективно вирішувати проблему незаконного паркування у великих містах, що робить обрану тему актуальною.

**Мета й завдання роботи.** Метою роботи є налагодження комунікації між громадянами та вповноваженими службами евакуації шляхом проектування та розробки комплексної програмної системи «Parking Control», що надає можливості для звітування, обробки та реагування щодо актів подібних правопорушень. Для досягнення цієї мети були поставлені наступні завдання:

- дослідити існуючи засоби звітування та механізми реагування щодо актів незаконного паркування;
- дослідити сучасні засоби та технології розробки комплексних програмних систем, а також архітектури їх взаємодії.
- ознайомитися із хмарними технологіями Azure та когнітивними сервісами нейромережевого розпізнавання;
- розробити технічне завдання до продукту відповідно до функціональних потреб громадян та евакуаційних служб;
- спроектувати та розробити комплексну програмну систему «Parking control».

**Об’єкт, методи та засоби розроблення.** Об’єктом розроблення є засіб автоматизації контролю паркування у вигляді сучасної комплексної програмної

системи на базі хмарної платформи Azure та засобів нейромережевого розпізнавання.

Під час розробки була використана еволюційна модель. Версії продукту розроблялися поетапно з ітераційною валідацією функціонального аспекту кінцевими користувачами: звичайними громадянами (клієнтський мобільний додаток) та представниками евакуаційної служби (веб-додаток адміністрування).

В якості інструментів створення були використані: інтегроване середовище розробки Visual Studio 2022, Microsoft SQL Server, Azure Portal та супутні програмні засоби.

**Можливі сфери застосування.** Розроблений програмний продукт може використовуватися великими містами, що мають контракти з евакуаційними службами для автоматизації процесу звітування щодо актів порушення правил паркування. Разом із цим, розроблена програмна база може бути адаптована до широкого спектру предметних областей, процеси яких потребують експертного контролю.

# РОЗДІЛ 1
# ТЕОРЕТИЧНА ЧАСТИНА

## 1.1 Аналіз предметної області та існуючих механізмів звітування щодо порушення правил паркування

За основу для вивчення існуючих механізмів евакуації у випадках незаконного паркування були обрані законодавчі та адміністративні положення міста Києва.

Згідно з постановою КМДА, що затвердило створення відповідної служби інспекції, громадянин, що став свідком порушення правил паркування, повинен звертатися до патрульної поліції. При цьому, як зазначають працівники інспекції, зробити це можна тільки у письмовій формі, адже часто виїзд оперативної групи не є раціональним у таких випадках. Для подачі заяви необхідно зробити декілька фотографій, на яких видно місцевість, дорожні знаки чи розмітку, вказати місто, адресу, дату, номерні знаки автомобіля та контактні дані громадянина, що звітує. Після чого звіт необхідно роздрукувати та відправити листом у відділення патрульної поліції. Поліція у свою чергу повинна протягом 30 днів розглянути скаргу, встановити особистість правопорушника та відповідним чином покарати його, якщо порушення дійсно мало місце.

Зрозуміло, що подібний довгий та бюрократизований процес не є зручним для громадянина, що став свідком правопорушення та не є оперативним в цілому.

Виходячи із аналізу існуючих механізмів звітування та алгоритмів роботи державних та евакуаційних служб відповідно до випадків незаконного паркування, можна виділити наступні процеси: звітування щодо правопорушення, обробка звіту експертом евакуаційної служби, евакуація автомобіля екіпажем, адміністрування людських та автомобільних ресурсів евакуаційної служби.

Процес звітування громадянином щодо правопорушення згідно вищеописаного дослідження, повинен дозволяти людині надавати наступну інформацію:

- панорамні фото, на яких якісно відображена уся необхідна інформація для визначення акту правопорушення;
- фото номерних знаків автомобіля, що на думку громадянина, порушує правила паркування;
- інформацію про локацію правопорушення для відповідного реагування служб у випадку необхідності.

Процес обробки отриманих звітів полягає у перегляді експертом наданих фото правопорушень, а також супутню інформацію. Відповідно до наданих панорамних фото оператором приймається рішення чи дійсно наявний акт правопорушення згідно актуальних нормативним документів. Результуючим кроком є відповідне реагування: підтвердження або відхилення того чи іншого звіту відповідно до висновків на етапі аналізу наданих фото. У випадку підтвердження правопорушення відбувається призначення випадку тому чи іншому евакуаційному екіпажу з огляду на його доступність, як фізичну, так і територіальну. Якщо ж згідно висновку експерта у рамках поточної заяви відсутнє правопорушення як таке, надається обґрунтування з посиланнями на відповідні правила паркування, що є зафіксованими в рамках держави.

На ряду із процесом обробки має місце адміністрування ресурсів та статистичного аналізу діяльності служби. Це перш за все перегляд статистичних показників щодо звітів (скільки було звітовано випадків, з яких районів, які тенденції кількості, яка частина звітів була підтверджена тощо) та управління обробниками, евакуаційними екіпажами та автомобілями.

## 1.2 Нейромережеве розпізнавання номерних знаків

У контексті бізнес-процесу, що розглядається, має місце автоматизація процесу розпізнавання номерних знаків, фото яких надає громадянин разом із

звітом про правопорушення. Це надасть можливість підтягнути інформацію про автомобіль та власника без необхідності ручного вводу номеру, що підвищить користувацький комфорт громадянина та відкине можливість помилки при вводі. У рамках цього розділу запропоновані підходи до локалізації номерного знаку з подальшим його розпізнаванням.

### 1.2.1 Локалізація номерного знаку на зображенні

Першим кроком є пошук регіону номерного знаку на оригінальному зображенні. Запропонований алгоритм базується на методах обробки зображень на основі границь.

Ці методи обробляють зображення, отримуючи границі шляхом визначення точок, які мають різкі зміни яскравості. Перепад яскравості в області номерного знаку є більш помітним, ніж у інших місцях зображення, адже має більш виражені границі та текстурні особливості. Разом із цим такі методи надмірно чутливі до небажаних границь, тобто вони не є підходящими для складних зображень. Для вирішення цієї проблеми використовується покращення зображення з подальшим використанням морфологічних методів.

#### 1.2.1.1 Покращення оригінального зображення

Зображення зі складним фоном зазвичай містить шум, що призводить до небажаної щільності границь, що визначаються на зображенні. Для усунення шумів виконаємо згортання зображення з використанням медіанного фільтру [1].

Головна ідея медіанного фільтру полягає у проходженні сигналу елемент за елементом, замінюючи кожний із них медіаною сусідніх. У поточному випадку двовимірного зображення, вікно проходження включає у себе всі елементи у колі заданого радіусу.

На рис. 1.1 зображено приклад знайдених границь до та після використання медіанного фільтру. Можна бачити, що значна кількість небажаних границь від, наприклад, дорожньої бруківки зникла, що значно покращило результат.

1.2.1.2 Визначення границь об’єктів на зображенні

Методи виявлення границь засновані на принципі знаходження пікселів з різкими змінами яскравості, що відповідають «кордонам» об’єктів на зображенні. Більшість методів можна поєднати у дві категорії: градієнтні та лапласівські методи [1]. Використаємо градієнтний метод для визначення границь.

Спочатку виконуємо згортання зображення за допомогою фільтра Собеля, щоб знайти приблизну абсолютну величину градієнта у кожній точці зображення.

Оператор використовує ядра $3 \times 3$, з якими згортають оригінальне зображення для визначення наближених значень похідних по горизонталі і вертикалі. Нехай A – зображення, а $G_x$ та $G_y$ – два зображення, на яких кожна точка містить наближені похідні по $x$ та $y$. Вони знаходяться наступним чином:

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \; and \; G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad (1.1)$$

де * - двовимірна операція згортки. Таким чином у кожній точці зображення наближене значення градієнту можна знайти через отриманні наближені значення похідних: $G = \sqrt{G_x^2 + G_y^2}$.

Потім ми виявляємо крайові точки за допомогою порогового значення.

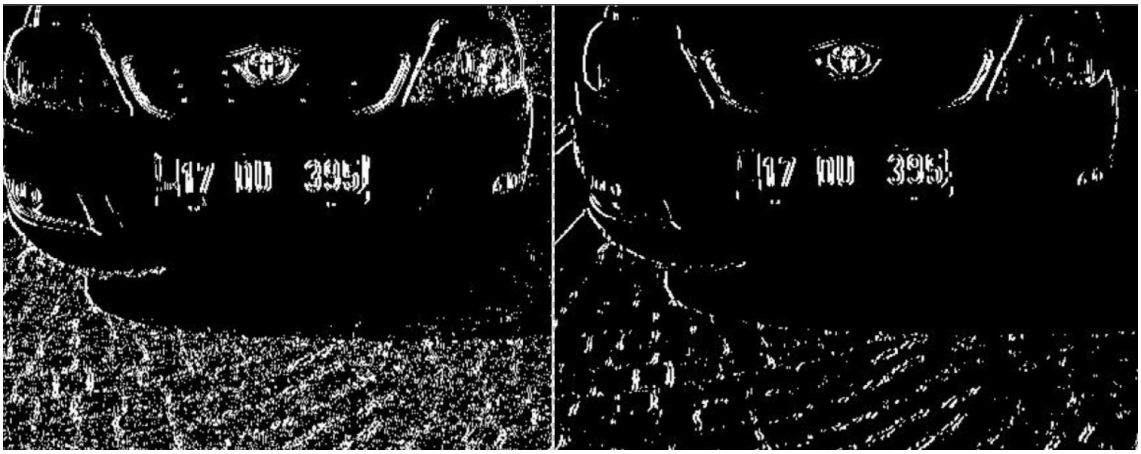


Рисунок 1.1. Визначення границь на зображенні до (зліва) та після (справа) застосування медіанного фільтру

Незважаючи на процедури покращення зображення перед процесом виявлення границь, у деяких випадках вертикальні границі окрім тих, що визначають номерний знак, можуть бути більш інтенсивними, що може призвести до некоректного визначення необхідного регіону. Задля видалення небажаних вертикальних границь визначимо у яких межах ми допускаємо їх довжину у рамках номерного знаку та видалимо усі ті, що не входять у сформований діапазон [2]. Граничні значення мають визначатися виходячи з розмірності вхідного зображення.

1.2.1.3 Виокремлення прямокутнику номерного знаку

Визначимо бажаний регіон як такий, що має найбільшу щільність вертикальних границь. Таким чином необхідно певним чином обрахувати цю щільність. Для цього виконаємо згортку зображення за наступними правилами:

1. Вхідне зображення розбивається на квадратні піксельні блоки невеликого розміру (2-3% розмірності основного зображення).
2. Для кожного такого блоку рахується кількість білих пікселів (що визначають попередньо знайдені границі).
3. Кожний піксель результуючого зображення (згортки) відповідає одному такому блоку. Його яскравість задається визначеною у відповідному блоці кількістю білих пікселів.

Результат роботи такої згортки зображений на рис. 1.2.

Рисунок 1.2. Розрахунок щільності вертикальних границь

Далі усі пікселі, значення яскравості яких менше за визначене порогове значення, видаляються із зображення. Після додаткового етапу згортки, розмірність якої визначена експериментальним шляхом на вхідних зображеннях, треба залишити лише цільовий регіон. Для цього усі пікселі, яскравість яких менша за максимальну яскравість поточної матриці, видаляються. У результаті чого отримуємо один, найбільш яскравий регіон зображення.

Для визначення знайденого регіону на оригінальному зображенні, значення пікселів кожного з квадратів згортки, що потрапив до результуючого регіону, залишаються, а усі інші – видаляються із зображення. Таким чином отримуємо знайдений прямокутний регіон на початковому зображенні.


Рисунок 1.3. Результат визначення регіону номерного знаку

1.2.1.4 Подальша обробка

Для підвищення ефективності кроку сегментації символів визначеного зображення номерного знаку, важливими до виконання є наступні 2 кроки:

1. Виправлення перекосу зображення. Зображення номерного знаку має бути горизонтальним. Таким чином на цьому кроці необхідно визначити кут перекосу зображення та виконати необхідний поворот.
2. Обрізання непотрібних областей виділеного регіону. Непотрібні області, тобто області, відмінні від безпосередньо символів пластини, можна спостерігати на отриманому зображенні прямокутного регіону (див. рис. 1.3). Для переходу до процесу сегментації, ці області необхідно видалити та зосередитися на якомога вузькій рамці, яка містить лише символи знаку.

### 1.2.2 Процес нейромережевого розпізнавання текстових зображень

Серед основних етапів розпізнавання тексту можна виділити наступні:

1. Вхідне зображення очищується від шуму та приводиться до вигляду, у якому можна ефективно виділяти та розпізнавати символи;
2. Зображення розбивається на блоки тексту, базуючись на особливостях його вирівнювання та розподілу;
3. Далі відбувається поділ на зображення рядків, а далі – окремих символів для окремої обробки кожного. Після чого системи розпізнавання працюють по своїм специфічним алгоритмам;
4. Виділені символи можуть порівнюватися з готовими шаблонами, розпізнаватися нейронними мережами чи виділяються характеристики символу, що зображений. На виході цього етапу отримується ймовірний варіант символу. Після чого робота може бути продовжена на основі інших методів для уточнення отриманого результату.

На першому етапі зазвичай виконується підвищення якості зображення шляхом застосування необхідних відновлювальних фільтрів. Другий етап починається з робіт по виявленню тексту на зображенні. Серед відомих підходів можна виділити наступні: на основі контурної інформації (скелетизація, виділення країв та кутів тощо), на основі кольорової інформації (метод гістограм, аналіз головних компонент) та на основі аналізу текстурної інформації (метод опорних векторів, штучні нейронні мережі, експертні системи) [3].

Наступним кроком є сегментація тексту, що відбувається у декілька етапів: виділення рядків, сегментація слів та сегментація символів. Для формалізації можемо представити вхідне зображення у градаціях сірого та як певну матрицю точок яскравості $A$: $A = \{a_{ij}\}, 0 \leq a_{ij} \leq a^{max}, i = 1..n, j = 1..m$, де $n$, $m$ – розміри зображення у пікселях.

Задача виділення рядків зводиться до знаходження верхніх та нижніх границь рядків тексту зображення [4]. Алгоритм сегментації рядків базується на

тому, що середня яскравість в зображеннях міжрядкових інтервалів значно нижча за зображення текстових рядків. Перш за все знаходиться середнє значення яскравості усіх піксельних рядків, а далі – усього зображення:

$$l_j = l_j(B) = \frac{1}{n}\sum_{i=1}^{n} a_{ij} \quad \rightarrow \quad l(B) = \frac{1}{m}\sum_{j=1}^{m} l_j(B) \tag{1.2}$$

Середня яскравість між рядками є невеликою. Таким чином яскравість верхньої та нижньої межі текстового рядку можна визначити через середню яскравість зображення (позначаємо $l^t$ та $l^b$ відповідно). Процес сегментації рядків полягає у перегляді масиву середніх значень $(l_1, \dots, l_m)$ та виявлені множини пар індексів $(l_i^t, l_i^b)$ піксельних рядків, що відповідають верхній та нижній граням зображення рядку номер $i$, що задовільнять наступні умови:

для верхньої границі:

$$(l_{i-2} < l^t) \wedge (l_{i-1} < l^t) \wedge (l_i > l^b) \wedge (l_{i+1} > l^b) \wedge (l_{i+2} < l^b) \wedge (l_{i+3} < l^b) \tag{1.3}$$

та для нижньої:

$$((l_i > l^t) \wedge (l_{i+1} > l^t)) \vee ((l_{i+1} < l^b) \wedge (l_{i+2} < l^b) \wedge (l_{i+3} < l^b)) \tag{1.4}$$

Таким чином формується множина пар індексів верхніх та нижніх границь рядків. Різниця між індексами – висота текстових рядків.

Перед початком процесу сегментації слів на основі отриманих рядків варто застосувати до вхідного зображення пороговий фільтр підвищення контрастності. Він дозволить знизити рівень шуму та значно поліпшить роботу алгоритму.

У свою чергу процес сегментації слів подібний до попередньо описаного, адже базується на тому, що середня яскравість в інтервалах між словами менша за середню яскравість зображення.

Виділення символів у рамках слова є більш складним процесом, адже часто символи в рамках слова розміщені близько один до одного та інтервали не виражені так яскраво, проте логіка роботи залишається колишньою. Пошук локальних мінімумів середньої яскравості відбувається на суміжних інтервалах зміни яскравості стовбців. Розмір інтервалу визначається висотою рядку та

зазвичай не перевищує 0.3 цього значення. Тобто розмір інтервалу визначаємо як $d_j = 0.3m$, де $m$ – висота слова.

Алгоритм сегментації символів можна описати наступним чином:

1. Для усіх піксельних стовбців знаходимо їх середнє значення яскравості $c_i$ за аналогічною до попередніх алгоритмів формулою;
2. Серед отриманих значень $c_i$ перший мінімум шукаємо на відрізку $i = 1, \dots, d_j$.
3. При знаходженні локального мінімуму для певного індексу $i^1_{min}$, наступний мінімум шукаємо на відрізку: $i = (i^1_{min} + 1), \dots, (i^1_{min} + 1 + d_j)$;
4. Процес повторюється до досягнення границі зображення слова. Усі значення, що відповідають локальним мінімумам, зберігаються у списку $W_0$.

Локальний мінімум яскравості у стовбці I є кандидатом на приналежність до міжсимвольного інтервалу, якщо значення середньої яскравості $c_i$ у ньому менше певної гранці яскравості $c_b$ і при цьому значення середньої яскравості у стовбцях на відстані двох пікселів зліва та справа більше за цю границю. Саму границю яскравості аналогічно визначаємо за середньою яскравістю:

$$c^b = k^b \frac{1}{n} \sum_{i=1}^{n} c_i(B) \tag{1.5}$$

де $0 < k^b < 1$ – коефіцієнт, а $n$ – ширина зображення слова.

Умова міжсимвольної границі тепер визначається наступним чином:

$$(c_i < c^b) \wedge ((c_{i-2} > c^b) \vee (c_{i+2} > c^b) \tag{1.6}$$

У результаті з попередньо отриманого списку $W_0$ вилучаємо стовбці, середня яскравість яких не задовільняє визначеній умові. Таким чином отримуємо зображення окремих символів, що можуть бути передані для безпосередньо розпізнавання.

Для цього можна використовувати повнозв’язну штучну нейронну мережу (див. рис. 1.4), що складається із вхідного, вихідного та прихованого шарів. Вхідний шар складається із $n*m$ нейронів, де $n$ – висота зображення, а $m$ –

ширина. Вихідній шар, відповідно, із $k$ нейронів, де $k$ – розмір алфавіту. Вихід нейронів одного шару є входом для іншого.

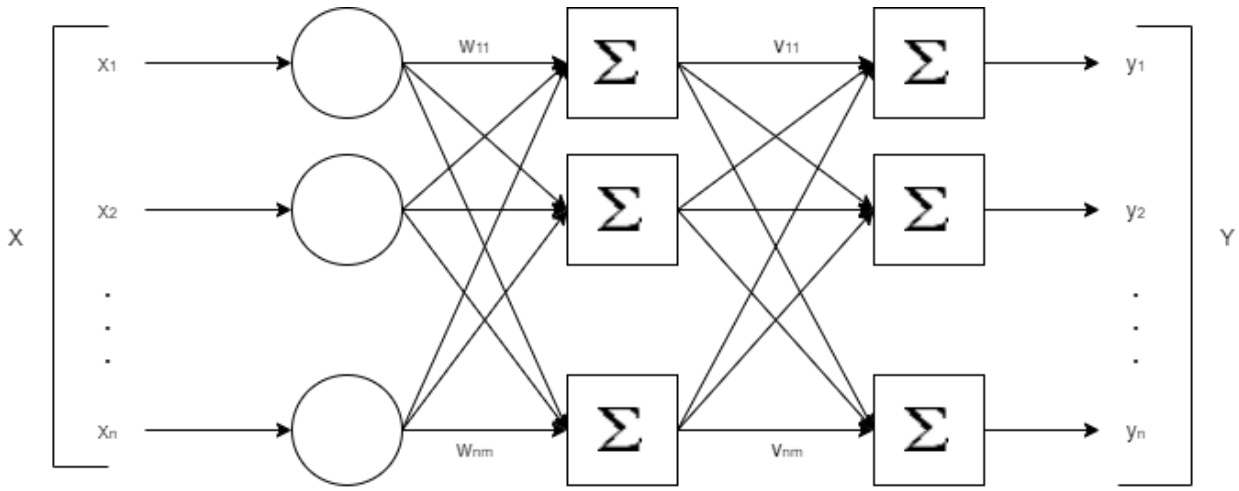



Рисунок 1.4. Повнозв’язна нейронна мережа

Математично нейрон – зважений суматор, у якого єдиний вихід визначається через його входи та матрицю ваг наступним чином:

$$y = f(u), u = \sum_{i=1}^{n} w_i x_i + w_0 x_0 \quad (1.7)$$

де $x_i$ та $w_i$ сигнали на входах нейронів та ваги входів. При цьому функція u називається індукованим локальним полем, а $f(u)$ – функція активації. Можливі значення сигналів на входах вважають заданими на інтервалі $[0,1]$ та визначаються як дискретні чи як аналогові. Додатковий вхід $x_0$ та відповідна йому вага $w_0$ використовуються для ініціалізації нейрону. У якості функції активації можна використовувати, наприклад, сигмоїдальну функцію, що є монотонно зростаючою, всюди диференційованою та дозволяє посилювати слабкі сигнали та не насичуватися від сильних.

Для коректної роботи нейронну мережу варто навчити. Одним із методів є метод зворотного поширення помилки. Його ідея полягає у поширенні сигналів помилки від виходів мережі до її входів, у напрямі, протилежному прямому поширенню сигналів у звичайному режимі роботи. Цей метод є модифікацією класичного методу градієнтного спуску. Після навчання моделі її можна використовувати для розпізнавання.

Взагалі навчання нейронної мережі зводиться до визначення зв’язків (синапсів) між нейронами і встановленню вагових коефіцієнтів. Інакше кажучи, сили цих зв’язків. Спрощено, алгоритми навчання нейронної мережі зводяться до визначення залежності силу зв’язку нейронів від числа прикладів, що підтверджують цю залежність [4].

Генерація тренувальної вибірки здійснюється шляхом застосування спотворених зображень, символів шрифту, який використовується у текстах, що розпізнаються. Серед операцій спотворення можна виділити операції масштабування (без збереження пропорцій вихідного зображення), обрізка кордонів зображення по всіх границях, операції повороту з певним кроком та зашумлення зображення символу. Перевага такого способу підготовки полягає у відсутності необхідності вручну проставляти мітки класів зображень, тому що завжди відомо зображення якого символу оброблюється в даний момент.

### 1.2.3 Переваги згорткових мереж для задач розпізнавання

Використання традиційного багатошарового персептрона для задач розпізнавання має ряд визначених недоліків [5]. Серед основних можна виділити:

- Велика розмірність вхідних зображень призводить до швидкого зростання кількості нейронів та мережевих синаптичних зв’язків. Разом із цим має збільшитися і навчальна вибірка, а за нею – час і складність процесу навчання.
- Ігноруючи мету навчання, компоненти вхідного шару представляються у довільному порядку. Таким чином фактично топологія вхідних даних не враховується, проте насправді вхідні зображення мають визначену двовимірну структуру, де наявна залежність між сусідніми пікселями з просторової точки зору.

Згорткові нейронні мережі, що заточені для розпізнавання двовимірних поверхонь з високими показниками інваріантності щодо різноманітних спотворень, представляють собою варіант архітектури багатошарового

персептрону, що включає у себе такі шари: згорткові, субдескретизації та повнозв'язні. Архітектура використовує переваги двомірної структури вхідних даних-зображень за допомогою методу локальної зв'язності. При цьому відбувається обмеження кількості зв'язків між нейронами прихованого згорткового шару та вхідними даними. Кожний нейрон прихованого шару пов'язаний тільки з обмеженою локальною (але такою, що не має розривів) ділянкою зображення.

Саму операцію згортки можна задати формулою:

$$(f \times \mathrm{g})[m,n] = \sum_{k,l} f[m-k, n-l] \cdot \mathrm{g}[k,l] \qquad (1.8)$$

де f – вихідна матриця зображення, а g – ядро згортки. Основна ідея у чергуванні згорткових та субдискретизуючих шарів. Сама мережа є односпрямованою та багатошаровою (див. рис. 1.5).

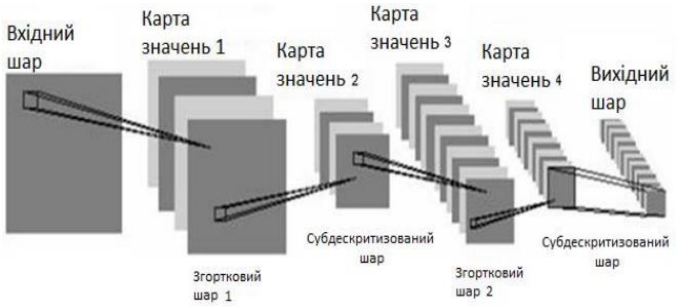



Рисунок 1.5. Базова структура згорткової нейронної мережі

Інваріантність до спотворень, масштабування чи зрушень досягається завдяки наступним концепціям реалізації:

- Вхідний сигнал від попереднього шару на кожний із нейронів йде від локального рецептивного поля. Така організація дозволяє досягти двовимірної зв'язності нейронів.
- Приховані шари мають безліч карт ознак, де нейрони мають загальні (shared) ваги. Це надає інваріантність до зміщень.
- Після кожного згорткового шару йде обчислювальний, що локально усереднює та робить підвибірку. Таким чином зменшується розширення карт ознак.

Згорткова нейронна мережа використовує спільні ваги, накладаючи штучне обмеження на алгоритм навчання зворотнім поширенням помилки таким чином, щоб кожний нейрон прихованого шару мав набір ваг, спільний з іншими нейронами цього шару. У випадку прямого поширення помилки така мережа виконує математичну операцію згортки вхідного зображення набором фільтрів, що надаються вагами прихованого шару. Проміжними результатами мережі є карти ознак – двомірні матриці, що представляють собою результат згортки окремим фільтром.

Шар субдискретизації виконує операцію групування карт ознак, розглядаючи регіони $p \times p$ та агрегуючи значення, отримані у результаті згортки. Таким чином знижується варіативність даних, що забезпечує стійкість до трансляцій локальної ознаки у межах окремого регіону. За умов, коли одна і та сама ознака є зсунутою на деякі значення $(\Delta x, \Delta y)$ у границях, що не перевищують $p$, відповідний нейрон, що інкапсулює локальну субдискретизовану ознаку зображення, буде все ще активним. Це забезпечує інваріантність до просторових спотворень. У якості агрегуючої функції шару субдискретизації можна використовувати функцію визначення середнього чи максимального значення.

Завдяки використанню декількох позмінних шарів згортки та субдискретизації, згорткова нейронна мережа дозволяє отримувати представлення, що незалежні від конкретного розміщення локальної ознаки у зображенні та однаковим чином реагувати на необхідні об’єкти, що присутні на довільному місці зображення. Операція згортки забезпечує інваріантність до трансляції локальних ознак по осях, проте варто мати на увазі, що архітектура згорткових нейронних мереж не передбачає стійкості до інших афінних перетворень, таких як обертання чи зеркальне відображення.

### 1.2.4 Запропонована нейронна мережа та її використання

У процесі розпізнавання сегментованих символів використаємо базову штучну нейронну мережу з одним прихованим шаром, якої достатньо для отримання хороших результати у розрізі поточного завдання (див. рис. 1.6). Побудована мережа є тришаровою мережею прямого поширення, для якої буде використано алгоритм навчання методом зворотного поширення помилки.

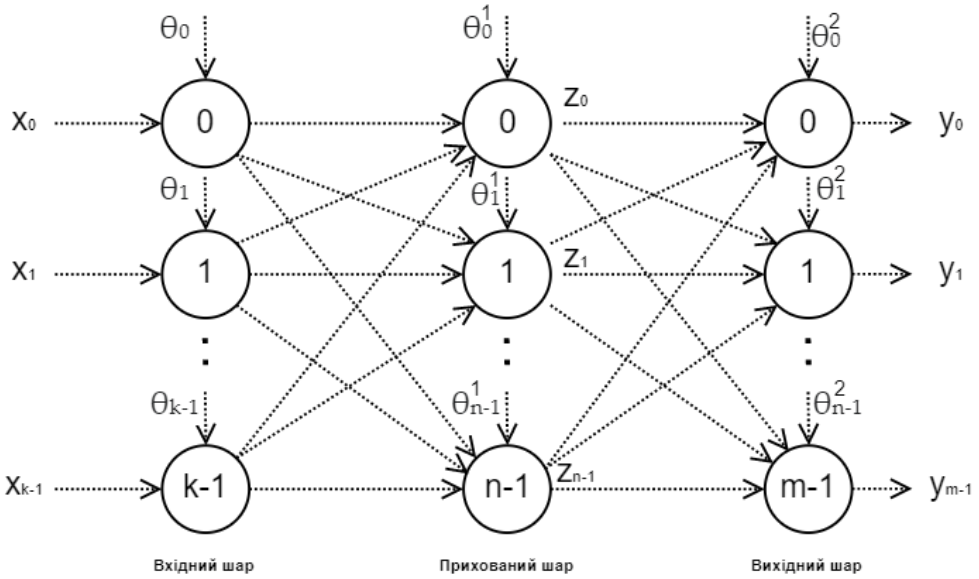



Рисунок 1.6. Структура мережі для розпізнавання символів знаку

На вхід до мережі символ подається у вигляді двовимірного масиву білих та чорних пікселів із зображення. Попередньо всі матриці приводяться до стандартизованого розміру (30 на 20 пікселів), розміщуючи символ у верхньому лівому куті матриці.

Разом із цим, щоб спростити аналіз складних структурних особливостей кожного символу, такі як точки з'єднання, ребра та цикли, кожен символ також проходить процес потоншення з використанням NWG алгоритму.

Елементи матриці зберігаються у вхідному масиві *x[k]*. Таким чином розмірність вхідного вектору рівна кількості елементів матриці із символом, що й визначає кількість нейронів вхідного шару. Таким чином $k = 30 * 20 = 600$.

Оптимальний розмір прихованого шару зазвичай знаходиться між розмірами вхідних та вихідних шарів або може становити 2/3 розміру вхідного

шару плюс розмір вихідного. Таким чином прихований шар був визначений розміром у $n = 300$ нейронів.

Розмір вихідного шару визначається кількістю символів, що використовуються у українських номерних знаках стандартного типу. Це 12 літер алфавіту та цифри від 0 до 9. Таким чином результуючий розмір – $m = 22$ нейрони.

На етапі тренування було використано 30 зразків кожного із символів, що загалом складає 660 наборів тестових даних. Запропонована мережа пройшла 500 ітерацій для кожного набору у процесі підбору параметрів для кожного шару. Таким чином було виконано 330,000 ітерацій для вхідних 660 наборів.

Після завершення навчання мережі та отримання відповідних значень ваги кожного з нейронів, вони використовуються для розпізнавання нових символів, видаючи результуючий масив *y[m]*. Якщо мережа навчена правильно, то одне зі значень вектору буде мати значення, що близьке до одиниці, а інші – до нуля.

# РОЗДІЛ 2
# ПРИЗНАЧЕННЯ ТА ВИМОГИ ДО СИСТЕМИ

## 2.1 Призначення та мета програмної системи

Призначення додатку «Parking Control» полягає в тому, щоб:

- забезпечити громадянам можливість у зручній та доступній формі повідомити вповноваженим службам щодо актів незаконного паркування, надавши усю нормативно затверджену інформацію;
- надати можливість експертам евакуаційної служби оперативно аналізувати надані звіти та відповідним чином реагувати на них;
- ввести систему адміністрування людських та транспортних ресурсів служби та аналітики ефективності роботи;
- налагодити механізм взаємодії відділу обробки звітів з евакуаційними екіпажами для ефективного інформування та реакції.

Метою програмної систему є зменшення кількості випадків незаконного паркування та підвищення ефективності роботи евакуаційних служб за сталої кількості ресурсів. При успішному впровадженні та інтеграції з відповідними службами, те чи інше місто має можливість налаштувати ефективну співпрацю з громадянами, значно підвищивши продуктивність реагування і таким чином забезпечити вищу якість життя громадянам без масштабних фінансових вливань на реформування.

## 2.2 Функціональні вимоги до системи

Виходячи із описаної мети та призначення програмної системи для проектування функціональних вимог було виділено чотири основні підсистеми:

- адміністратора евакуаційної служби;
- обробника звітів;
- евакуаційного екіпажу;
- громадянина.

Вимоги задавалися у тому числі шляхом створення діаграм прецедентів (див. додаток А). Далі у розділі наведені формалізовані функціональні вимоги щодо кожної із підсистем.

Адміністратор евакуаційної служби повинен мати широкі можливості щодо управління ресурсами та аналізу роботи служби. Відповідні функції наведені у табл. 2.1.

Таблиця 2.1

Функціональні вимоги до підсистеми адміністратора

| Функція | Завдання |
|---|---|
| Аналіз статистичних показників роботи служби | Можливість перегляду інформації за наступними показниками: скільки звітів з яких районів було отримано, відсоток підтверджених та завершених звітів, персональна статистика ефективності обробників та евакуаційних екіпажів. |
| Адміністрування людських ресурсів | Додавання, видалення, редагування, призначення на райони обробників звітів та евакуаційних співробітників, формування екіпажів. |
| Управління технічними ресурсами евакуації | Додавання, видалення, редагування інформації щодо евакуаційних машин та їх характеристик. Призначення машин до того чи іншого попередньо сформованого екіпажу. |

Обробник звітів у свою чергу відповідальний безпосередньо за аналіз та опрацювання інформації, що надходить від громадян. Функціональні вимоги щодо цієї підсистеми відображені у табл. 2.2.

У свою чергу підсистема громадянина має надати йому можливість сформувати та відправити всю необхідну інформацію до системи. У табл. 2.3 представленні необхідні функції щодо цього процесу.

Таблиця 2.2

Функціональні вимоги до підсистеми обробника звітів

| Функція | Завдання |
|---|---|
| Робота з профілем | Управління основною контактною інформацією про себе, перегляд особистої статистики щодо призначених звітів. |
| Перегляд призначених звітів | Перегляд звітів, що потребують обробки чи вже були опрацьовані, фільтрація за основними показниками, вибір конкретного для детального опрацювання. |
| Обробка звіту | Перегляд наданої інформації щодо звіту: інформація про локацію, водія, панорамні фото правопорушення. Можливість відповідної реакції: відмова у випадку відсутності правопорушення за думкою працівника (з наданням аргументації) чи підтвердження звіту із можливістю його призначення до евакуаційного екіпажу. Перегляд історичної інформації щодо уже оброблених звітів. |

Таблиця 2.3

Функціональні вимоги до підсистеми громадянина

| Функція | Завдання |
|---|---|
| Робота з профілем | Управління основною контактною інформацією про себе, перегляд особистої статистики щодо наданих звітів. |
| Звітування щодо правопорушення | Надання наступної інформації щодо правопорушення: фото номерів автомобіля правопорушника, фото панорами, місцезнаходження та опціональна додаткова інформація. |

| | |
|---|---|
| Інформування щодо результатів обробки | Можливість переглядати інформацію щодо того, який висновок надав експерт відповідного до того чи іншого звіту: у випадку відмови – причина, у випадку підтвердження – чи було відповідне реагування у вигляді евакуації автомобіля правопорушника. |

Останньою виділеною у процесі формування функціональних вимог є підсистема члена екіпажу евакуаційної команди. Сформовані функціональні вимоги зосереджені у табл. 2.4.

Таблиця 2.3

Функціональні вимоги до підсистеми працівника евакуації

| Функція | Завдання |
|---|---|
| Робота з профілем | Управління основною контактною інформацією про себе, перегляд особистої статистики щодо призначених звітів, складу своєї евакуаційної команди та автомобілів у розпорядженні. |
| Робота з призначеними звітами | Перегляд призначених звітів, фільтрація за основними показниками. Можливість надання інформації про виконання того чи іншого звіту чи надання пояснення відносно причин неможливості евакуації. |

## 2.3 Нефункціональні вимоги

Доступність. Кожна з користувацьким підсистем має бути доступна у будь-який час. У випадку запланованих технічних робіт чи аварійних ситуацій, користувачі системи мають отримувати відповідне сповіщення, а сам доступ до підсистем повинен відновитися впродовж доби.

Безпека. Доступ до підсистем має відбуватися виключно після успішної аутентифікації користувача. Сесійні токени та ключі на стороні клієнта мають

бути недоступними до програмного зчитування. Відповідні програмні запобіжники мають бути розроблені для захисту системи від XSRF та XSSI атак. Має бути забезпечена можливість блокування користувача довільної підсистеми. При цьому обмеження доступу у режимі реального часу не є критичним, а відповідна реакція клієнтської підсистеми має відбуватися при наступному після блокування запиту на API загального серверу системи. Комунікація між підсистемами має відбуватися виключно на основі захищеного HTTPS протоколу.

Локалізація. Інтерфейс має бути доступним українською та англійською мовами. При цьому задля запобігання розбіжностей у трактуванні, значення дат у рамках системи відображаються у міжнародному форматі – YYYY-MM-DD.

Продуктивність та швидкодія. Стандартна конфігурація системи має витримувати одночасну роботу 10 експертів та автоматичну обробку 20 наданих громадянами звітів. При цьому час відображення наданого громадянином звіту у експертній підсистемі не має перевищувати 5 хвилин. Ретроспективний термін зберігання оброблених звітів – 6 місяців.

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ СИСТЕМИ «PARKING CONTROL»

## 3.1 Розробка бізнес-процесів та архітектури програмної системи

У ході вибору підходів до побудови клієнтських інтерфейсів, що мають надавати заданий у вимогах функціонал, було сформовано два положення:

1. Для громадян та членів евакуаційної команди найбільш вдалим інструментом для взаємодії із систему є мобільний телефон, що наразі у більшості є під рукою. Разом із цим адаптивні веб-сайти часто не надають того нативного досвіду користування, як мобільні додатки.
2. Для обробників звітів та адміністраторів має місце більш серйозне рішення, адже їм надається більше функціональне навантаження. Оскільки десктоп рішення потребують операційних потужностей та додаткових кроків для впровадження та підтримки, перевагу слід надати інтерактивному веб-додатку.

Таким чином, інтерфейси взаємодії користувачів сформовані у вигляді двох компонент: веб-додаток та мобільний додаток.

Виходячи із аналізу предметної області та функціональних вимог було розроблено архітектуру, що продемонстрована на рис. 3.1.

Загальний алгоритм роботи системи можна описати наступним чином:

1. Громадянин надає дані. Через API доступу до даних, створюється звіт у базі даних системи. Разом із цим фото панорами та номеру автомобіля завантажуються до Azure Blob Storage. Унікальний ідентифікатор звіту співпадає з іменами завантажених до сховища фотографій. Він є основним ключем доступу до інформації щодо звіту як у Blob Storage, так і у базі даних системи.
2. Завантаження файлу до Blob Storage активує виконання Azure Durable функції, що виконує наступні кроки: передає завантажене фото номерного знаку до Azure Cognitive сервісів, що виконують розпізнавання номерного знаку та повертають розпізнане значення. За цим номером наступна Azure

функція робить запит до державного реєстру та знаходить дані про автомобіль та його власника. Після чого записує дані до бази системи, використовуючи унікальний ідентифікатор, що наданий поточному звіту.

3. Після попередніх двох кроків ми отримуємо остаточно сформовані дані щодо звіту у базі даних системи і таким чином веб-додаток надає інформацію обробнику. У той же час фото панорами береться безпосередньо з Blob Storage, де й було попередньо розміщено. Опрацювання звіту відбувається шляхом безпосередньої взаємодії веб-додатку з базою даних системи, адже окремого API доступу, як наприклад для мобільного додатку, веб-додаток не потребує.

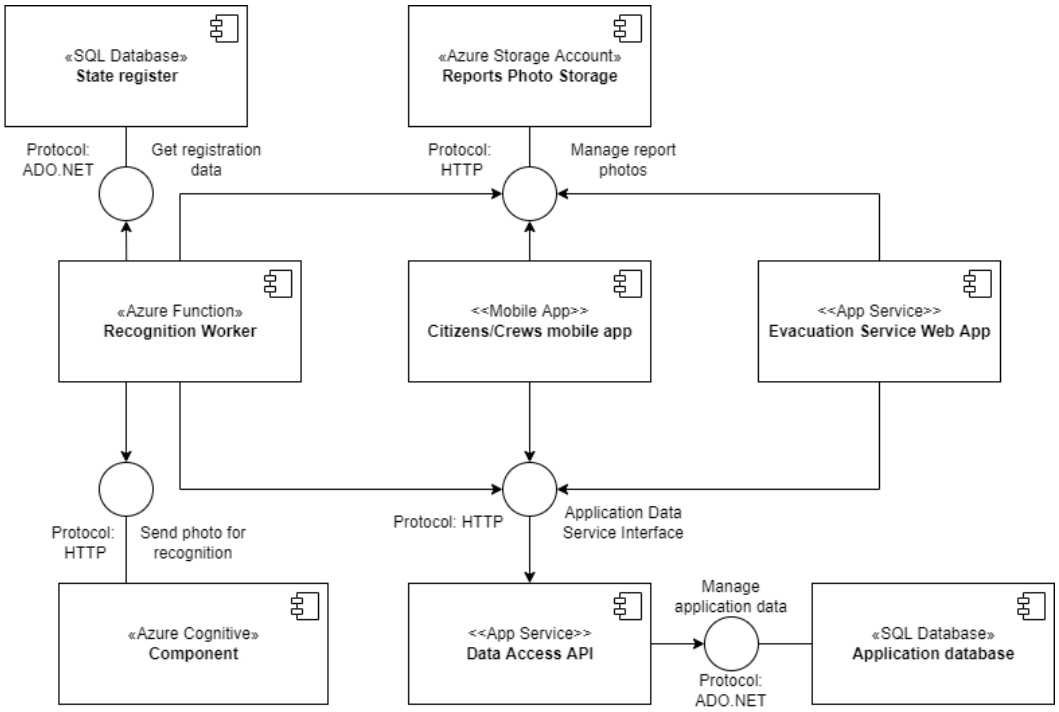



Рисунок 3.1. Архітектура програмної системи

Розроблена архітектура, завдяки своїм модульності та елементам безсерверних обчислень у вигляді Azure функцій, дозволяє отримати ефективну систему, що здатна до легкого розширення як з точки зору навантажень, так і щодо розширення тих бізнес-процесів та логіки, що вона підтримує. В цілому, при накопиченні достатньо великої кількості спільної логіки доступу до даних між веб-додатком та API мобільного додатку, можливий варіант з повним її перенесенням до API, що в свою чергу дозволить позбавитися дублювання та

інкапсулювати логіку доступу. Це у свою чергу надасть змогу легше додавати нові користувацькі інтерфейси взаємодії, адже вони не матимуть необхідності реалізовувати логіку взаємодії з даними.

Безпосередньо розподіл етапів процесу обробки звіту між самостійними програмними та інфраструктурними підсистемами зображено на рис. 3.2.

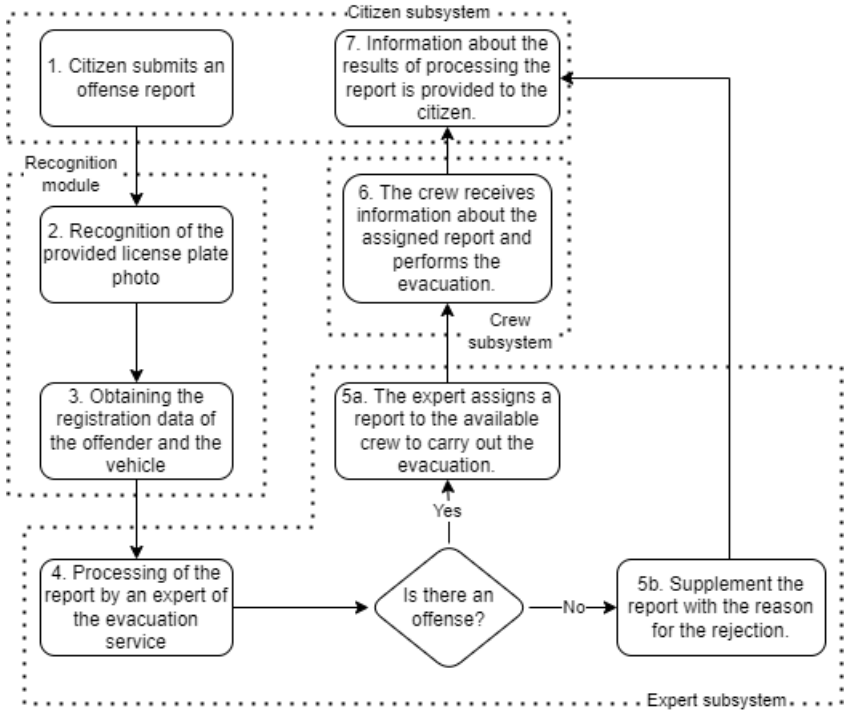



Рисунок 3.2. Процес обробки звіту у розрізі підсистем

## 3.2 Проектування бази даних системи

Створення бази даних, що відповідала б основним вимогам предметної області та на ряду з цим мала зручну для програмної взаємодії та оптимізовану структуру, відбувалося шляхом виконання наступних етапів:

1. Додатковий аналіз предметної області, що дозволяє виділити основні сутності та зрозуміти структуру їх взаємодії.
2. Етап інфологічного проектування, що полягає у частково формалізованому описі об’єктів предметної області у термінах деякої семантичної моделі.
3. Даталогічне проектування.
4. Фізичне проектування, тобто відображення логічної структури бази даних у структуру зберігання.

### 3.2.1 Аналіз предметної області

У якості підходу був обраний компромісний варіант об’єднання функціонального та предметного підходу.

З одного боку уже перед етапом проектування бази даних, були відомі функції та комплекси задач, для обслуговування яких створюється база даних та було виділено мінімальний набір об’єктів предметної області під цей опис. З іншого боку, при виділенні сутностей та взаємозв’язків приділялася увага можливості подальшого розширення програмної системи додатковим фунціоналом.

Виходячи із предметної області та бізнес-процесу, що автоматизується системою, були послідовно виділені сутності бази даних системи, що покривали задачі, описані у функціональних вимогах (див. табл. 3.1).

Таблиця 3.1

Основні сутності бази даних системи

| Сутність | Опис |
|---|---|
| Звіт | Сукупність інформації, що надається громадянином щодо зафіксованого правопорушення. Є центральною сутністю системи. |
| Правопорушник | Основні дані про автомобіль та його власника, необхідні для обробки та евакуації. |
| Обробник | Сутність, що втілює особу обробника звітів. |
| Евакуаційний співробітник | Сутність, що втілює особу співробітника евакуаційної служби, який займається евакуацією. |
| Евакуаційний екіпаж | Об’єднання декількох евакуаційних співробітників, що утворює команду та є безпосередньо сутністю, якій призначається звіт до виконання. |
| Евакуаційний автомобіль | Сутність, що втілює основні евакуаційні характеристики того чи іншого евакуаційного |

| | |
|---|---|
| | автомобіля для ефективного призначення відповідно до специфіки транспортного засобу, що потребує евакуації. |

3.2.2 Інфологічне проектування

Етап мав за мету побудову інформаційної моделі найвищого рівня абстракції без орієнтації на якусь конкретну СКБД чи модель даних.

Серед широкого спектру моделей для подання даних, що визначають вид і зміст інформаційної моделі, була обрана модель «Сутність-зв'язок» що стала наразі фактичним стандартом в інфологічному моделюванні. Вибір аргументований тим, що цей тип моделей є наглядним, дозволяє проектувати БД з великою кількістю об'єктів та атрибутів, а також реалізований в багатьох системах автоматизованого проектування баз даних.

ER-модель складається із сутностей, зв'язків між ними, атрибутів, їх доменів та ключів. Кожен об'єкт предметної області характеризується певними набором атрибутів, що відображає його властивості. Атрибути, у свою чергу, використовуються для визначення того, яка інформація повинна бути зібрана щодо того чи іншого об'єкту.

Результат інфологічного проектування наведений на рис. 3.3 у вигляді спрощеної ER-діаграми. Окрім виділених у результаті аналізу сутностей, також на даному етапі модель бази даних враховувала сутність громадянина, а також необхідність певної структури для збереження адрес для того щоб надавати можливість надавати інформацію щодо локації правопорушення. Між наведеними сутностями існує один із можливих зв'язків: один до одного, один до багатьох, багато до одного або багато до багатьох. Опис деяких з них наведений у таблиці 3.2.

Таблиця 3.2

Опис зв’язків між сутностями

| Сутності | Зв’язок | Пояснення |
|---|---|---|
| Громадянин – Звіт<br>Обробник – Звіт<br>Екіпаж - Звіт | 1 до N | Громадянин може надавати, обробник опрацьовувати багато звітів. Екіпаж проводити евакуацію відповідно до багатьох звітів. У той же час звіт має лише одного звітувальника, обробника та виконавця. |
| Обробник – Локація<br>Екіпаж – Локація | N до N | Обробник може бути прикріплений певних районів, що зосереджують у собі локації. У той же час локації можуть опрацьовуватися одночасно декількома обробниками. Аналогічна ситуація з евакуаційними екіпажами. |
| Звіт – Інформація про правопорушника | 1 до 1 | Звіт може мати лише одну відповідно до нього сутність, що зосереджує інформацію про правопорушника. У той же час, звісно, сутність інформації є слабкою сутністю та має мати хоча б один та лише один звіт у зв’язку. |

Разом із цим деякі атрибути на приведеній діаграмі позначені як складні з метою компактності візуалізації (тобто деякі з них все ж утворюють пласку структуру зі своєю моделлю, а не є окремою ієрархічною структурою). Розгорнутий зміст відносно кожного атрибуту приведений у табл. 3.3.

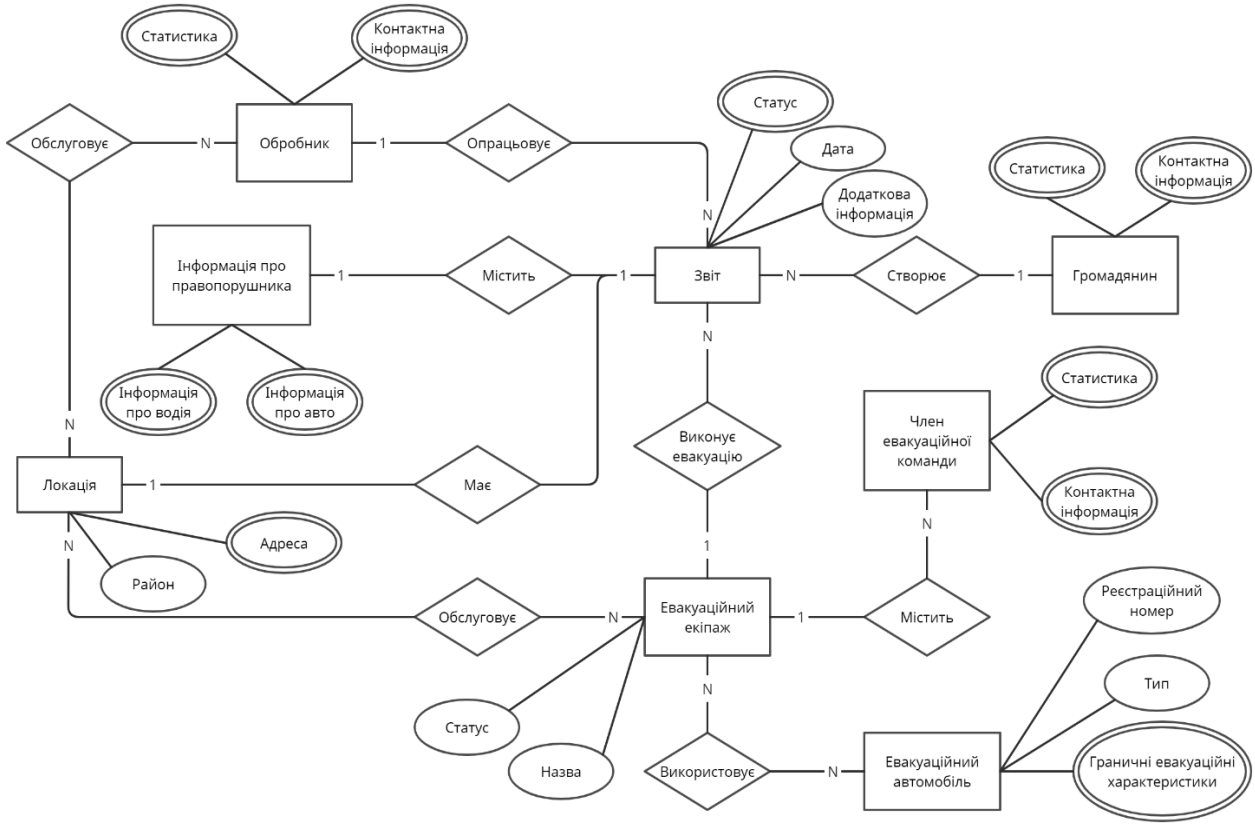



Рисунок 3.3. ER діаграма

Таблиця 3.3

Зміст складних атрибутів ER-діаграми

| Атрибут | Сутності | Зміст |
|---|---|---|
| Контактна інформація | Громадянин<br>Обробник<br>Член екіпажу | Складний атрибут, наявний у кожної сутності, що є користувачем системи: громадянин, обробник, член евакуаційної команди. В цілому структура спільна для кожного та містить у собі ім’я, контактний телефон, email та допоміжну інформацію профілю. |
| Статистика | Громадянин<br>Обробник<br>Член екіпажу | Кожна сутність, що оперує звітами має аналітичну статистику, що відображає загальну кількість звітів (з якою користувач у |

| | | |
|---|---|---|
| | | якомусь зв’язку), яка частина з них була оброблена, підтверджена та виконана. |
| Адреса | Локація | Складена інформація про локацію, що містить район, адресу та її номер. |
| Статус | Звіт | Інформація щодо того, чи був поточний звіт оброблений, підтверджений та виконаний. |
| Інформація про водія, авто | Інформація про правопорушника | Реєстраційні дані про транспортний засіб (тип, модель, номер, розміри тощо) та його водія. |
| Граничні евакуаційні характер. | Евакуаційний автомобіль | Максимальні значення ширини, висоти, довжини та ваги транспортного засобу, що може бути евакуйований цим автомобілем. |

### 3.2.3 Даталогічне та фізичне проектування

На етапі логічного проектування була створення схема бази даних на основі реляційної моделі шляхом виконання наступних кроків: перетворення концептуальної моделі у логічну модель даних, аналіз отриманої моделі на функціональну повноту, уточнення обмежень цілісності бази даних та графічного подання логічної моделі у вигляді їх атрибутів, обраних первинних та зовнішнім ключів та зв’язків між таблицями.

База даних проекту реалізовувалася на базі Microsoft SQL Server, була створена підходом Code First у Entity Framework Core та модифікувалася шляхом механізму послідовних міграцій змін у моделях сутностей. Схема бази даних візуалізована засобами Microsoft SQL Server Management Studio та наведена у додатку Б.

Основні сутності та їх зв’язки були реалізовані згідно сформованої на інфологічному етапі схеми. Разом із тим складні атрибути у деяких випадках трансформувалися у одну пласку модель з більшою кількістю властивостей, а деякі – у ієрархічну структуру таблиць. Наприклад, інформація про правопорушника була імплементована у вигляді однієї таблиці ReportVehicles,

що містить у собі усі необхідні властивості відповідно до вказаних атрибутів. У свою чергу логіка локацій представляє собою три таблиці: район, вулиця та конкретний її номер (Districts, Addresses, Locations). Подібна реалізація має місце з точки зору оптимізації, адже на відміну від першого прикладу, тут не випадок зв'язку «один до одного». Райони містять багато вулиць, вулиці – багато номерів.

Окремої уваги заслуговує реалізація моделей, що є частиною бізнес-логіки та мають безпосередню прив'язку до звітів, але за сумісництвом є користувачами системи та приймають участь у процесі авторизації. Для того щоб не поєднувати модель для фізичного користувача системи із сутністю, наприклад, обробника, була використана наступний двоетапний варіант:

1. Користувач системи при реєстрації (або створенні адміністратором) на даному етапі розробки вноситься у таблицю саме користувачів системи. Вона містить у собі сутності, що безпосередньо приймають участь у процесі авторизації та не зосереджують у собі ніякої бізнес-логіки стосовно процесів додатку.
2. Відповідно до ролі користувача вводиться запис до таблиці, що відповідає за сутність конкретної ролі у системі (ReportsHandlers, EvacuationCrewmates тощо) і вже до яких виконується прив'язка звітів та навколо яких реалізуються бізнес-процеси, специфічні для рішення. Кожний такий запис, відповідно, має посилання на того чи іншого користувача системи із вищеописаної таблиці базових користувачів.

Зв'язки типу «багато до багатьох», як у випадку з групами та парами, реалізовані за допомогою введення допоміжних таблиць. Правила інтерпретації моделей Entity Framework Core та гнучкий засіб модифікації Fluent API дозволяють генерувати та динамічно модифікувати бази без використання SQL-запитів безпосередньо у спеціалізованому програмному забезпеченні.

### 3.3 Розробка веб-частини

Після формування функціональних вимог та проектування архітектури, наступним питанням до вирішення стає вибір технологій розробки. Це питання

особливо гостро стоїть у сфері технологій для веб-розробки, адже нині це є одним з тих напрямків, що мають найбільш активні темпи розвитку. Виходячи з того, що на будь-якому ринку попит породжує пропозицію, зараз існує велика кількість опцій навіть у рамках сфери однієї мови програмування чи платформи.

### 3.3.1 Вимоги до технології згідно з поставленою задачею

Серед вимог, що були поставлені до функціональних та технічних можливостей технології можна виділити наступні:

1. Широкі можливості щодо створення зручного та інтерактивного користувацького інтерфейсу.
2. Ефективна інтеграція з хмарною платформою Microsoft Azure та відповідними сервісами як основними інструментами хостингу, розпізнавання та забезпечення фрагментів безсерверного функціонування.
3. Низький рівень вимог щодо потужностей клієнтських машин для комфортного використання додатків.

Виходячи із необхідністю ефективної інтеграції з Azure, а також компетенції відповідно до наявних технологій, було прийнято рішення в першу чергу зосередитися на платформі .NET і тих інструментах, що вона пропонує для створення сучасних та інтерактивних веб-додатків.

Наразі основним інструментом є ASP.NET Core та, відповідно, основні його варіації: MVC, Razor Pages та Blazor. Проаналізувавши можливості технологій щодо створення інтерактивного та сучасного інтерфейсу без необхідності у широкому використанні JavaScript, а також ряду інших можливостей, що будуть описані далі, перевага була надана саме фреймворку Blazor.

### 3.3.2 Вибір платформи Blazor

Blazor являє собою технологію створення веб-додатків з інтерактивним користувацьким інтерфейсом, які можуть працювати як на стороні серверу, так і на стороні клієнта. При цьому при визначенні коду, у тому числі на клієнтській

стороні, використовується C# замість JavaScript. Це у свою чергу надає можливість використовувати потужності платформи .NET на усіх етапах розробки, починаючи з серверної частини і закінчуючи клієнтською [7].

Наразі Blazor розділений на дві повноцінні підсистеми:

1. Blazor Server, що дозволяє створювати веб-додатки, які відпрацьовують на стороні серверу;
2. Blazor WebAssembly для створення додатків клієнтської сторони, що запускаються безпосередньо у браузері користувача.

Таким чином, після вибору серед .NET фреймворків, необхідно зробити вибір бажаної платформи Blazor, яка найкращим чином відповідає вимогам.

Blazor WebAssembly має можливість виконуватися безпосередньо у браузері клієнта так, як це роблять JavaScript фреймворки, такі як React, Vue чи Angular. Це стає можливим завдяки використанню нового відкритого стандарту – WebAssembly (WASM). Ця технологія була представлена міжнародним консорціумом W3C та розробниками від Mozilla, Microsoft, Google та Apple. Вона являє собою набір інструкцій для стекової віртуальної машини, що можуть бути ціллю компіляції для високорівневих мов програмування, серед яких є і C#. Код WebAssembly має доступ до повної функціональності браузера через JavaScript завдяки JavaScript Interoperability. У свою чергу код .NET, що виконується через інструкції WASM, запускається у ізольованому програмному середовищі браузера. При побудові та запуску додатку Blazor WebAssembly, файли с кодом C# та Razor компілюються у збірки .NET. Після чого скрипт завантажує середовище виконання та збірки до клієнтського браузера. Таким чином додаток фактично відпрацьовує на стороні клієнта.

Blazor Server у свою чергу відпрацьовує на стороні серверу. Оновлення елементів користувацького інтерфейсу та обробка подій виконується шляхом взаємодії клієнта та сервера через SignalR (див. рис. 3.4). Тобто коли користувач взаємодіє з додатком у браузері, викликаючи події користувацького інтерфейсу, то клієнтська сторона відправляє інформацію про подію, сервер її опрацьовує та

відправляє клієнту у відповідь інструкції щодо того, як необхідно оновити елементи інтерфейсу [7].

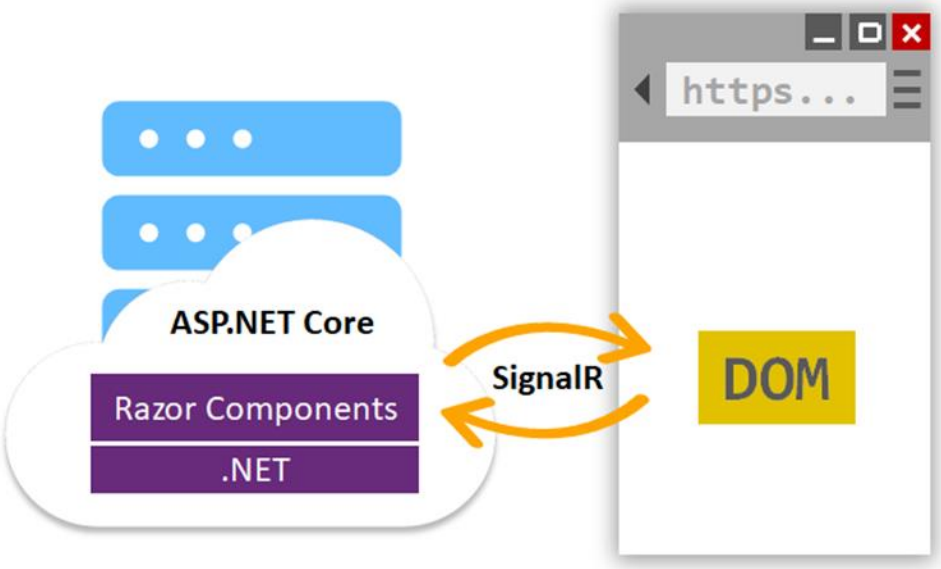



Рисунок 3.4. Принцип роботи Blazor Server додатків

Розглянувши основні принципи роботи обох підсистем, варто згадати про вимогу, що система не повинна мати великих вимог щодо машин користувача. Blazor WebAssembly, незважаючи на переваги у вигляді швидкодії та можливості функціонування без підключення до мережу, потребує підтримку браузером користувача технології WebAssembly, а також певних потужностей комп'ютера, адже додаток відпрацьовує на стороні клієнта. Виходячи із цього вибір був зроблений на користь саме Blazor Server варіанту.

### 3.3.3 Аспекти використання фреймворку

Побудова додатку відбувалася на основі компонентів, що представляють собою деякий самостійних елемент інтерфейсу. Подібний підхід дозволяє інкапсулювати логіку рендерингу та взаємодії з користувачем, а також повторно використовувати компоненти за необхідності, як це відбувається у популярних Javascript фреймворках клієнтської сторони.

Клас компоненту представляється у вигляді Razor розмітки з відповідним розширенням. Подібний підхід дозволяє комбінувати HTML розмітку з кодом C#

і таким чином отримувати динамічну генерацію інтерфейсу. На рис. 3.5 можна бачити фрагмент коду, що відповідає за відображення списку звітів для обробника. Таким чином ми маємо можливість в якості бажаного змісту тих чи інших елементів розмітки задавати властивості моделі.

```
<div class="row" style="margin-bottom: 15px">
    <div class="col-6">
        <div><b>Date: </b></div>
        <p>@report.ReportedDate</p>
    </div>
    <div class="col-6">
        <div>
            <b>Is processed: </b>
            <p>@(report.IsProcessed ? "Yes" : "No")</p>
        </div>
    </div>
</div>
```

Рисунок 3.5. Демонстрація використання Razor розмітки

Компонент, як і звичайні класи, може містити змінні, зо зберігають стан компонента, та методи, що відповідають за його логіку. Вони у свою чергу задаються в рамках спеціального блоку @code. Наприклад, отримання моделі відбувається шляхом ін’єкції відповідного сервісу та виклику методу підвантаження із бази даних у одному із методів, що відповідає певному етапу життєвого циклу компоненту. Значення, що повертається з сервісу, зберігається у бажаній властивості компоненту для використання у розмітці (див. рис. 3.6).

Разом із цим можна бачити, що властивість ReportId визначена спеціальним атрибутом [Parameter]. Це означає, що ідентифікатор звіту, що підлягає відображенню, передається з батьківського компоненту і визначається як звичайний атрибут тегу. Подібна параметризація дозволяє зробити компоненти більш гнучкими та підходящими до повторного використання.

```
[Parameter]
public string ReportId { get; set; }

private Report report;
private string panoramaPhotoLink;

protected override async Task OnInitializedAsync()
{
    var guidReportId = Guid.Parse(ReportId);
    report = await ReportService.GetReportByIdAsync(guidReportId);

    panoramaPhotoLink = ReportService.GetReportPanoramaLink(guidReportId);
```

Рисунок 3.6. Визначення логіки компоненту

За аналогічним підходом відбувається побудова динамічного інтерфейсу всього додатку. Повний приклад задання однієї із сторінок додатку можна побачити у додатку В.1.

### 3.4 Azure Cognitive сервіси для нейромережевого розпізнавання

Одним із архітектурних завдань, що мало бути вирішеними – використати можливості комп’ютерного зору для процесу розпізнавання номерних знаків правопорушника задля позбавлення користувача необхідності вводити його вручну, не маючи відповідної кваліфікації у розробці подібних систем власноруч.

Платформа хмарних сервісів Microsoft Azure надає таку можливість. Azure Cognitive Services – це заздалегідь створені API, які доступні розробникам, щоб допомогти у створенні інтелектуальних програмних систем не маючи безпосереднього досвіду у машинному навчанні чи експертизи у сфері AI. У звичному процесі машинного навчання розробник повинен зібрати навчальний набір даних, а далі перевірити валідність набору за допомогою відповідних моделей, після чого оцінити результати. У випадку з Cognitive services задача простіша – надати дані, але і те лише за необхідності специфічних потреб тих чи інших бізнес-процесів. Навчання та пошук найкращих моделей відбувається автоматично [8].

У контексті задачі необхідним до розгляду був функціонал цих сервісів у аспекті комп’ютерного зору, а саме визначення об’єктів та розпізнавання

символів для того щоб спершу точно визначити область номерного знаку, а після – розпізнати безпосередньо його значення.

Принцип роботи API визначення об’єктів досить простий – отримуючи на вхід бажане зображення, у відповідь отримуємо координати прямокутника (у пікселях), що обмежує номерний знак. Далі стандартними засобами .NET необхідно обрізати початкове зображення згідно результатів розпізнавання та отримане зображення знову відправити на обробку API, яке в свою чергу уже виконає процес розпізнавання символів.

Разом із бібліотекою CognitiveServices ми отримуємо SDK, основним інструментом якого є ComputerVisionClient, що в свою чергу має широкий набір методів відповідно до можливостей платформи. Для використання клієнту необхідно мати кінцеву точку підключення до зареєстрованого на порталі Microsoft Azure когнітивного сервісу, що передається як параметр конструктора. Для зручного використання було розроблено універсальний метод, що приймає на вхід бажаний метод розпізнавання та визначає стратегію повторюваного виклику.

Таким чином для, наприклад, розпізнавання уже підготовленого зображення номерного знаку у метод варто передати як метод RecognizeTextAsync, де параметрами передається посилання на зображення (у випадку поточного додатку – посилання на файл додатку у Blob Storage) та тип розпізнавання тексу. У випадку з номерними знаками тип був визначений як «Printed». Метод поверне об’єкт типу TextOperationResult, що містить результат розпізнавання у вигляді масиву рядків розпізнаного тексту, перший і єдиний з яких і буде бажаним значенням розпізнаного номерного знаку. Повний фрагмент класу, що відповідає за розпізнавання символів підготовленого зображення номерного знаку приведено у додатку В.2.

### 3.5 Azure Functions як інструмент безсерверних обчислень

Вирішивши питання щодо інструментів розпізнавання номерних знаків, необхідно було розв’язати архітектурну задачу щодо того, який компонент

системи буде виконувати роботу щодо розпізнавання номерного знаку, звернення до бази реєстру за інформацією та запису до бази даних системи відповідно до ідентифікатору звіту. Концептуально ця частина бізнес-процесу не повинна відноситися ні до веб-додатку, ні до мобільного додатку та його API, адже є окремою структурною одиницею логіки. Разом із цим створення окремого обчислювального серверу для подібної задачі не є раціональним рішенням.

Платформа Azure має найбільш підходяще рішення для подібних задач – Azure функції. Це безсерверний концепт, що дозволяє розгортати та виконувати фрагменти коду без потреби у серверній інфраструктурі, веб-сервері чи якихось додаткових конфігураціях [9].

### 3.5.1 Azure Durable Functions для організації бізнес-процесу

Разом із цим необхідно зважати на те, що за ідеєю кожна функція представляє собою окремий логічний крок, етап обробки чи обчислень, а описаний бізнес-процес, що необхідно винести, складається з трьох. Таким чином, користуючись лише базовою версією Azure функцій та бажаючи дійсно зберегти модульність та виконання принципу єдиного обов’язку, варто створити три окремі функції, що будуть виконуватися почергово. Для реалізації взаємодії можна, звісно, використовувати повідомлення у чергах, коли одна функція відправляє по завершенню своєї роботи повідомлення, а наступна активується за його отримання, але подібний підхід є надмірною складністю.

Описаний бізнес-процес почергових операцій є звичайним шаблоном ланцюга функцій (див. рис. 3.7) і для вирішення подібних задач існує розширення Azure Durable Functions. Воно дозволяє створювати функції з підтримкою стану у безсерверному середовищі та визначати бажані робочі процеси завдяки функції оркестрації.

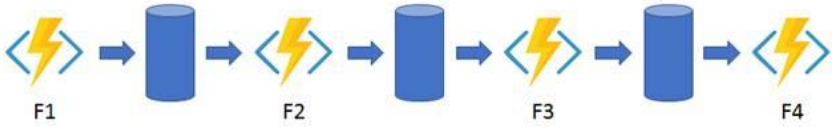



Рисунок 3.7. Шаблон ланцюга функцій

### 3.5.2 Аспекти реалізації процесу розпізнавання

Перш за все необхідно задати клієнт оркестратора. Це функція, що відповідальна за запуст та зупинку оркеструючої функції та її моніторинг. Саме клієнтська функція задає яким саме чином буде викликатися виконання. Виходячи із бізнес-процесу, процес розпізнавання та завантаження даних повинен відбуватися після завантаження клієнтським мобільним додатком файлу з номерним знаком до сховища. В цілому одним із варіантів у такому випадку було б використання Blob Storage Trigger, що викликав би виконання при завантаженні файлів, проте з точки зору зручності та ефективності реалізації було прийнято рішення на користь звичайного Http Trigger, на кінцеву точку якого звертається мобільний клієнт після завантаження. Оперуючи IDurableOrchestrationClient, клієнтська функція запускає необхідний оркестратор.

Функція оркестрації, безпосередньо, задає бажаний бізнес-процес. У поточному випадку він полягає у почерговому виклику трьох функцій активності через IDurableOrchestrationContext. Кожна така функція є контейнером для фрагменту незалежної логіки та відповідає за свій окремий модуль обробки:

1. Приймає ідентифікатор звіту та відправляє відповідний файл з номерним знаком із blob сховища до когнітивних сервісів, що описані у попередньому підрозділі, та отримує результат.
2. Приймає розпізнаний номерний знак та звертається до бази реєстру, повертаючи у моделі інформацію про автомобіль та особистість правопорушника.
3. Приймає модель, що зосереджує необхідну інформацію, та виконує запис до бази даних системи.

Після успішного виконання всіх трьох функцій процес є завершеним, а база даних системи містить повну інформацію щодо поточного звіту. Фрагменти реалізації функції клієнту оркестратора, самого оркестратора та деяких функцій активності наведені у додатку В.3.

## 3.6 Розробка мобільного додатку

Виходячи із того, що обрані вище компоненти архітектури так чи інакше відносяться до екосистеми розробки Microsoft, то вибір технології мобільної розробки, що найкращим чином буде взаємодіяти з рештою архітектури, не був надто складним, адже стек інструментів розробки цієї екосистеми має варіант і для створення мобільних додатків – Xamarin Forms. Окрім цього, ця платформа надає можливість створювати додатки з єдиною логікою одразу для трьох платформ: Android, iOS та UWP.

### 3.6.1 Архітектура роботи Xamarin.Forms

Xamarin працює на платформі Mono, яка може працювати з Linux, iOS та іншими. На рівні кожної платформи Xamarin використовує ряд субплатформ, таких як Xamarin.Android та Xamarin.iOS, через які додатки можуть напряму відправляти запити до прикладних інтерфейсів на пристроях [10].

З Xamarin.Android код C# компілюється у Intermediate Language, який при запуску створює нативну збірку. Код напряму не може звертатися до API Android та для цього використовує простори імен, що надаються Android Runtime. Спеціальний прошарок Managed Callable Wrappers (MCW) дозволяє транслювати виклики у нативні та звертатися до просторів імен Android.* та Java.*. І навпаки – Android Runtime звертається до додатку на Xamarin через прошарок Android Callable Wrappers.

Додатки Xamarin.iOS у свою чергу замість JIT-компіляції використовують AOT-компіляцію (Ahead-of-Time) коду C# у нативний ARM-код. Селектори транслюють виклики коду Objective-C в код на C#, а реєстратори – навпаки. Таким чином ці два компоненти і утворюють прошарок взаємодії, що на ілюстрації визначений як «bindings».

Загальна архітектура Xamarin відображена на рис. 3.8.

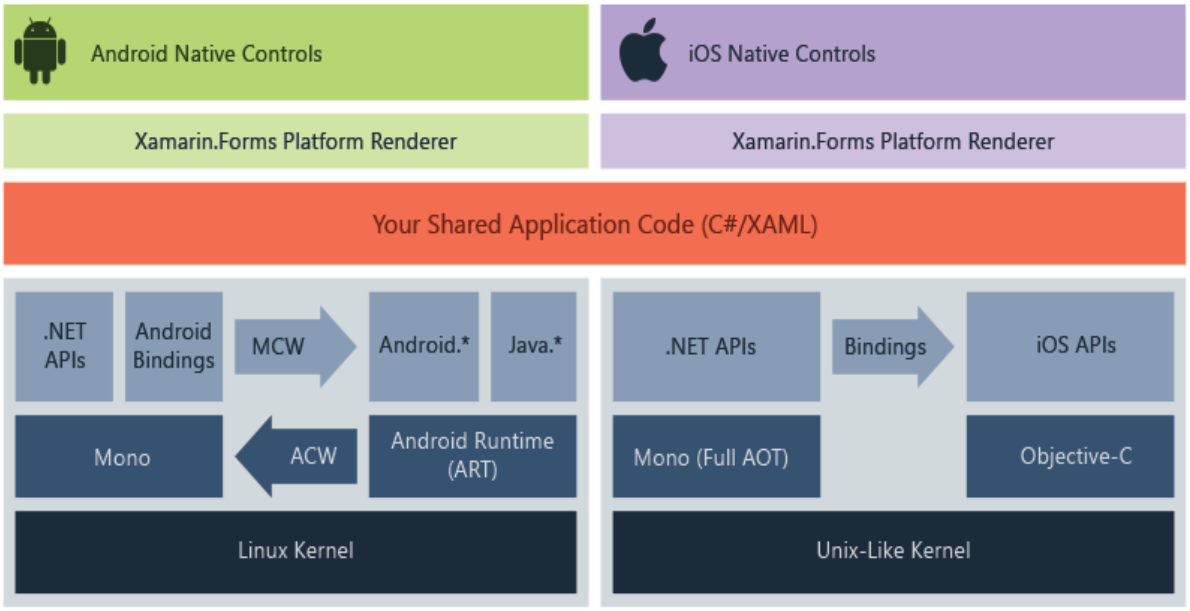



Рисунок 3.8. Архітектура роботи Xamarin

### 3.6.2 Підходи та програмні рішення розробки

У якості інструменту створення інтерфейсу був обраний варіант з його описом у XAML, що є мовою розмітки на основі XML. Порівнюючи з можливістю його задання кодом C#, це надало можливість розділити графічний інтерфейс від логіки додатку, а також описувати його у більш зрозумілій формі, що легко підтримується та оновлюється.

У ході розробки мобільного додатку використовувався шаблон Model-View-ViewModel, що полягає у розділенні функціональної складової на три компоненти:

1. View. Представлення чи користувацький інтерфейс;
2. Model. Модель чи дані, які використовуються у додатку;
3. ViewModel. Проміжний шар між представленням та даними, який забезпечує їх взаємодію.

Таким чином можна зменшити зв’язність між компонентами та розділити відповідальність між ними [11].

Приклад задання таким чином однієї із сторінок, а саме сторінки надання локації та додаткової інформації, розміщений у додатку В.4. У якості моделей використовуються доменні сутності: район, адреса та локація. Логіка у свою

чергу зосереджена у класі LocationProvidingPageViewModel.cs, що відповідає за завантаження даних з прошарку сервісів (у вигляді доменних моделей) та визначає властивості, що буде використовувати представлення у вигляді LocationProvidingPage.xaml. Разом із цим варто зазначити, що ViewModel реалізує інтерфейс INotifyPropertyChanged, що дозволяє сповіщувати систему про зміни його властивостей. У даному випадку при виборі локаційної сутності більш високого рівня, завантажувати відповідні сутності нижчих ієрархічних рівнів.

# РОЗДІЛ 4
# ПРОГРАМНА СИСТЕМА «PARKING CONTROL»

Наразі комплексна програмна система «Parking Control» має три реалізовані підсистеми, що описані у функціональних вимогах: адміністратора, обробника звітів та громадянина. Перші дві – на базі веб-додатку, а функціональне навантаження третьої зосереджено у мобільному додатку.

Інтерфейс має інтуїтивно зрозумілий функціонал і, таким чином, додатки не потребують додаткової користувацької кваліфікації. У той же час у поточній версії єдина мова локалізації – англійська. Тому для впевненого користування довільної із підсистем існує необхідність у базовому володінні нею.

## 4.1 Загальний функціонал підсистем веб-додатку

Для входу у підсистему адміністратора необхідно пройти авторизацію. Для неавторизованих користувачів у верхній частині додатку відображена кнопка «Log in». Після натискання якої відбувається перехід до сторінки авторизації (див. рис. Д.1.1). У випадку ж авторизованого користувача, відповідно відображаються кнопки налаштувань та виходу з поточного профілю.

При переході до налаштувань, відкривається сторінка адміністрування аккаунту, де користувач має можливість змінити пароль, пошту чи додати двофакторну автентифікацію (див. рис. Д.1.2). Усі форми мають валідацію а коректність введених даних та сповіщають користувача при помилковому вводі.

Швидкий та зручний перехід між функціональними блоками досягається завдяки статичному боковому меню, що містить опції відповідно до ролі поточного користувача. Разом із цим він є адаптивним, що дозволяє зручно користуватися рішенням навіть з браузерів мобільних пристроїв (див. рис. Д.1.3).

## 4.2 Підсистема адміністратора

Для адміністратора у наявності три функціональні вкладки: перегляд статистики, адміністрування обробників та адміністрування евакуаційних екіпажів.

На вкладці статистики виведені основні показники щодо звітів за вказаний користувачем період (див. рис. Д.2.1). Візуалізація містить:

- лінійний графік кількості отриманих та завершених звітів за місяцями;
- кругову діаграму, що демонструє пропорційне співвідношення підтверджених та відхилених звітів;
- гістограму кількості звітів за районами.

Вкладка адміністрування обробників звітів приводить їх список у вигляді таблиці, стовпчиками якої є ім’я обробника, його статус, кількість активних звітів та райони до яких він належить, а також кнопку для додавання нового обробника (див. рис. Д.2.2). Таблиця надає можливість сортування та фільтрації за всіма колонками. За вибором бажаного запису відбувається перехід до інформації щодо конкретного обробника.

При переході до конкретного обробника, адміністратор може переглядати основні дані, а також бачити особисту статистику обробника у двох кругових діаграмах (див. рис. Д.2.3). Перша демонструє відношення оброблених звітів до тих, що ще потребують опрацювання, друга – співвідношення підтверджених та відхилених звітів. Разом із цим є можливість змінити статус та райони, які має опрацьовувати поточний обробник.

Можливості адміністрування евакуаційних екіпажів в цілому аналогічні, що стосується можливостей перегляду, додавання та редагування. Але у випадку з екіпажами налаштуванню підлягає також склад екіпажу та евакуаційні автомобілі, що є у розпорядженні.

### 4.3 Підсистема обробника звітів

Перш за все обробнику надається можливість адміністрування власного профілю та перегляд статистики (див. рис. Д.3.1). В цілому інтерфейс подібний до того, що використовує адміністратор для перегляду інформації щодо обробника, окрім того, що тут відсутня можливість редагування статусу та районів. Але разом з тим наявний функціонал для зміни чи видалення фотографії профілю.

На вкладці звітів у свою чергу наявний їх список з інформацією про дату, статус та локацію (див. рис. Д.3.2). Наявна аналогічна можливість сортування, а також фільтрації за всіма властивостями, у тому числі з широкими можливостями щодо налаштування діапазону дат (див. рис. Д.3.3).

При виборі конкретного звіту із списку користувач переходить безпосередньо до його обробки (див. рис. Д.3.4, Д.3.5). Інформація про звіт виводиться у трьох структурних блоках:

- основні деталі: дата звітування, локація, статус та додаткова інформація;
- інформація про автомобіль та власника: реєстраційний номер, модель, тип автомобіля, його розміри, вага, а також контактні дані власника та його ідентифікаційний номер;
- панорамні фото правопорушення.

Далі, якщо звіт ще не був попередньо обробленим, існує дві опції: підтвердити чи відхилити (див. рис. Д.3.6). У випадку підтвердження обробник має можливість вибрати екіпаж на який буде назначена евакуація. Список містить у собі евакуаційні екіпажі, що підходять за районами, є доступними та мають відповідні автомобілі у розпорядженні. Якщо ж за тих чи інших причин обробник приймає рішення відмовити у евакуації, то він повинен вказати причину відмови у відведеному полі.

Якщо ж звіт був уже переведений у якийсь з описаний вище статусів, то обробник має можливість назначити звіт на інший екіпаж, скасувати у випадку

попередньо підтвердженого звіту чи навпаки – підтвердити попередньо скасований.

### 4.3 Підсистема громадянина

Громадянин у свою чергу через додаток на мобільний отримує можливість надати усю необхідну нормативну інформацію щодо правопорушення. Для цього процес розділений на три інтерфейсні етапи: надання фото панорами правопорушення, фото номерного знаку та дані локації з додатковою інформацією.

Сторінки надання фото панорами та номера по суті аналогічні (див. рис. Д.4.1). Щодо способу надання фото користувач має дві опції (див. рис. Д.4.2):

- Зробити нову фотографію. При цьому відкривається камера мобільного телефону та користувач моє можливість зробити бажане фото. Далі він може або підтвердити зроблену фотографію, або спробувати ще раз;
- Вибрати серед існуючих фотографій на його пристрої. При виборі цієї опції користувач можете через провідник вибрати фото із галереї або бажаної папки.

У свою чергу сторінка надання локації дозволяє користувачу вибрати почергово район, вулицю та її номер (див. рис. Д.4.3). Дані ієрархічно підтягуються виходячи із того, який варіант був вибраний рівнем вище. У текстовому полі додаткової інформації користувач може вказати дані, що він вважає корисними (уточнені орієнтири, специфіка автомобіля тощо).

Далі громадянин може або підтвердити введену інформацію та передати звіт у обробку, або повернутися до якогось із попередніх кроків.

# ВИСНОВКИ

У ході виконання роботи “Методи нейромережевого розпізнавання об’єктів в системі громадського контролю законності паркування” було виконано наступні завдання:

1. Проаналізовано існуючи засоби звітування та реагування щодо актів незаконного паркування, формалізовано відповідні бізнес-процеси;
2. Досліджено підходи до локалізації та розпізнавання номерних знаків. Запропоновано алгоритм на основі методів визначення границь, з подальшою символьною сегментацією та нейромережевим розпізнаванням;
3. Вивчено технології для розробки комплексних програмних систем на базі хмарних технологій: Blazor, Xamarin Forms, Azure Durable Functions, Azure Cognitive Services;
4. Спроектовано та розроблено програмну систему автоматизації надання та обробки звітів щодо незаконного паркування.

Отже, розроблена комплексна програмна система надає можливість громадянам у зручній та ефективній формі надавати усю необхідну нормативну інформацію щодо актів незаконного паркування, а евакуаційним службам, у свою чергу, обробляти та відповідним чином реагувати на звернення громадян.

Рішення уже зараз може впроваджуватися для тестування у містах задля підвищення ефективності роботи служб та зменшення кількості випадків незаконного паркування без відповідної реакції. Разом із цим, розроблена програмна база може бути адаптована до широкого спектру предметних областей, процеси яких потребують експертного контролю.

У ході подальшої розробки планується розширення функціоналу, впровадження підсистеми евакуаційного екіпажу та дослідження можливості інтеграції системи дисциплінарних покарань.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Gonzalez R. Digital image processing / R. Gonzalez, R. Woods., 2012. – 701 с.
2. Pattern recognition letter. // Elsevier. – 2005. – №15. – С. 2431–2438.
3. Гафаров Ф. М. Штучні нейронні мережі та їх використання [Електронний ресурс] / Ф. М. Гафаров, А. Ф. Галимянов // Видавництво Казанського університету. – 2018. – Режим доступу до ресурсу: https://kpfu.ru/staff_files/F1493580427/NejronGafGal.pdf.
4. Навойчик А. В. Розпізнавання текстових зображень з використанням нейронних мереж [Електронний ресурс] / А. В. Навойчик, Н. И. Гурин – Режим доступу до ресурсу: https://rep.bntu.by/bitstream/handle/data/27276/Raspoznavanie_tekstovyh_izobrazhenij_pri_pomoshchi_nejronnyh_setej.pdf?sequence=1&isAllowed=y.
5. Синєглазов В. Глибокі нейронні мережі для вирішення завдань розпізнавання і класифікації зображення [Електронний ресурс] / В. Синєглазов, О. Чумаченко – Режим доступу до ресурсу: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf.
6. Nijhuis J. A. Car license plate recognition with neural networks [Електронний ресурс] / J. A. Nijhuis, M. H. Ter Brugge // IEEE – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/487708.
7. ASP.NET Core Blazor documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0.
8. Azure Cognitive Services documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/cognitive-services/.
9. Azure Durable Functions documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/azure-functions/durable/.

10. Xamarin.Forms documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/.
11. The Model-View-ViewModel Pattern [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

# ДОДАТОК А – ДІАГРАМИ ПРЕЦЕДЕНТІВ КОРИСТУВАЦЬКИХ ПІДСИСТЕМ

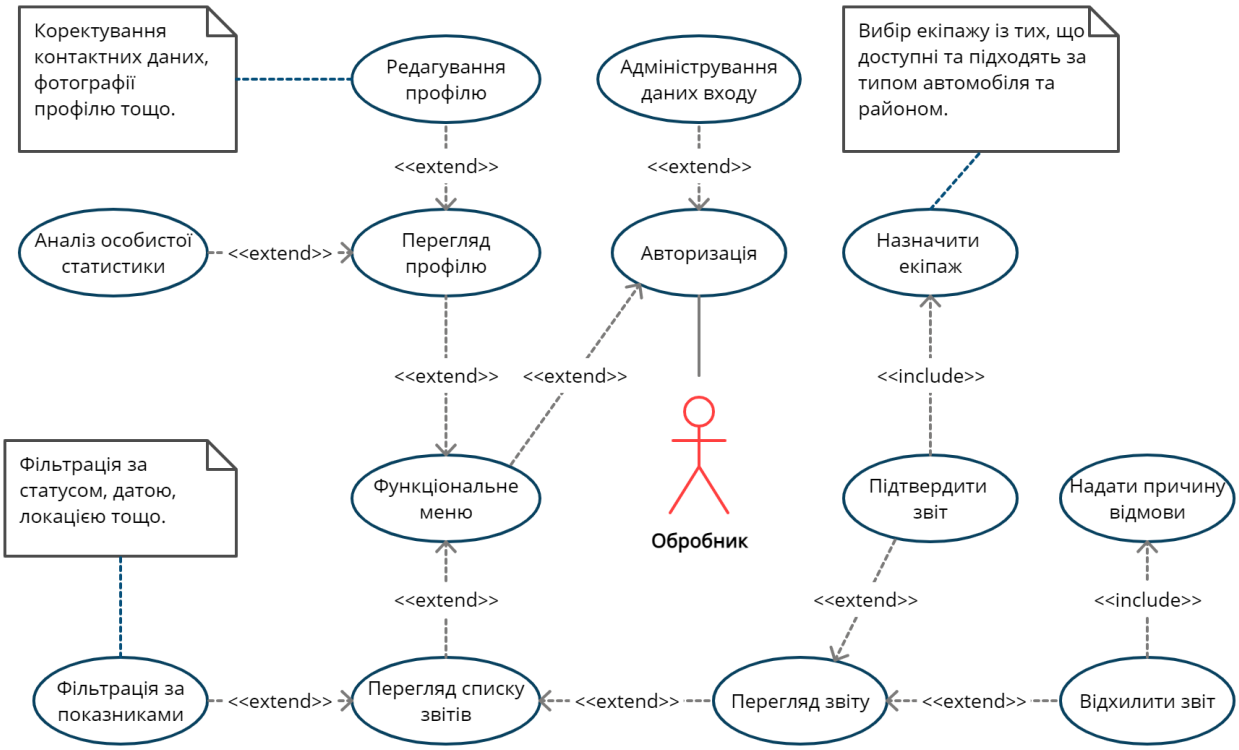



Рисунок А.1. Діаграма прецедентів підсистеми обробника

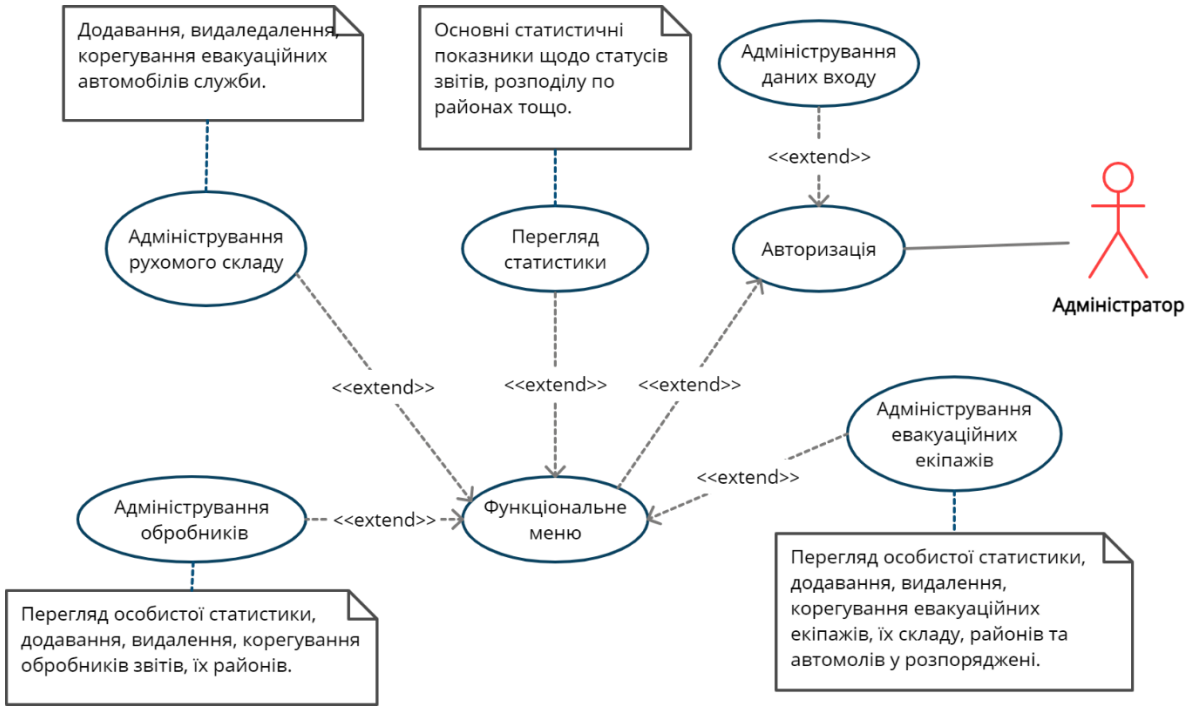



Рисунок А.2. Спрощена діаграма прецедентів адміністратора

# ДОДАТОК Б – ДІАГРАМА БАЗИ ДАНИХ СИСТЕМИ

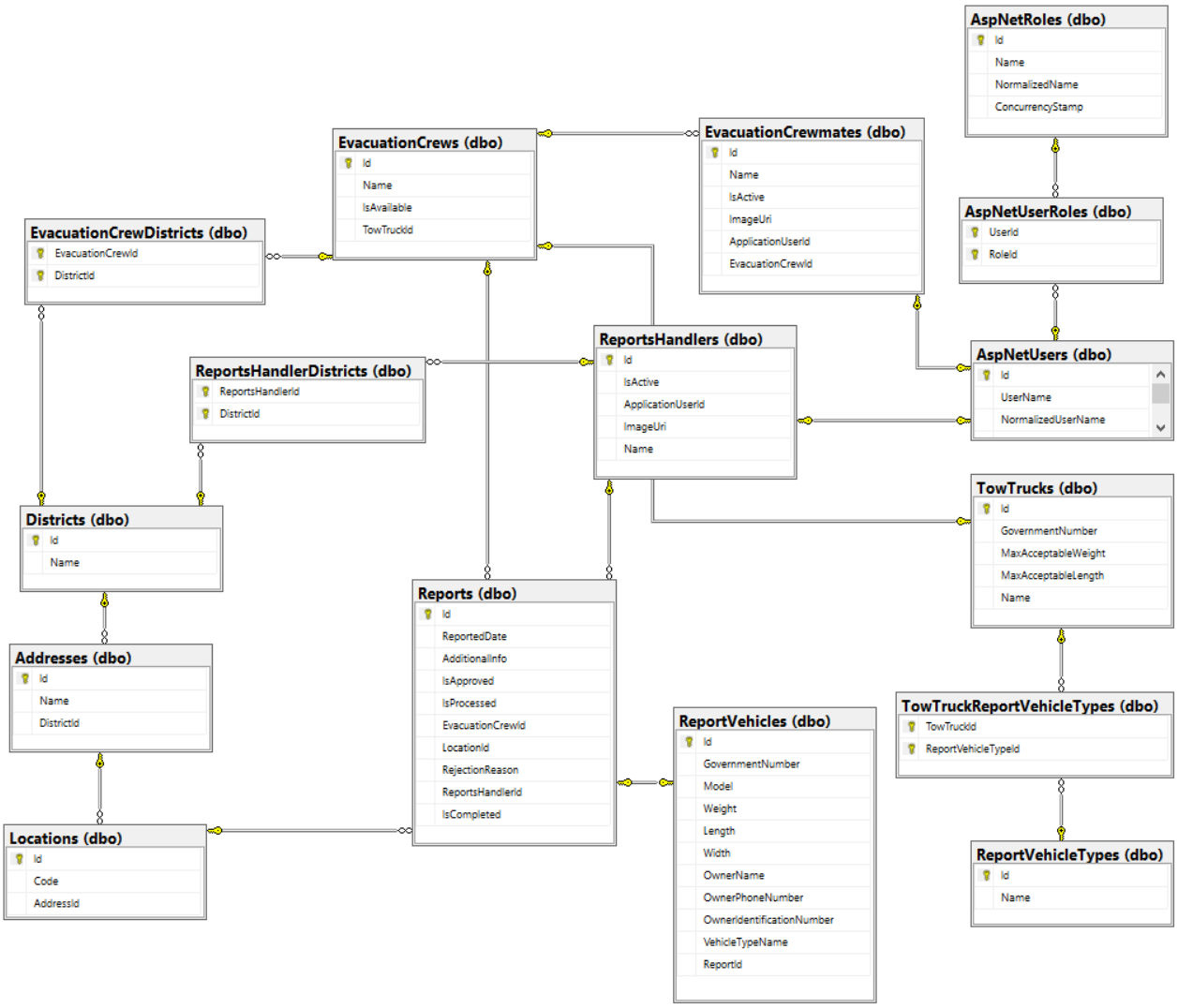



Рисунок Б.1. Діаграма бази даних системи

# ДОДАТОК В – ФРАГМЕНТИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

## В.1 Приклад задання інтерфейсу та логіки компонента

```
@page "/reporthandlers"

@inject IReportsHandlerService reportHandlersService
@inject ILocationService locationService
@inject NavigationManager navigationManager
@inject DialogService dialogService

@attribute [Authorize(Roles = RolesConstants.Administrator)]

@if (reportHandlers != null)
{
    <RadzenCard>
        <div style="margin-bottom: 20px;">
            <RadzenButton Icon="add" Text="Create handler"
                          Click="@(args => dialogService.Open<ReportsHandlerAddingDialog>("Handler creating"))" />
        </div>
        <RadzenGrid Data="@reportsHandlerResponses" AllowFiltering="true" AllowPaging="true" AllowSorting="true"
                    TItem="ReportsHandlerResponse" AllowColumnResize="true" RowClick="(args) => OpenReportsHandler(args.Id)">
            <Columns>
                <RadzenGridColumn TItem="ReportsHandlerResponse" Property="Name" Title="Name" Width="200px" />
                <RadzenGridColumn TItem="ReportsHandlerResponse" Property="IsActive" Title="Is active" Width="100px">
                    <FilterTemplate>
                        <RadzenDropDown @bind-Value="currentActiveState" TextProperty="Text" ValueProperty="Value" Style="width:100%"
                                        Change=@OnFilterChange
                                        Data="@(Enum.GetValues(typeof(BooleanState)).Cast<BooleanState>().Select(t => new { Text = $"{t}", Value = t }))" />
                    </FilterTemplate>
                </RadzenGridColumn>
                <RadzenGridColumn TItem="ReportsHandlerResponse" Property="ActiveReports" Title="Active reports" Width="150px" />
                <RadzenGridColumn TItem="ReportsHandlerResponse" Property="Districts" Title="Districts" Bubble="true">
                    <FilterTemplate>
                        <RadzenDropDown @bind-Value="currentDistrictIds" Multiple="true" Data="districts" TextProperty="Name" ValueProperty="Id"
                                        Change="OnFilterChange" />
                    </FilterTemplate>
                </RadzenGridColumn>
            </Columns>
        </RadzenGrid>
```

Рисунок В.1.1. Створення інтерфейсу с Razor розміткою

```
@code {
    private IEnumerable<ReportsHandler> reportHandlers;
    private IEnumerable<ReportsHandler> filteredReportHandlers;
    private IEnumerable<ReportsHandlerResponse> reportsHandlerResponses;

    private IEnumerable<District> districts;
    private IEnumerable<Guid> currentDistrictIds;

    private BooleanState currentActiveState = BooleanState.All;

    public enum BooleanState ...

    protected override async Task OnInitializedAsync()
    {
        reportHandlers = await reportHandlersService.GetReportsHandlersAsync();
        districts = await locationService.GetDistrictsAsync();

        currentDistrictIds = districts.Select(d => d.Id);
        reportsHandlerResponses = ConvertGetResponse(reportHandlers);

        dialogService.OnClose += DialogCloseAsync;
    }

    private void OnFilterChange()
    {
        filteredReportHandlers = reportHandlers;

        if (currentActiveState != BooleanState.All)
            filteredReportHandlers = filteredReportHandlers.Where(rh => rh.IsActive == Convert.ToBoolean(currentActiveState));

        filteredReportHandlers = filteredReportHandlers
            .Where(rh => rh.ReportsHandlerDistricts.Select(rh => rh.DistrictId).Intersect(currentDistrictIds).Any());

        reportsHandlerResponses = ConvertGetResponse(filteredReportHandlers);
    }

    private void OpenReportsHandler(Guid reportsHandlerId)
    {
        navigationManager.NavigateTo("/reportshandler" + "/" + reportsHandlerId);
    }
```

Рисунок В.1.2. Задання логіки компоненту у фрагменті @code

## В.2 Реалізація класу розпізнавання з Azure Cognitive

```
private async Task<TextOperationResult> RecognizeAsync<T>(Func<ComputerVisionClient, Task<T>> GetHeadersAsyncFunc,
    Func<T, string> GetOperationUrlFunc) where T : new()
{
    using var client = new ComputerVisionClient(clientCredentials) { Endpoint = subscriptionEndpoint };

    T recognizeHeaders = await GetHeadersAsyncFunc(client);
    string operationUrl = GetOperationUrlFunc(recognizeHeaders);
    string operationId = operationUrl.Substring(operationUrl.LastIndexOf('/') + 1);

    TextOperationResult result = await client.GetTextOperationResultAsync(operationId);
    for (int attempt = 1; attempt <= maxRetryTimes; attempt++)
    {
        if (result.Status == TextOperationStatusCodes.Failed || result.Status == TextOperationStatusCodes.Succeeded)
            break;

        await Task.Delay(queryWaitTimeInSecond);
        result = await client.GetTextOperationResultAsync(operationId);
    }

    return result;
}
```

Рисунок В.2.1. Універсальна функція для методів ComputerVisionClient

```
public async Task<string> RecognizeUrlAsync(string imageUrl)
{
    var textOperationResult = await RecognizeAsync(
        async (ComputerVisionClient client) => await client.RecognizeTextAsync(imageUrl, TextRecognitionMode.Printed),
        headers => headers.OperationLocation);

    var stringResult = ProcessTextOperationResult(textOperationResult);
    return stringResult;
}
```

Рисунок В.2.2. Метод розпізнавання символів на зображенні

```
private string ProcessTextOperationResult(TextOperationResult textOperationResult)
{
    StringBuilder stringBuilder = new StringBuilder();

    if (textOperationResult != null && textOperationResult.RecognitionResult != null &&
        textOperationResult.RecognitionResult.Lines != null && textOperationResult.RecognitionResult.Lines.Count > 0)
    {
        foreach (var line in textOperationResult.RecognitionResult.Lines)
        {
            foreach (var word in line.Words)
            {
                stringBuilder.Append(word.Text);
            }
        }
    }

    return stringBuilder.ToString();
}
```

Рисунок В.2.3. Метод обробки результату розпізнавання

## В.3 Реалізація класу розпізнавання з Azure Cognitive

```
[FunctionName("PlateRecognitionOrchestrator_HttpTriggerStart")]
0 references
public async Task<HttpResponseMessage> HttpStart(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post")] HttpRequestMessage request,
    [DurableClient] IDurableOrchestrationClient starter,
    ILogger log)
{
    var requestContent = await request.Content.ReadAsStringAsync();
    string instanceId = await starter
        .StartNewAsync("PlateRecognitionOrchestrator", input: requestContent);

    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
    return starter.CreateCheckStatusResponse(request, instanceId);
}
```

Рисунок В.3.1. Функція клієнту оркестратора

```
[FunctionName("PlateRecognitionOrchestrator")]
0 references
public async Task RunOrchestrator(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var requestContent = context.GetInput<string>();
    var plateBlobDetails = JsonConvert.DeserializeObject<LicensePlateBlobDetails>(requestContent);

    var recognizedPlate = await context
        .CallActivityAsync<string>(FunctionConstants.RecognizePlateByPhoto, plateBlobDetails.Name);

    var vehicle = await context
        .CallActivityAsync<Vehicle>(FunctionConstants.GetVehicleDataByPlate, recognizedPlate);

    var reportVehicle = mapper.Map<ReportVehicle>(vehicle);
    reportVehicle.ReportId = plateBlobDetails.Name;

    await context.CallActivityAsync(FunctionConstants.UploadVehicleDataToStorage, reportVehicle);
}
```

Рисунок В.3.2. Функція оркестратора

```
[FunctionName(FunctionConstants.RecognizePlateByPhoto)]
0 references
public async Task<string> RecognizePlateByPhoto([ActivityTrigger] Guid plateBlobName)
{
    var plateBlobUrl = blobStorageService.GetPlateBlobUrlByName(plateBlobName.ToString());

    var plateNumber = await plateRecognitionService.RecognizeUrlAsync(plateBlobUrl);
    return plateNumber;
}
```

Рисунок В.3.3. Одна із Activity функцій

## В.4 Реалізація MVVM шаблону з Xamarin

```
public class LocationProvidingPageViewModel : ViewModelBase
{
    1 reference
    public IEnumerable<District> Districts { get; set; }

    0 references
    public string AdditionalInfo { get; set; }

    District selectedDistrict;
    0 references
    public District SelectedDistrict
    {
        get { return selectedDistrict; }
        set
        {
            if (selectedDistrict != value)
            {
                selectedDistrict = value;
                OnPropertyChanged();
            }
        }
    }
```

Рисунок В.4.1. Фрагмент задання ViewModel

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    3 references
    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Рисунок В.4.2. Реалізація INotifyPropertyChanged інтерфейсу

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:viewmodels="clr-namespace:ParkingControl.Mobile.ViewModels"
             x:DataType="viewmodels:LocationProvidingPageViewModel"
             x:Class="ParkingControl.Mobile.Views.Reporting.LocationProvidingPage">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
                <StackLayout VerticalOptions="Start">
                    <Frame BackgroundColor="Orange" Padding="24">
                        <Label Text="incident address" HorizontalTextAlignment="Center"
                               TextColor="White" FontSize="24"/>
                    </Frame>
                    <Label Text="District:"/>
                    <Picker Title="Select district" ItemsSource="{Binding Districts}" ItemDisplayBinding="{Binding Name}"
                            SelectedItem="{Binding SelectedDistrict}" />
                    <Label Text="Address:"/>
                    <Picker Title="Select district" ItemsSource="{Binding SelectedDistrict.Addresses}" ItemDisplayBinding="{Binding Name}"
                            SelectedItem="{Binding SelectedAddress}" />
```

Рисунок В.4.3. Фрагмент задання інтерфейсу з XAML

```
[XamlCompilation(XamlCompilationOptions.Compile)]
5 references
public partial class LocationProvidingPage : ContentPage
{
    private readonly ReportingRequest reportingRequest;

    1 reference
    public LocationProvidingPage(ReportingRequest reportingRequest)
    {
        this.reportingRequest = reportingRequest;
        BindingContext = new LocationProvidingPageViewModel();

        InitializeComponent();
    }

    0 references
    protected async override void OnAppearing()
    {
        var vm = BindingContext as LocationProvidingPageViewModel;
        vm.Districts = await App.LocationsManagerInstance.GetDistrictsAsync();

        base.OnAppearing();
    }
```

Рисунок В.4.4. Підв’язка ViewModel під клас представлення

# ДОДАТОК Д – ІНТЕРФЕЙС РОЗРОБЛЕНОЇ СИСТЕМИ

## Д.1 Загальні елементи інтерфейсу веб-додатку

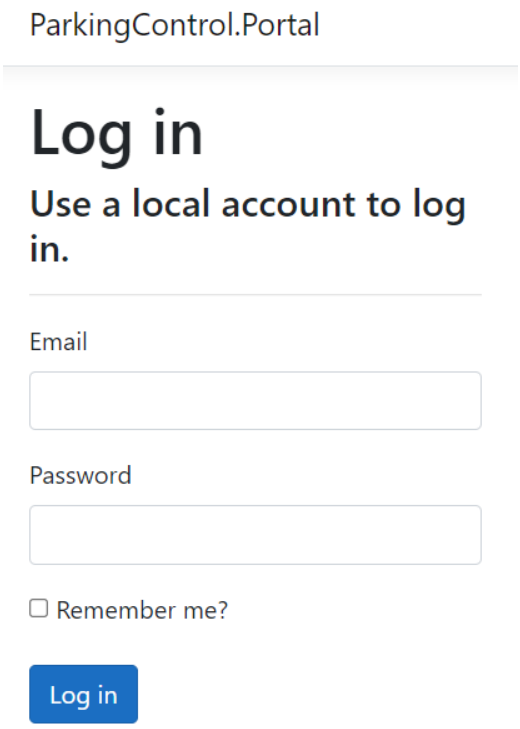



Рисунок Д.1.1. Сторінка авторизації користувача

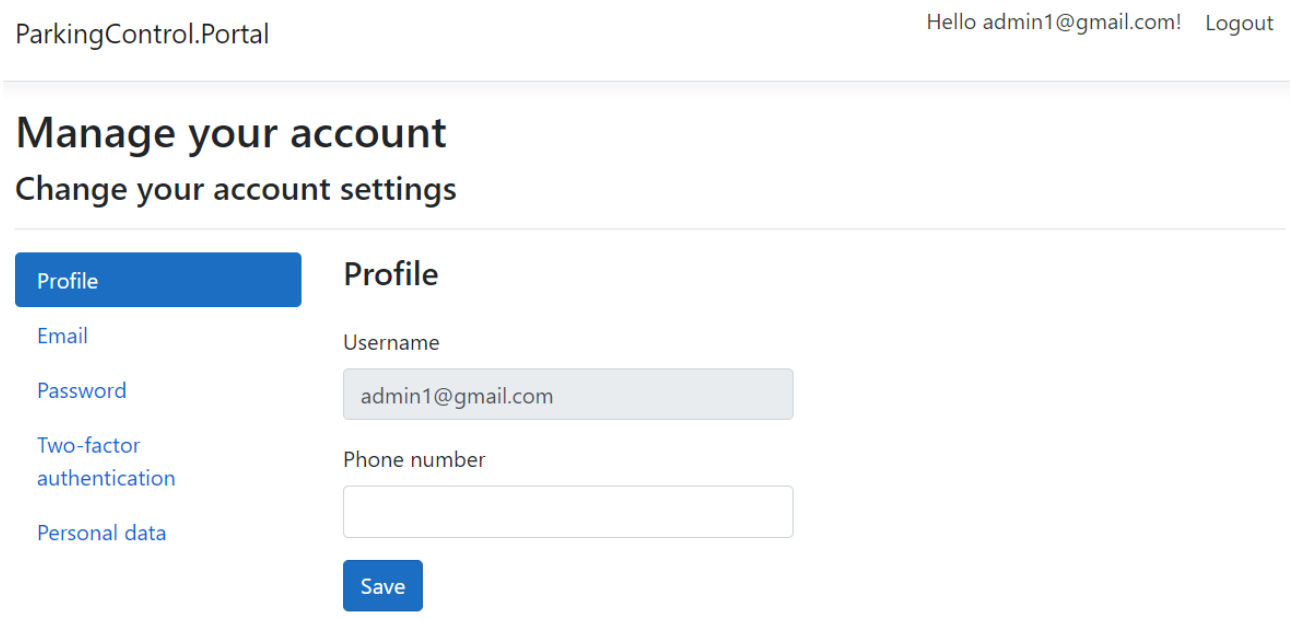



Рисунок Д.1.2. Сторінка адміністрування аккаунту

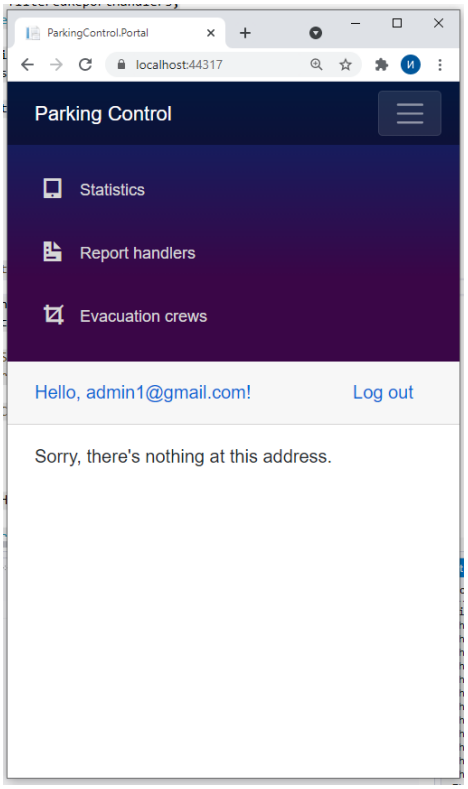



Рисунок Д.1.3. Адаптивне навігаційне меню

## Д.2 Підсистема адміністратора

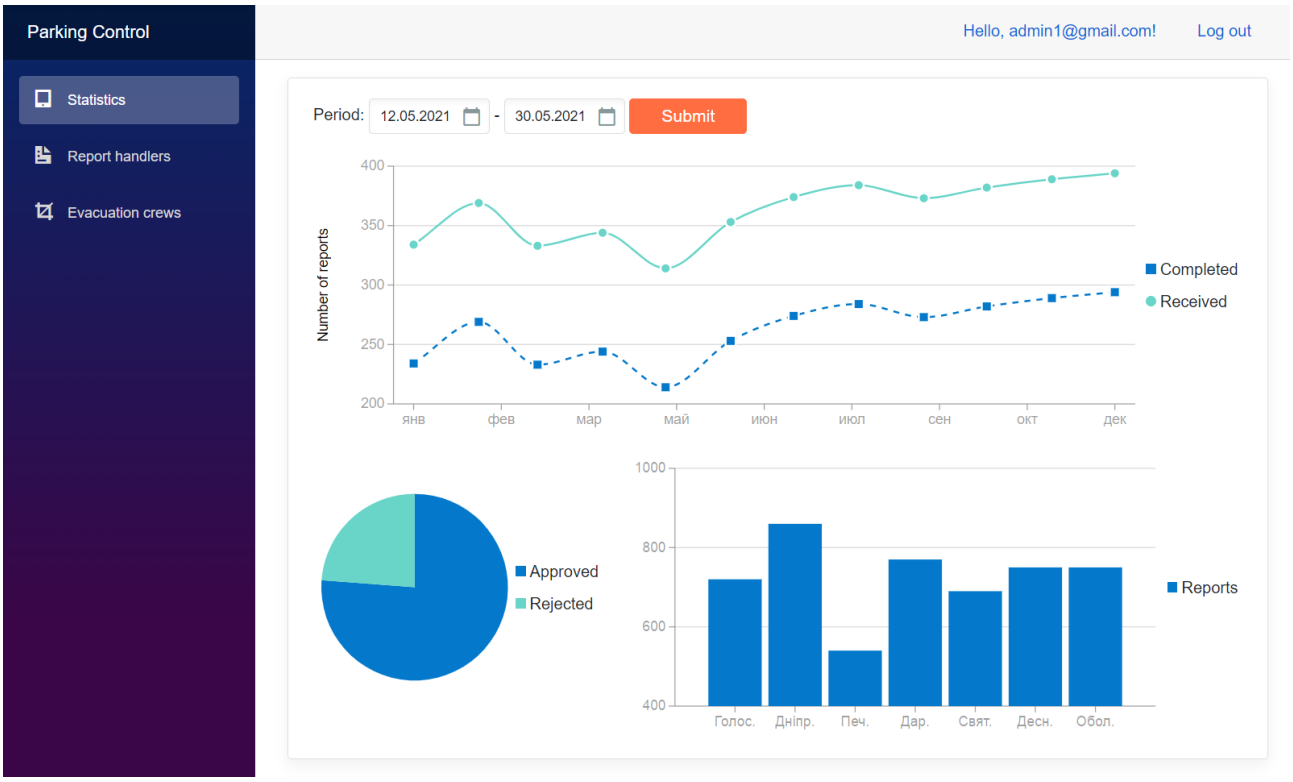



Рисунок Д.2.1. Сторінка перегляду статистики

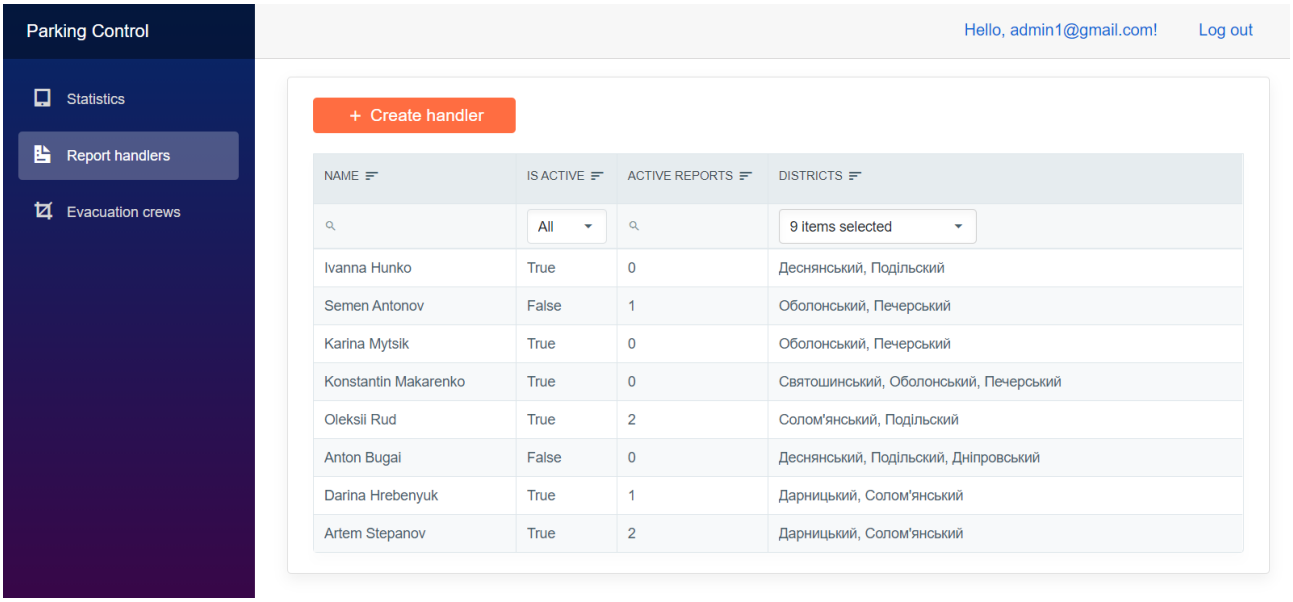



Рисунок Д.2.2. Сторінка перегляду обробників системи

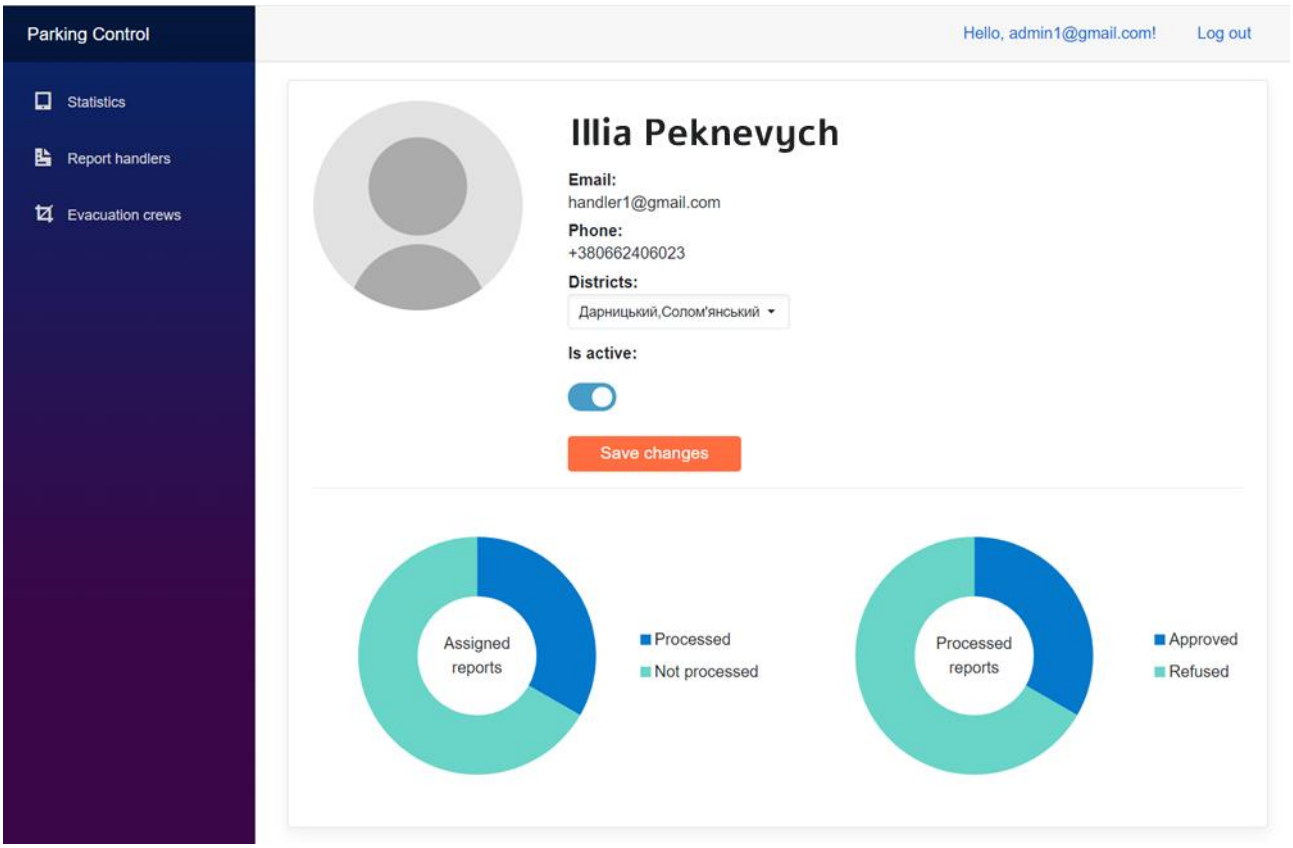



Рисунок Д.2.3. Сторінка перегляду інформації про конкретного обробника

## Д.3 Підсистема обробника звітів

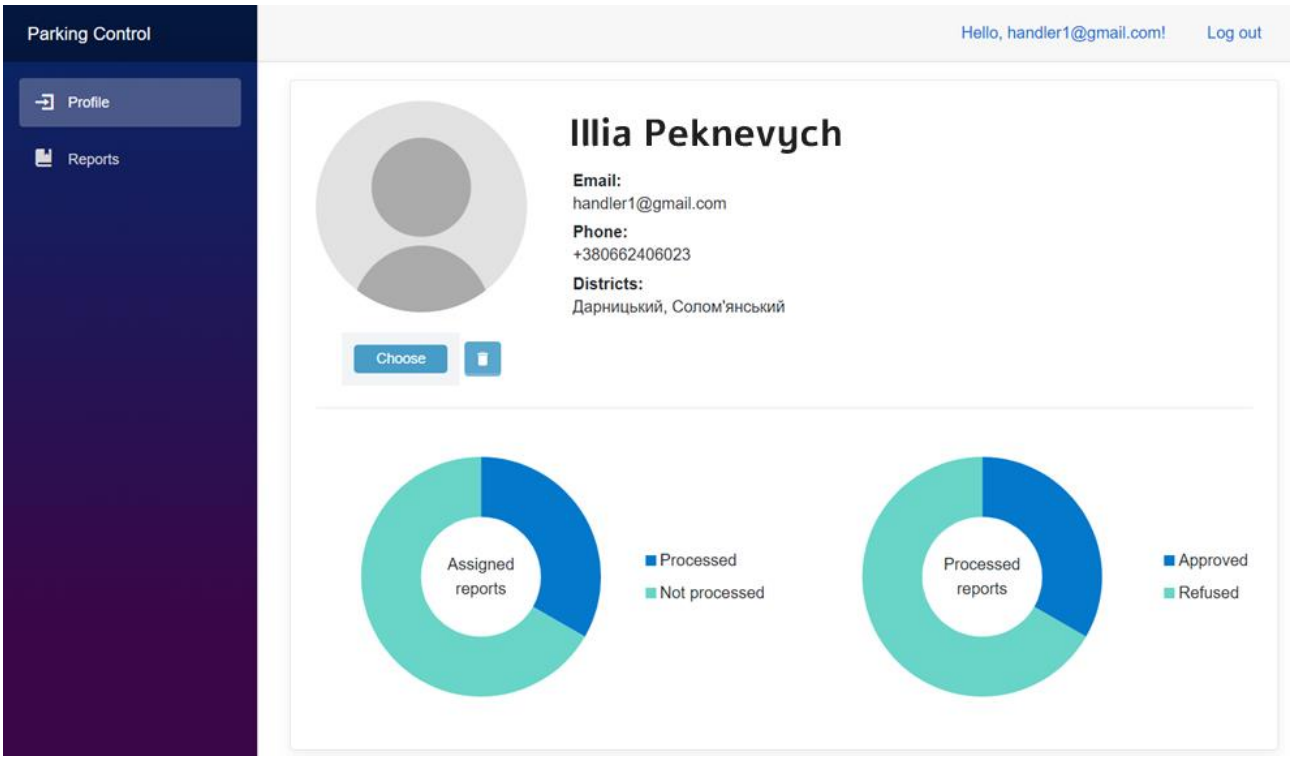



Рисунок Д.3.1. Сторінка перегляду особистої інформації

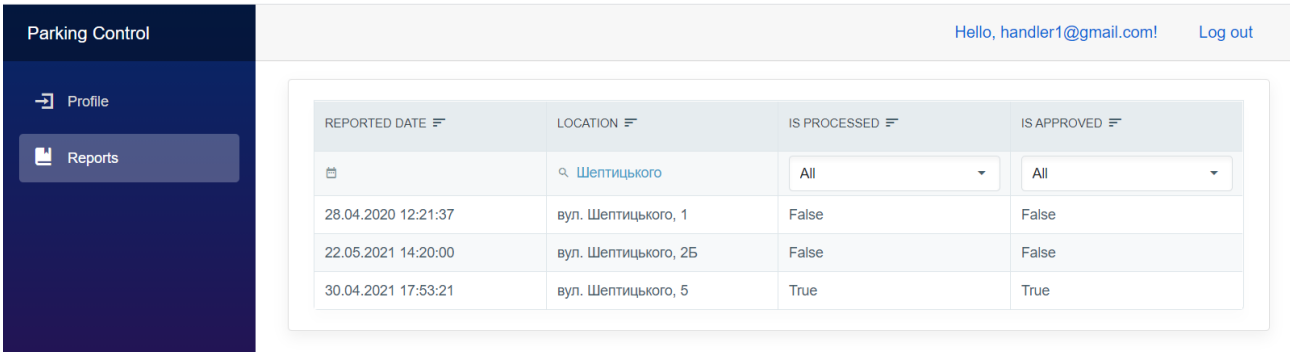



Рисунок Д.3.2. Сторінка перегляду списку звітів

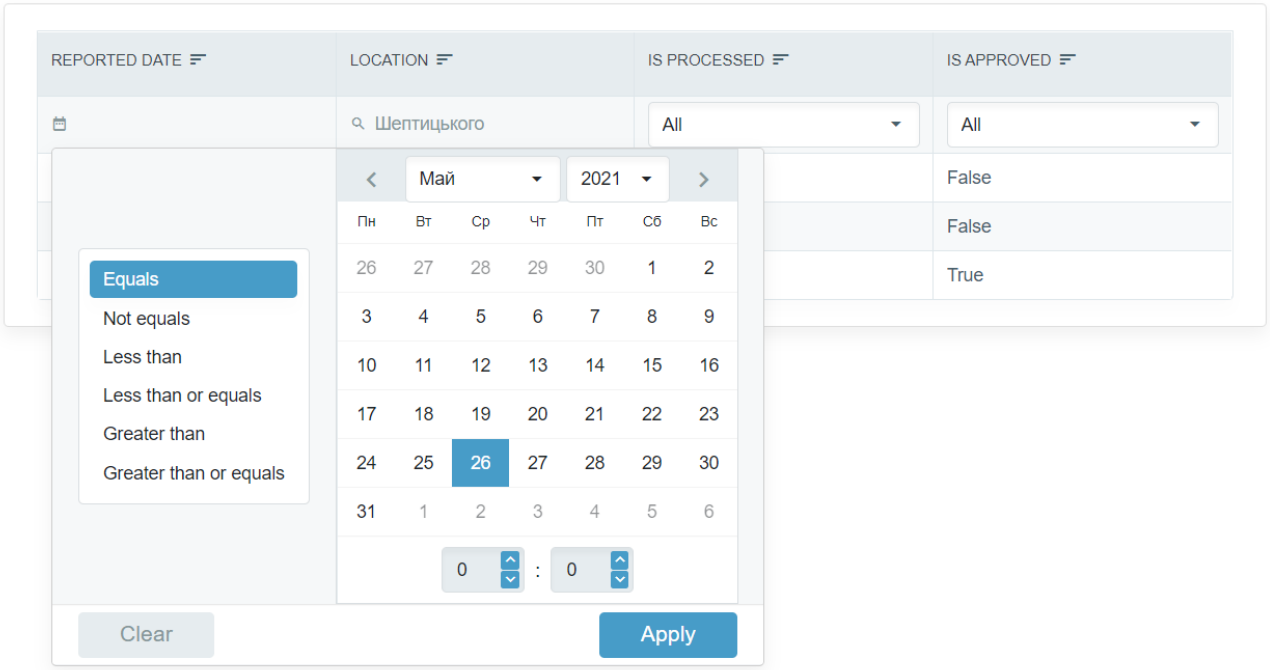



Рисунок Д.3.3. Функціонал налаштування дати

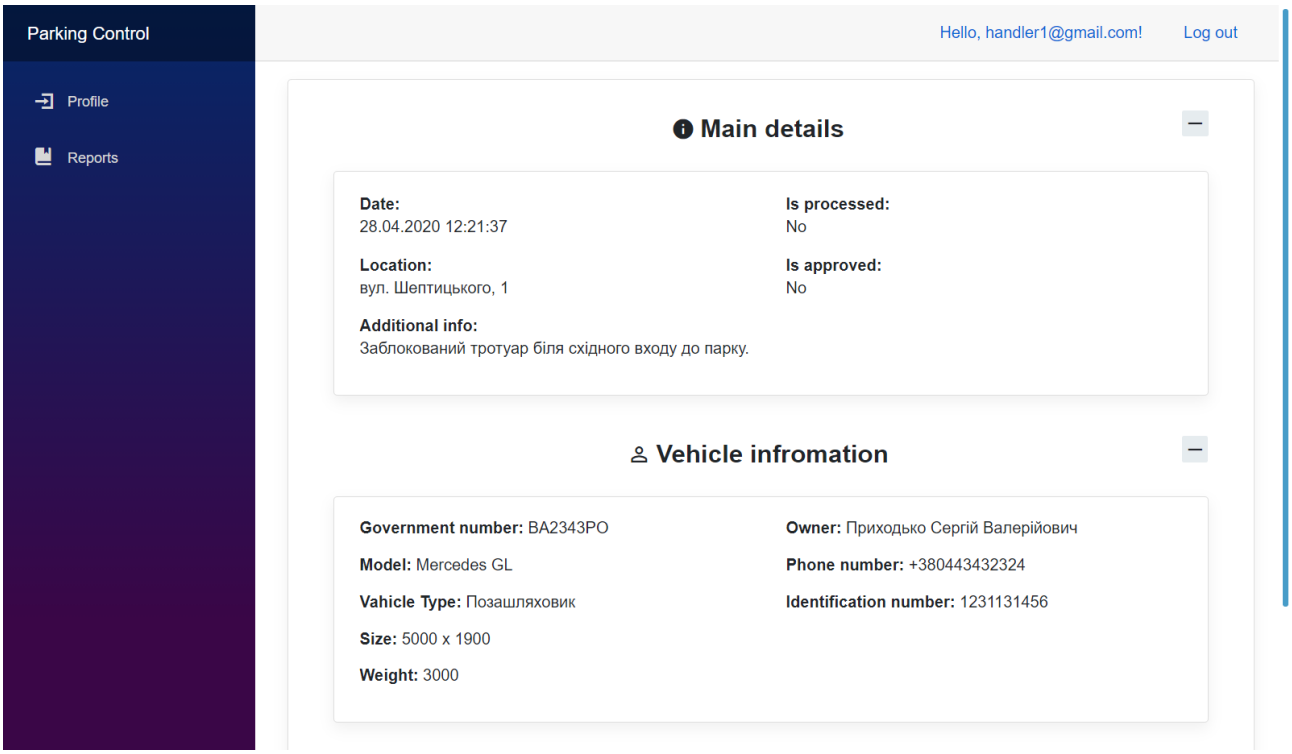



Рисунок Д.3.4. Основні деталі щодо звіту

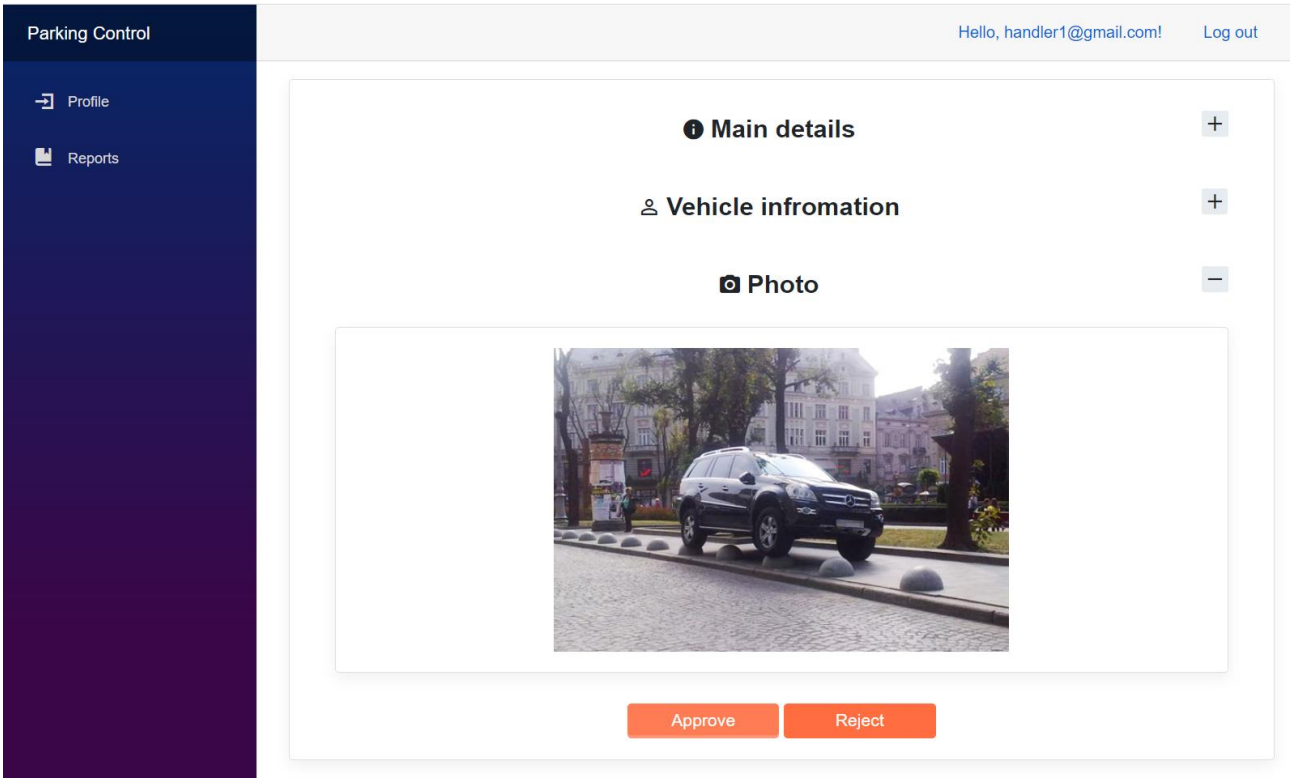



Рисунок Д.3.5. Перегляд панорами правопорушення

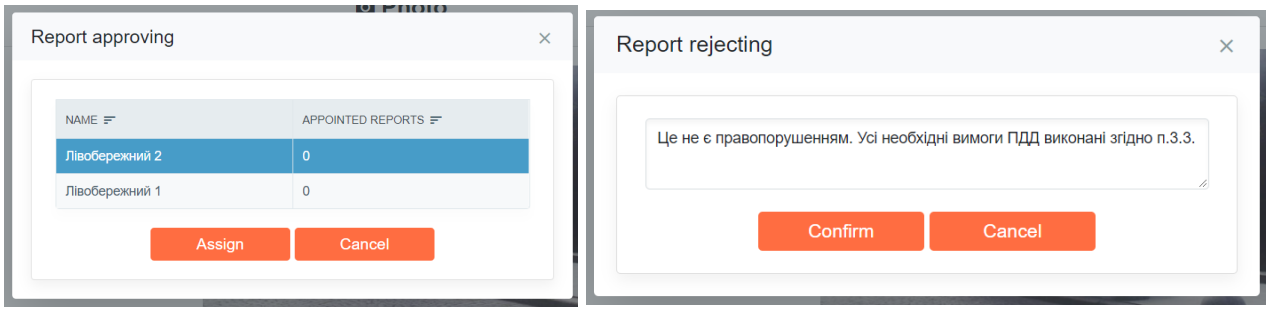



Рисунок Д.3.6. Діалогові вікна реакції на звіт

## Д.4 Підсистема громадянина

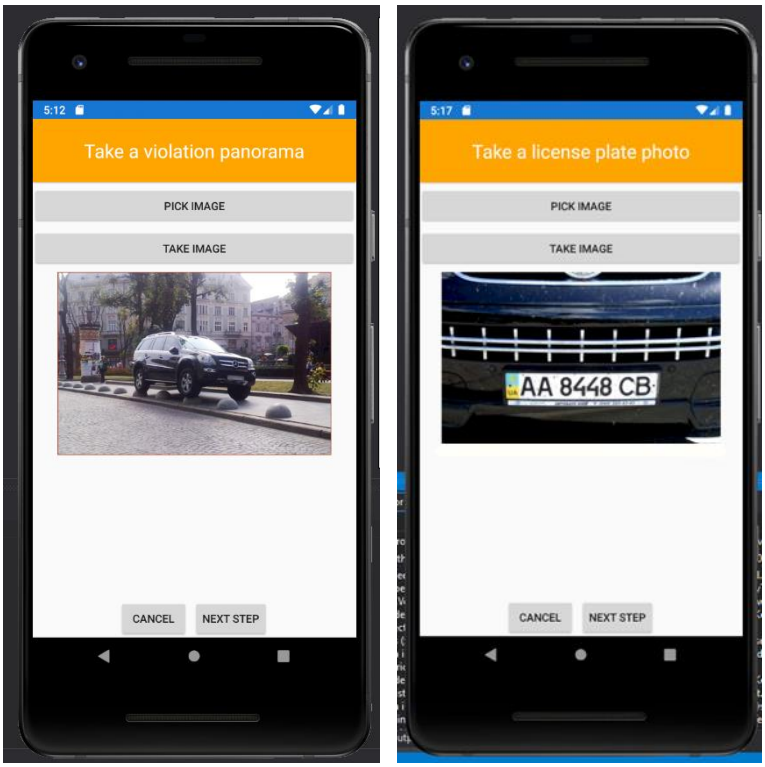



Рисунок Д.4.1. Функціонал надання фотографій до звіту

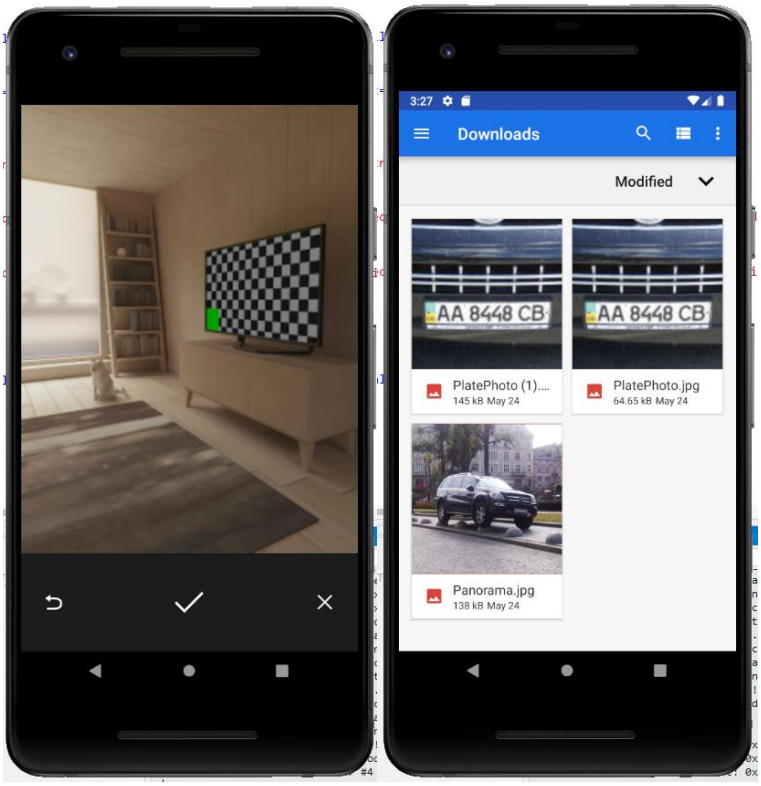



Рисунок Д.4.2. Опції вибору фотографій

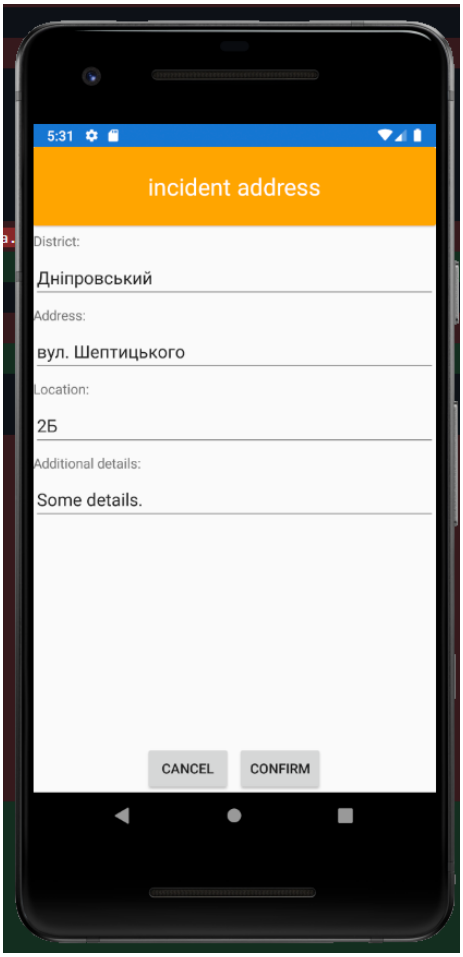



Рисунок Д.4.3. Функціонал надання локації та додаткової інформації

## ДОДАТОК Е – SOFTWARE ARCHITECTURE DOCUMENT

***Taras Shevchenko National University of Kyiv***

# Public Parking Control
# Software Architecture Document (SAD)

**CONTENT OWNER: Illia Peknevych**

| DOCUMENT NUMBER: | RELEASE/REVISION: | RELEASE/REVISION DATE: |
|---|---|---|
| 1 | v.1 | 20 April, 2022 |

# Table of Contents


**1 Documentation Roadmap ..... 1**

**1.1 Purpose and Scope of the SAD ..... 1**

**1.2 How the SAD Is Organized ..... 2**

**1.3 Stakeholder Representation ..... 3**

**1.4 Viewpoint Definitions ..... 3**

1.4.1 Use-Case Viewpoint Definition ..... 4
1.4.1.1 Abstract ..... 4
1.4.1.2 Stakeholders and Their Concerns Addressed ..... 4
1.4.1.3 Elements, Relations, Properties, and Constraints . 4

1.4.2 Process Viewpoint Definition ..... 5
1.4.2.1 Abstract ..... 5
1.4.2.2 Stakeholders and Their Concerns Addressed ..... 5
1.4.2.3 Elements, Relations, Properties, and Constraints . 5

1.4.3 Logical Viewpoint Definition ..... 5
1.4.3.1 Abstract ..... 5
1.4.3.2 Stakeholders and Their Concerns Addressed ..... 5
1.4.3.3 Elements, Relations, Properties, and Constraints . 5

1.4.4 Deployment Viewpoint Definition ..... 5
1.4.4.1 Abstract ..... 5
1.4.4.2 Stakeholders and Their Concerns Addressed ..... 6
1.4.4.3 Elements, Relations, Properties, and Constraints . 6

1.4.5 Data Viewpoint Definition ..... 6
1.4.5.1 Abstract ..... 6
1.4.5.2 Stakeholders and Their Concerns Addressed ..... 6
1.4.5.3 Elements, Relations, Properties, and Constraints . 6

**1.5 How a View is Documented ..... 6**

**1.6 Relationship to Other SADs ..... 7**

**2 Architecture Background ..... 8**

**2.1 Problem Background ..... 8**

2.1.1 System Overview ..... 8

2.1.2 Goals and Context ..... 8

2.1.3 Significant Driving Requirements ..... 8

**2.2 Solution Background ..... 9**
2.2.1Architectural Approaches ..... 9
2.2.2Analysis Results ..... 10
2.2.3Requirements Coverage ..... 10

**2.3 Product Line Reuse Considerations ..... 11**

**3 Views ..... 12**

**3.1 Use Case View ..... 12**
3.1.1View Description ..... 12
3.1.2Primary presentation ..... 12
3.1.3Elements catalog ..... 12
3.1.4Architecture background ..... 13

**3.2 Process View ..... 13**
3.2.1View Description ..... 13
3.2.2Primary presentation ..... 13
3.2.3Elements catalog ..... 14
3.2.4Architecture background ..... 14

**3.3 Logical View ..... 14**
3.3.1View Description ..... 14
3.3.2Primary presentation ..... 15
3.3.3Elements catalog ..... 15
3.3.4Architecture background ..... 15

**3.4 Deployment View ..... 15**
3.4.1View Description ..... 15
3.4.2Primary presentation ..... 16
3.4.3Elements catalog ..... 16
3.4.4Architecture background ..... 17

**3.5 Data View ..... 17**
3.5.1View Description ..... 17
3.5.2Primary presentation ..... 17
3.5.3Elements catalog ..... 17
3.5.4Elements catalog ..... 18

**4 Referenced Materials ..... 19**

**5 Directory..... 20**

**5.1 Glossary..... 20**

**5.2 Acronym List ..... 21**

# Documentation Roadmap

## Purpose and Scope of the SAD

This SAD specifies the software architecture for Public Parking Control system**.** This document provides a comprehensive architectural overview of the system, using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

**What is software architecture?** The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD.

**Elements and relationships**. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

**Multiple structures.** The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So, it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing now—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other

[1] Here, a system may refer to a system of systems.

implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure; in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 0.

## How the SAD Is Organized

This SAD is organized into the following sections:

**Section 0 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.

**Section 2 ("Architecture Background") explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.

**Section 0 (Not** applicable.

**Views")** **specify the software architecture**. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.4) and is a representation of one or more structures present in the software.

**Sections 4 ("Referenced Materials") and 5 ("Directory") provide reference information for the reader.** Section 4 provides look-up information for documents that are cited

elsewhere in this SAD. Section 5 is a *directory*, which includes a glossary and acronym list.

## Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.4.

*Table 1: Stakeholders' concerns matrix*

| Concern / Stakeholder | UI | Programming tools | Performance | Architecture | Development costs |
|---|---|---|---|---|---|
| Customer | High | Low | High | Medium | High |
| End user | High | None | High | None | None |
| Developer | High | High | High | High | Medium |
| Tester | High | Medium | High | Medium | Medium |

## Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.1, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a

representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.4 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

*Table 2: Stakeholders and Relevant Viewpoints*

| **Stakeholder** | **Viewpoint(s) that apply to that class of stakeholder's concerns** |
|---|---|
| Customer | Use-Case, Process, Deployment |
| End user | Use-Case, Process |
| Developer | Use-Case, Process, Logical, Data, Deployment |
| Tester | Use-Case, Process, Deployment |

## Use-Case Viewpoint Definition

### Abstract

Viewpoint describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

### Stakeholders and Their Concerns Addressed

All stakeholders of the system, including the end-users for high-level overview of the relationships between use cases, actors, and subsystems.

### Elements, Relations, Properties, and Constraints

Elements of the use-case viewpoint are actors that portrays any entity (or entities) that perform certain roles in the system and use cases that describes the interactions that take place between actors and the system during the execution of business processes.

Viewpoint relationship is a type of model element that adds semantics to a model by defining the structure and behavior between the model elements. It includes activity edges, associations, dependencies, and generalizations.

## Process Viewpoint Definition

### Abstract

Viewpoint describes the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system.

### Stakeholders and Their Concerns Addressed

All stakeholders of the system, including the end-users to understand the main processes of the application at the level of their sequential interaction.

### Elements, Relations, Properties, and Constraints

Elements are the main actions that occur during the described process, at a high level of abstraction with an emphasis on the business process itself. They are combined into groups to determine the subsystems in which certain stages of the process take place.

Relationships between the stages of the process determine the sequence of their execution.

## Logical Viewpoint Definition

### Abstract

A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers.

### Stakeholders and Their Concerns Addressed

The viewpoint is helpful within the functional analysis, and it's also serves to identify common mechanisms and design elements across the various parts of the system, which can be useful for developers and testers.

### Elements, Relations, Properties, and Constraints

The system is decomposed into a set of key abstractions, taken (mostly) from the problem domain, in the form of objects or object classes. They exploit the principles of abstraction, encapsulation, and inheritance. So used elements of the viewpoint are layers, classes, boundaries, and controls. Relations specifies association, aggregation, inheritance, or usage of elements within each other.

## Deployment Viewpoint Definition

### Abstract

A description of the deployment view of the architecture describes the various physical nodes for the most typical platform configurations.

### Stakeholders and Their Concerns Addressed

Viewpoint shows the mapping of the software onto the hardware and shows the system's distributed aspects. It's useful for developers who deploy system infrastructure as well as for customers because of security aspects of nodes distribution and communication firewalls. Along with this, for effective testing, a high-level understanding of the distribution of the system at the level of physical components is also necessary.

### Elements, Relations, Properties, and Constraints

Elements represent computational, communicational or data resources of the system. Relations between two nodes define communication between elements in any form: HTTP requests, via message brokers etc.

## Data Viewpoint Definition

### Abstract

Describes the architecturally significant persistent elements in the data model.

### Stakeholders and Their Concerns Addressed

The viewpoint is useful for developers who may operate with the data model at the business logic level to understand the essential models of the system and the relationships between them.

### Elements, Relations, Properties, and Constraints

Elements of view is any distinguishable object that is to be represented in the database. Relationship: used to designate a logical association between entities. It is usually qualified by the cardinality of the participant entities: one-to-one, one-to-many, or many-to-many relationship. Conceptually, there are no topological constraints with respect to the relations in a data model.

# How a View is Documented

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5.

Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

Section 3.i: Name of view.

Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.

Section 3.i.2: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.

Section 3.i.3: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of subsections for (respectively) elements, relations, interfaces, behavior, and constraints.

Section 3.i.4: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.

## Relationship to Other SADs

Not applicable.

# Architecture Background

## Problem Background

The current rapid pace of urbanization is one of the main problems of large cities - oversaturation with cars. In addition to heavy traffic on time, there is also a lack of packing areas. Each new building, hundreds of which appear on the first vacant lawn each year, covers the need for parking spaces at best by 40%. The logical consequence of this is that drivers often violate all generally accepted parking rules, creating inconvenience for others, and sometimes even creating emergencies. City authorities enter contracts with evacuation services, but there is still no centralized means by which a citizen can report an illegal parking act, and the evacuation service operator can analyze the offense and call the crew only if necessary.

### System Overview

The main purpose of the system is to automate communication between citizens and evacuation services operating in the city. Such an automation system is being created with the aim of improving the quality of life in cities by controlling cases of illegal parking in the most efficient manner.

### Goals and Context

For the target city of the system, a citizen who has witnessed a violation of parking rules must contact the patrol police. However, according to the inspectorate, this can be done only in writing. To do this, you need to take a few photos that show the area, road signs or markings, indicate the city, address, date, car license plates and contact details of the reporting citizen. The report must then be printed out and sent by letter to the patrol police department. The police, in turn, must review the complaint within 30 days, identify the offender and punish him accordingly.

Thus, based on the analysis of existing reporting mechanisms and algorithms of state and evacuation services, the following processes were identified: reporting on offenses, processing the report by an evacuation service expert, evacuation of a car by crew, administration of human and automobile resources of evacuation service.

The offense reporting process as described above should allow a person to provide the following information:

- Panoramic photos which show all needed information to determine the act of the offense.
- Car license plate to identify the offender.
- Location of the offense for appropriate services response if necessary.

### Significant Driving Requirements

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

- The human factor in identifying the offender must be minimized. Thus, manual entry of license plates is excluded.
- A mechanism for integration with state registers is needed to obtain information about the infringer on the provided license plates.
- The possibilities of deploying system components are limited, in view of this, full cloud compatibility with the target platform is required.
- The scale of using the system directly depends on its convenience for users in each of the roles. For citizens, it is important to provide the necessary information as quickly and conveniently as possible, which dictates the need for native accessibility of the system. At the same time, it is important to consider that the part of the system associated with the processing of reports and the administration of resources has large amount of complex functionality and the system must have high portability.
- Since access to the relational storage of reports will be performed by many users at the same time, it is necessary to ensure the integrity of the operations and data performed.

# Solution Background

## Architectural Approaches

The approach to providing information about the violation by the user is a fundamental point in the formation of the architecture. In view of the requirements to exclude manual entry of identification data, the choice was made in the direction of providing a license plate in the form of a photo. At the same time, the idea of assigning the responsibility to the expert to manually search for the necessary information in the registers by the number in the photo potentially reduced the efficiency of the process and did not exclude the human factor from it.

It was decided to automate the process of recognition in the photo by using computer vision. At the same time, due to limited resources, the option of developing our own neural networks to solve this popular problem was not considered. Given the target platform for deploying resources, the built-in Computer Vision service on Azure is used for these purposes.

Further, the system is divided into 4 functional components:

- Mobile app. It is used by citizens to provide all the information necessary for the report, as well as by evacuation crews to receive information about assigned evacuations. The choice in favor of the mobile application was indicated by its ease of accessibility and native user experience. It was clear that for users in these roles, a smartphone would be the tool for accessing the system. And a web application, in the context of the requirements set, cannot compete with its native counterpart.
- Web application. Unlike the subsystems of the citizen and the evacuation crew, the functional load for experts and administrators is much higher. In view of this, personal computers are the priority target machine for these subsystems. Requirements for accessibility and portability justified the choice in favor of a web application with a near-native user experience, which a demanding desktop application could not provide.

- Data access API. To organize a single point of access to data, which has appropriate mechanisms to protect data and ensure the integrity of operations performed, all software components of the system access the database through a specialized API.
- Recognition functions. The task of this component is to encapsulate the logic for obtaining data about the offender from a license plate photo. The first function accesses the CV resource to obtain a string value, and the second receives the corresponding information from the registers. In this way, maximum modularity and extensibility of this participatory process is achieved.

.NET was chosen as the target platform for the implementation of the above components. The following features were considered:

- Since the system assumes cloud hosting, the target platform should ideally have tight integration with the cloud provider.
- To facilitate future support, all software components must be implemented on a single platform.

The multi-level organization of the code with elements of the microservice architecture made it possible to achieve a modular implementation broken down by duties.

## Analysis Results

According to the load tests, the allocation of the data access layer as a separate component made it possible to separate the load of operating with the database from the capacities of web and mobile applications. In this way, the API can independently scale depending on the workloads, as well as encapsulate the logic of maintaining the integrity of parallel data operations.

All components of the system successfully interact with each other thanks to a single platform, which made it possible to use common resource libraries.

An independent license plate recognition module provides efficient closed integration with computer vision service and government registries.

## Requirements Coverage

Exclusion of the human factor in the identification of the offender: The identification process takes place by processing the photo of the license plate using computer vision with further automated receipt of the relevant information from the registries.

Cloud compatibility: All system components are designed in such a way as to be directly reflected as a resource on cloud platforms. Thus, even the level of IaaS is redundant, the system operates with PaaS.

Subsystems UX: The citizen and evacuation worker subsystems are implemented as a mobile application, which provides the necessary accessibility and ease of use. In turn, the more loaded expert and administrator subsystems are delivered as a full-fledged web application.

Database consistency: All operations with the system database are organized through a separate software component that ensures the sequence of operations and data integrity.

## Product Line Reuse Considerations

Not applicable.

# Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

## Use Case View

### View Description

The view contains use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system. Central process of the system is a report processing. A citizen accesses the system and report about the offence. Evacuation service expert evaluates the situation and in case of violation confirmation may assign the report to available crews that carry out the evacuation.

### Primary presentation

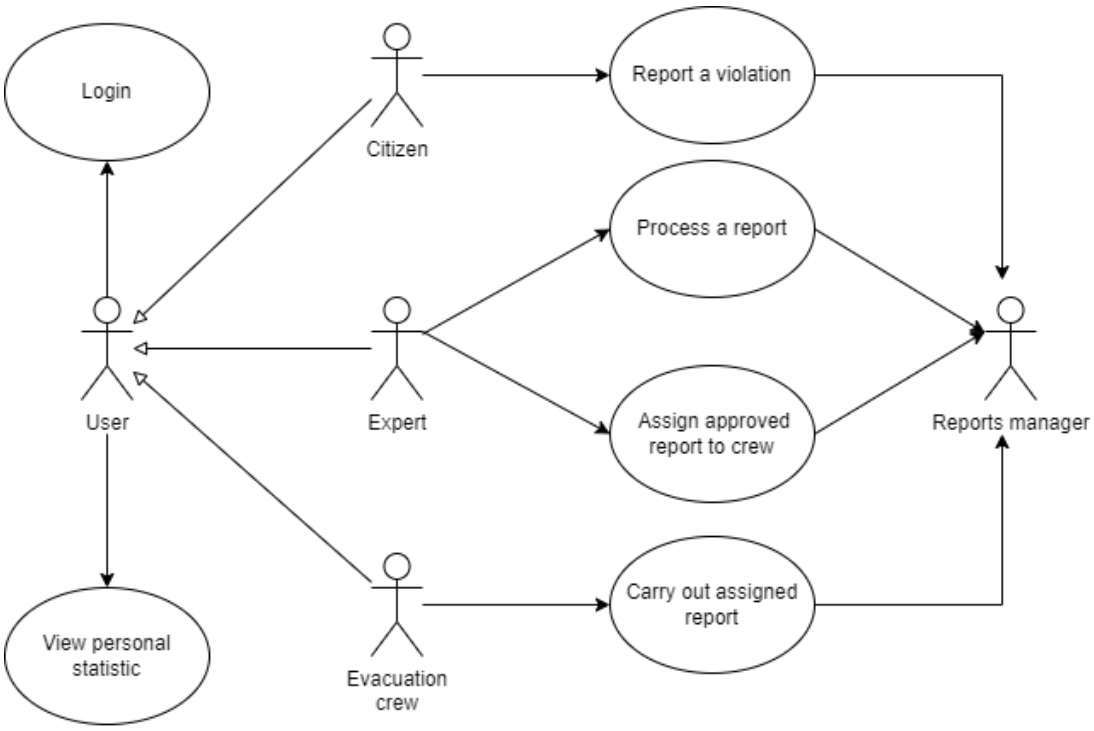



*Figure 1: Architecturally significant Use Cases*

### Elements catalog

Login. This use case describes how a user logs into the Public Parking Control system. The actors starting this use case are Citizens, Experts and Evacuation Crews.

View Personal statistic. Use case describes user's ability to view personal statistics. For citizens, these are submitted reports, for experts - processed reports, for crews – carried out as assigned.

Report a violation. This use case allows a citizen to report a violation of the parking rules by providing panoramic photos and location.

Process a report. This use case allows an expert to view the submitted violation report with all the necessary information and decide on its validity.

Assign approved report to crew. This use case allows an expert to assign a valid report to one of the available evacuation crews.

Carry out assigned report. This use case allows an evacuation crew to choose one of the assigned reports and, in accordance with it, perform the evacuation of the violator's car.

### Architecture background

The authentication process is centralized for all users, regardless of the hosting platform of their subsystem. At the same time, the report manager encapsulates all the necessary logic for storing and accessing data.

## Process View

### View Description

The view describes the stage-by-stage processing of the report as the main process of the system's life. The division of stages into responsible subsystems gives an understanding of how the process is distributed among the components of the system.

### Primary presentation

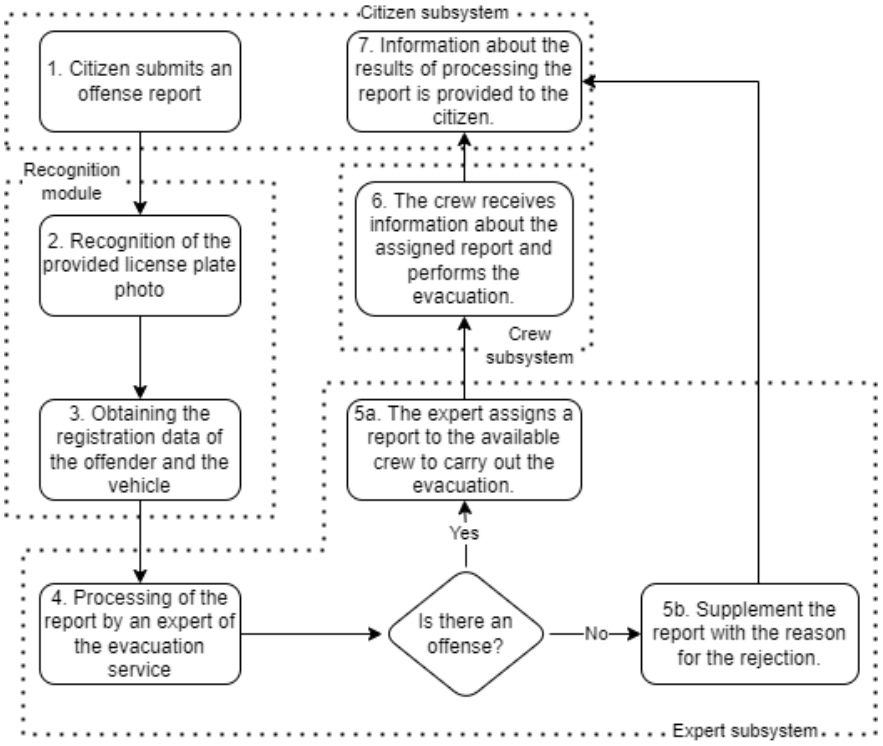



*Figure 2: Report processing flowchart*

### Elements catalog

1: A citizen who discovered the offense generates a report containing the location, panoramic photo, and photo of the license plate, as well as supporting information if necessary.

2: Automated process of license plate recognition on the provided photo with subsequent transfer of its string value.

3: Integration with state automobile registries with obtaining detailed information about the car and its owner.

4: Based on the information received, the expert of the evacuation service decides on the validity of the report.

5a: If the fact of violation is confirmed in the report, the expert assigns it to one of the crews. In this case, the choice of the crew is based on its load.

5b: If the expert did not find an offense in the submitted report, the system makes it possible to reject it. In this case, a prerequisite is an indication of the reason for the refusal.

6: When a report is assigned to a crew, its members receive a corresponding notification within their subsystem. After the evacuation is completed, the report is marked as completed.

7: Regardless of the validity of the submitted report, upon completion of its processing, the citizen is provided with the resulting information.

### Architecture background

The stages of the process are clearly divided into the corresponding subsystems, which allows them to be implemented by separate components at the software and physical levels.

## Logical View

### View Description

System applications are divided into layers based on the N-tier architecture. The layering model is based on a responsibility layering strategy that associates each layer with a particular responsibility. So, this view describes architectural layers and their dependencies on each other.

### Primary presentation

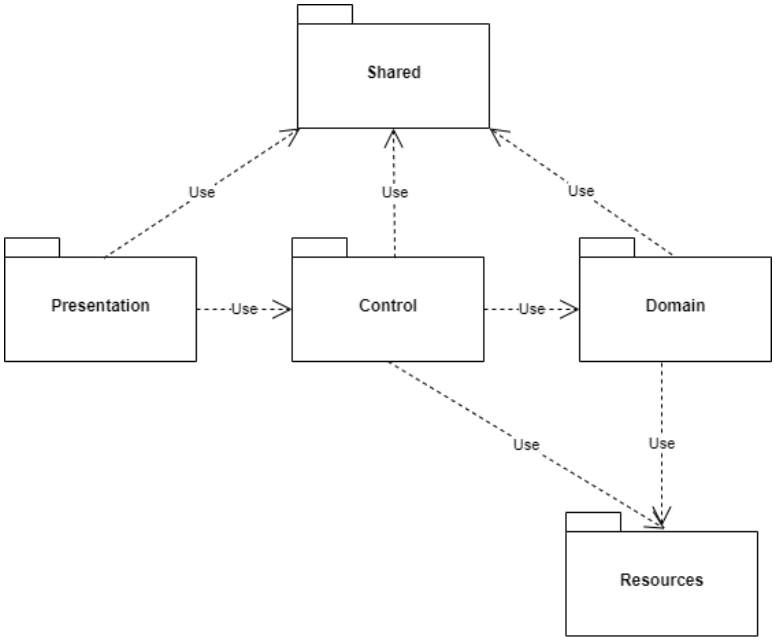



*Figure 3: Software's layered architecture diagram*

### Elements catalog

Presentation Layer. Deals with presentation logic and the pages rendering.

Control Layer. Manages the access to the domain layer.

Domain Layer. Related to the business logic and manages the accesses to the resources layer.

Resources Layer. Responsible for the access to the enterprise information system (database, data warehouse, storages etc.)

Shared Layer. Gathers the common objects reused through all the layers.

### Architecture background

This strategy has been chosen because it isolates various system responsibilities from one another, so that it improves both system development and maintenance.

## Deployment View

### View Description

All system components are deployed within the Azure cloud infrastructure. Within the diagram, each node is the corresponding platform cloud resource (except mobile app that uses corresponding publishing platforms).

## Primary presentation

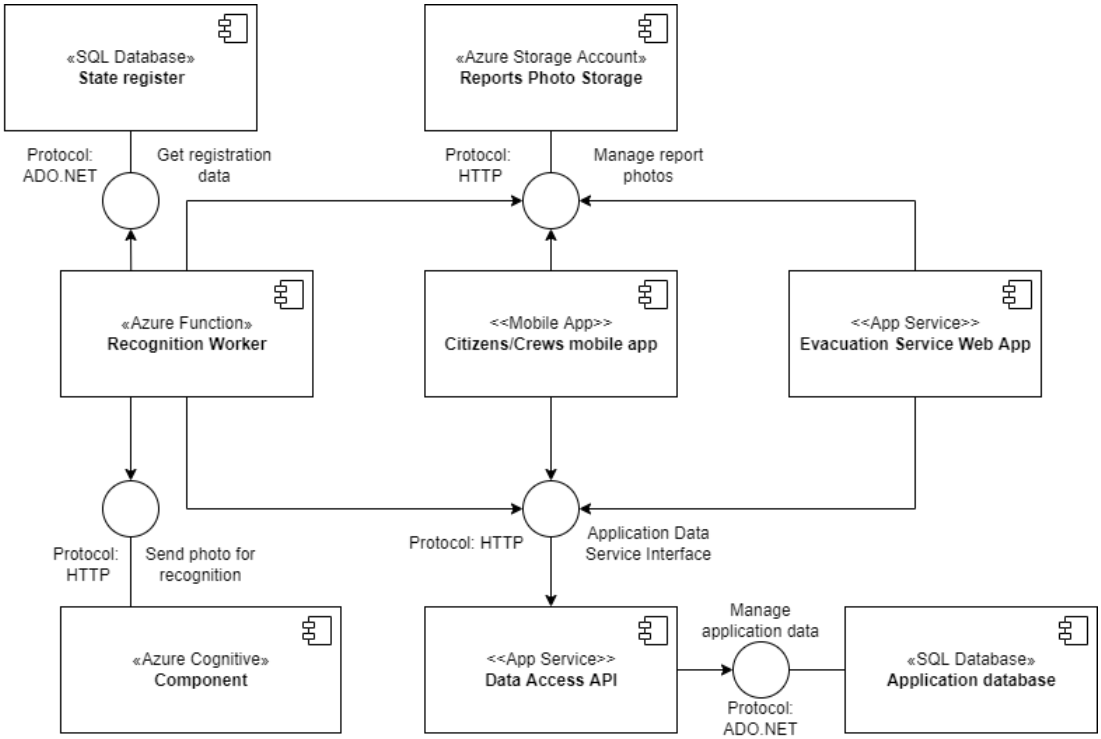



*Figure 4: Component diagram*

## Elements catalog

Evacuation Service Web App. Portal for evacuation experts and administrators. Provides the process of processing the report by an expert with its subsequent assignment to the evacuation crew, as well as the possibility of administering the resources of the evacuation service (experts, evacuation vehicles, etc.)

Citizens/Crews mobile app. Provides a process for submitting a report by a citizen, as well as displaying information about assigned reports for evacuation crews.

Data Access API. Contains operations for interacting with system data, serves as a single-entry point to the database.

System database. SQL Database that contains all the main data of the system, apart from state registries.

Photo Storage for Reports. Storage account for blobs that stores photos that citizens attach to their reports. Serves as photo source for Web App and Recognition Worker.

Recognition Worker. Azure Function that provides the process of recognition of a photo of a license plate via Azure Cognitive with the subsequent obtaining of the corresponding data from state registries.

Azure Cognitive. A computer vision service that performs the task of recognizing a license plate in a photo.

State Register. Data storage of the state register of automobiles. It is an external government source and is not part of the deployment component of the system.

### Architecture background

The view displays several architecturally important decisions at once:

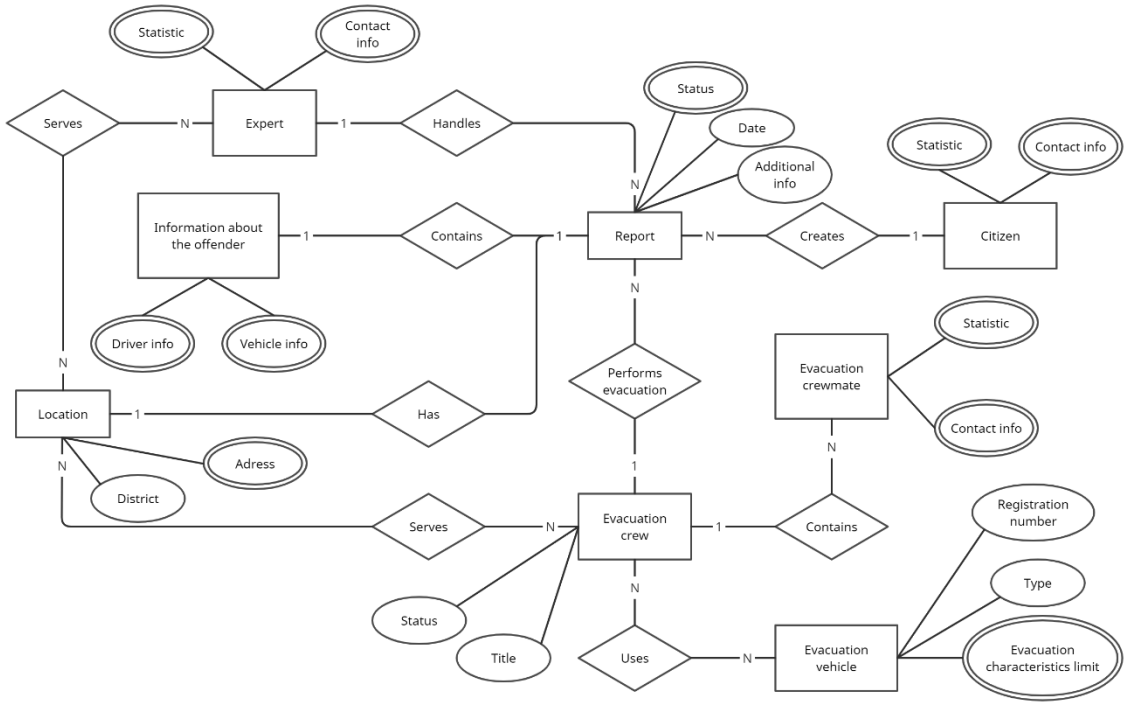
- In view of the different requirements of the subsystems in terms of user experience, they are divided into mobile and web applications.
- To maintain the integrity and sequence of performing operations with data by several clients (web, mobile, functions set etc.), such operations are placed in a separate API.
- The process of obtaining registry data from a license plate photo is encapsulated in a scalable block of functions.

## Data View

### View Description

The view shows architecturally significant persistent elements in the data model as well as their relationships with each other and some additional properties for each entity.

### Primary presentation




### Elements catalog

Report. The central entity of the system. It contains information about the offense transmitted by the citizen and is the subject of processing for the expert and further appointment for the evacuation crew.

Citizen, Expert, Evacuation crewmate. Projection of real users of the system depending on their role.

Evacuation crew. The crew consists of several crewmates as well as recovery vehicles at their disposal. The characteristics of the available evacuation vehicles of the crew determine whether it is suitable for a particular report.

Location. A report created by a citizen is specific to a specific location. At the same time, evacuation experts and crews maintain their list of locations determined in districts scope. Thus, the distribution of the load between the workers of the service is achieved.

Information about the offender. Registration data about the car and its owner, pulled from the registers based on the car number.

## Elements catalog

The star-like structure of building a data model is dictated by the organization of the business processes of the system. Since most processes are focused on reports, such an organization makes it possible to simplify data handling as much as possible by reducing the chains of relationships.

# Referenced Materials


| | |
|---|---|
| Barbacci 2003 | Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>. |
| Bass 2003 | Bass, Clements, Kazman, *Software Architecture in Practice*, second edition, Addison Wesley Longman, 2003. |
| Clements 2001 | Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley Longman, 2001. |
| Clements 2002 | Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002. |
| IEEE 1471 | ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000. |

# Directory

## Glossary

| Term | Definition |
|---|---|
| software architecture | The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. |
| view | A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint. |
| view packet | The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets. |
| viewpoint | A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. |

# Acronym List

| | |
|---|---|
| API | Application Programming Interface; Application Program Interface; Application Programmer Interface |
| ERD | Entity Relationship Diagram |
| HTTP | Hypertext Transfer Protocol |
| IEEE | Institute of Electrical and Electronics Engineers |
| OO | Object Oriented |
| OS | Operating System |
| SAD | Software Architecture Document |
| SQL | Structured Query Language |
| UML | Unified Modeling Language |
| | |
| | |
| | |
| | |
| | |
| | |
| | |