НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Формалізація поведінки автомобіля за особливостями руху на основі камер відеоспостережень»

Виконав (-ла):
студент (-ка) IV курсу, групи ТВ-61
Гарник Олексій Ігорович _____

Керівник:
доцент, к.т.н.
Шаповалова Світлана Ігорівна _____

Рецензент:
к.т.н. ст. викладач кафедри ТЕУТ та АЕС _____
Рачинський А.Ю.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Гарнику Олексію Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи _Формалізація поведінки автомобілів за особливостями руху на основі камер відеоспостережень_____ _____
керівник роботи___доцент, к.т.н. Шаповалова Світлана Ігорівна_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи _____”5” червня 2020р._____
3. Вихідні дані до роботи __Мови програмування — Python, C#, Typescript, середовища розробки — PyCharm, Visual Studio 2019, Visual Studio Code_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) _розробити інтелектуальну систему на прикладі веб системи, що дозволяє визначити траєкторію руху транспортних засобів за допомогою методів виявлення та алгоритмів відстеження, ідентифікації об’єктів, що в майбутньому стануть базою для обробки та математичного аналізу. Розробити мікросервісну архітектуру, що забезпечить складну та масштабовану програму. Розробити інтуїтивно зрозумілий і зручний користувацький інтерфейс_____

5. Перелік ілюстративного матеріалу_Призначення та актуальність, Мета, Задачі та вимоги, Виявлення транспортних засобів за допомогою YOLOv3, Відстеження

транспортних засобів за допомогою фільтра Калмана, Ідентифікація транспортних засобів з Угорським алгоритмом, Загальна схема моніторингу руху транспортних засобів, архітектура застосунку, діаграма прецедентів, ER-модель, взаємодія користувача з системою, Результати проаналізованого відеофайлу, Проблеми та компенсації неточності виявлення об'єктів, Випадок часткової оклюзії, Випадок повної оклюзії, Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "11" ____жовтня____ 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	11.10.2019 - 23.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020 - 04.03.2020	
4.	Розробка структур окремих підсистем	05.03.2020 - 12.04.2020	
5.	Програмна реалізація системи	13.04.2020 - 17.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020 - 07.06.2020	
7.	Захист програмного продукту	27.05.2020	
8.	Передзахист	10.06.2020	
9.	Захист	15.06.2020	

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Бакалаврська дипломна робота Гарника Олексія Ігоровича на тему “Формалізація поведінки автомобіля за особливостями руху на основі камер відеоспостережень”.

Мета роботи — створення інтелектуальної системи на прикладі веб застосунку, що дозволяє визначити траєкторію руху транспортних засобів за допомогою методів виявлення та алгоритмів відстеження, ідентифікації об’єктів, що в майбутньому стануть базою для обробки та математичного аналізу.

Під час виконання бакалаврської роботи був проведений аналіз існуючих методів відстеження об’єктів. Була спроектована мікросервісна архітектура програмного забезпечення, що дозволяє масштабувати систему в майбутньому, модифікувати існуючі сервіси. Розроблений інтуїтивно зрозумілий та зручний користувацький інтерфейс.

Результатом роботи є працездатна та протестована інтелектуальна веб система, опис архітектури серверної частини та клієнтського застосунку.

Розробка веб системи відбувалася в середовищах: Visual Studio 2019, PyCharm і Visual Studio Code.

Обсяг роботи: 72 сторінок, 38 рисунків, 44 формул, 11 використаних джерел, 3 додатки.

Ключові слова: згорткові нейронні мережі, фільтр Калмана, виявлення об’єктів, трекінг об’єктів, комп’ютерний зір, Угорський алгоритм.

ABSTRACT

Bachelor's thesis of Oleksiy Garnik on the topic "Formalization of vehicles behavior according to the peculiarities of movement on the basis of video surveillance cameras".

The aim of the work is to create an intelligent system on the example of a web application that allows to determine the trajectory of vehicles using detection methods and tracking algorithms, identification of objects that will become the basis for future processing and mathematical analysis.

During the bachelor's thesis, an analysis of existing methods of detecting and tracking objects was conducted. A microservice software architecture was designed to scale the system in the future and modify existing services. Developed intuitive and user-friendly interface.

The result is a working and tested intelligent web system, a description of the architecture of the server part and the client application.

Web system development took place in the following environments: Visual Studio 2019, PyCharm i Visual Studio Code.

Explanatory note: 72 p., 38 fig., 44 formulas, 11 references.

Keywords: convolution neural networks, Kalman filter, object detection, object tracking, computer vision, Hungarian algorithms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

YOLO — You Only Look Once

SPA — Single Page Application

KF — Kalman Filter

EKF — Extended Kalman Filter

UKF — Unscented Kalman Filter

UT — Unscented Transform

IOU — Intersection Over Union

API — Application Programming Interface

HTTP — HyperText Transfer Protocol

AMQP — Advanced Message Queuing Protocol

CRUD — Create Read Update Delete

DTO — Data Transfer Object

NMS — Non Maximum Suppression

ВСТУП

Комп’ютерний зір і штучний інтелект — одні з найбільш затребуваних напрямків в сучасному світі ІТ-технологій. На сьогоднішній день комп’ютерний зір стрімко зростаюча сфера цифрового виробництва, що широко застосовується в багатьох галузях економіки, таких як автономні автомобілі, системи допомоги водію, системи контролю тощо.

Відеоспостереження вже давно увійшло в наше життя, як ознака безпеки і є її невід’ємною частиною. По-перше, однією з найбільших переваг його застосування є те, що лише наявністю воно знижує рівень злочинності та кількість правопорушень. Камери відеоспостереження слугують стримуючим фактором. По-друге, відеоспостереження дозволяє вести історію записів, які надалі можна буде застосовувати згідно часу та даті під час розслідування реальних подій. Цим самим камери підвищують рівень громадської безпеки.

Проте слід зауважити, що в більшості випадків системи відеоспостереження є пасивними спостерігачами. Всі рішення про те, які з подій, що відбуваються на екрані є інцидентами приймаються оператором. Задача оператора здебільшого зводиться до протяжних періодів очікування чогось незвичного на відеомоніторі. При цьому на екрані монітора спостерігача відображається картинка з десятків камер, і в результаті людині досить просто помилитися. Спостереження є дуже важлива робота, але в той же час дуже втомлива. Крім цього не завжди вистачає професійних людських ресурсів для її виконання та супроводу. За статистичними даними та оцінками психологів, середній час утримання уваги людини на одному об’єкті не перебільшує 14 хвилин. Тому актуальним є створення системи інтелектуального відеоспостереження, задача яких є розпізнавання незвичних подій та об’єктів на кадрах відеоспостереження.

1. ПОСТАНОВКА ЗАДАЧІ З ФОРМАЛІЗАЦІЇ ПОВЕДІНКИ АВТОМОБІЛЯ ЗА ОСОБЛИВОСТЯМИ РУХУ НА ОСНОВІ КАМЕР ВІДЕОСПОСТЕРЕЖЕННЯ

Особливу популярність набувають системи контролю на дорогах. Проте на сьогоднішній день порушення правил дорожнього руху не обмежуються лише фіксуванням перевищення швидкості. Існують не менш суттєві порушення, що можуть призводити до дорожніх транспортних пригод. Тому для забезпечення необхідних умов та даних для аналізу поведінки автомобілів на дорозі та вирішення перелічених вище задач, в першу чергу, необхідно визначити траєкторію руху транспортних засобів. Саме завдяки цим даним, можна чітко зрозуміти пересування об’єктів з відеофайлів та після відповідного математичного аналізу вивести заключення щодо порушення. Таким чином задача підвищення безпеки дорожнього руху є актуальною, а створення відповідного програмного та алгоритмічного забезпечення має практичну значущість.

Для розробки відповідного програмного забезпечення було запропоновано веб систему функціями якої є забезпечення користувача:

1. Особистим кабінетом, де він зможе завантажувати відео для подальшого аналізу та переглядати звіти про виконану роботу. Користувач отримує структуровані дані у вигляді наборів координат відносно відповідних кадрів, а також візуалізовані дані у виглядів графіків, що зберігаються в особистому кабінеті.

2. Можливістю довільно позначати необхідні зони, що надалі допоможуть при подальшому аналізі. Наприклад, місця недозволені для паркування, додаткові параметри для визначення швидкості, проїзд в заборонених частинах тощо.

Вимогами до програмної системи є забезпечення:

Вимогами до програмної системи є забезпечення:

1. Ефективності та працездатності системи, тобто в разі аварійної ситуації, при деяких неполадках в роботі більш складних сервісів (наприклад того, що відповідає за визначення траєкторії), застосунок має продовжувати працювати, а користувач —

все ще не втратити можливість переглядати історію своїх проаналізованих відеофайлів.

2. Гнучкої архітектури програмного забезпечення, що дозволить з легкістю масштабувати систему в майбутньому, вносити зміни, а саме розробляти додаткові сервіси з аналізом даних та винесенням заключенням щодо поведінки автомобілів.

3. Інтуїтивно зрозумілим і зручним користувацьким інтерфейсом.

Для реалізації описаних вище функцій необхідно виконати наступні задачі:

1. Для забезпечення користувача особистим кабінетом потрібно реалізувати процес автентифікації на основі JSON Web Token (JWT).

2. Провести аналіз існуючих алгоритмів для виявлення, відстеження та ідентифікації транспортних засобів.

3. Розробити єдину точку входу (API Gateway) за допомогою Ocelot.

4. Реалізувати взаємозв’язок між мікросервісами на основі подій за допомогою RabbitMQ.

5. Розробити незалежні модулі для мікросервісної архітектури:

- DrawingService, що дозволяє позначати необхідні додаткові зони для аналізу.
- AnalyzerService, що виконує задачі виявлення, відстеження та ідентифікації транспортних засобів.
- StorageService, що відповідає за збереження даних, а також розсилає результати проаналізованих відео файлів на пошту користувачам.
- IdentityService, що відповідає за задачі авторизації та реєстрації.

6. Для реалізації користувацького інтерфейсу використати фреймворк Angular, що націлений на розробку SPA рішень та ставить в пріоритеті на якість розробки з упором на максимально можливу ефективність (швидкість роботи програми, масштабування тощо).

Отже, метою дослідницької роботи є створення інтелектуальної системи на прикладі веб застосунку, що дозволяє визначити траєкторію руху транспортних засобів за допомогою методів виявлення та алгоритмів відстеження, ідентифікації об’єктів, що в майбутньому стануть базою для обробки та математичного аналізу.

2. ІНСТРУМЕНТИ ТА ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Проаналізувавши вимоги до поставленої задачі було вирішено розробити інтелектуальну систему на прикладі веб застосунку. Такий вибір обґрунтовується наступними перевагами:

1. Користувач не потребує встановлення на свою машину великовагового програмного забезпечення. Все, що потрібно для повноцінної роботи — це браузер, який зазвичай поставляється з операційною системою, і доступ в Інтернет.

2. Встановлюючи додатки на свій комп’ютер, мимоволі доводиться брати на себе обов’язки адміністратора, що дозволяє недосвідченим користувачам масу клопоту. Додаток потрібно встановити і запустити, потім налаштувати під себе, а також виправити можливі помилки, які потребують негайного вирішення. У випадку з браузерним додатком, що фактично лежить на сервері, турбуватися про це не доведеться.

3. Веб додатки не вимогливі до ресурсів і не пред’являють ніяких вимог до апаратної платформи. Це означає, що немає ніякої різниці, скільки мегабайт оперативної пам’яті встановлено на комп’ютері користувача і з під якої операційної системи він працює. Головними вимогами є наявність браузера і доступ в Інтернет.

4. Немає ніяких проблем з підтримкою старих версій програм. Коли з’являється нова версія десктопного чи мобільного додатку, користувачам нерідко доводиться вирішувати проблеми, пов’язані з оновленням вже встановленої на їх машині копії. У випадку з браузерними додатками таких проблем не виникає — існує тільки одна версія, в якій працюють всі користувачі, і в разі виходу нової всі без винятку автоматично переходять на неї, часом навіть не помічаючи цього.

5. Веб додатки дозволяють своїм користувачам бути по-справжньому мобільними. Тобто ви можете працювати в мережі, зберігати результати своєї роботи на сервері і, в разі необхідності, мати до них доступ звідусіль, де є вихід в Інтернет.

Для розробки даного застосунку були використані наступні середовища програмування:

1. Visual Studio 2019 для розробки відповідних модулів програми, таких як сервіси авторизації та реєстрування в системі, а також мікросервіса, що відповідає за збереження даних у вигляді історії оброблених відеофайлів, розсилки результатів користувачам на пошту.

2. PyCharm для розробки мікросервіса, що відповідає за визначення, відстеження та ідентифікації транспортних засобів.

3. Visual Studio Code для розробки клієнтської частини.

2.1. Технології та мови програмування

Для розробки інтелектуальної веб системи було прийнято рішення про використання мікросервісної архітектури. Тому застосунок поділяється на окремі незалежні модулі розроблені за допомогою різних технологій. Основною мовою програмування було обрано C#. Обґрунтування вибору мови:

1. Дана мова використовує об’єктно-орієнтований підхід до програмування у всьому. Це означає, що необхідно описувати абстрактні конструкції на основі предметної області, а потім реалізовувати між ними взаємодії. Даний підхід користується великою популярністю, тому що дозволяє не тримати в голові всю інформацію.

2. Замість того, щоб писати велику кількість рядків коду, є можливість використати готову конструкцію, а компілятор зробить всю необхідну роботу. Незважаючи на те, що деякі такі конструкції не є найоптимальнішими з точки зору продуктивності. Але все це перекривається внаслідок легкості читання коду і високою швидкістю розробки.

3. Наявність великої кількості бібліотек і шаблонів, що дозволяють зекономити час в розробці.

Саме цією мовою розроблені окремі модулі:

1. IdentityService, що відповідає за задачі авторизації та реєстрації.

2. StorageService, що відповідає за збереження даних, а також розсилає результати проаналізованих відео файлів на пошту користувачам.

Для розроблення наступних модулів було використано мову програмування Python та бібліотеку OpenCV:

1. DrawingService, що дозволяє позначати необхідні додаткові зони для аналізу.

2. AnalyzerService, що виконує задачі виявлення, відстеження та ідентифікації транспортних засобів.

Вибір був зроблений на користь Python, адже мова ідеально підходить для задач машинного навчання, дозволяє створювати швидкі прототипи рішень, тобто дозволяє експериментувати з великою кількістю ідей проектування. Також Python має велику кількість бібліотек для роботи з даними, що знову дозволяє зекономити час під час реалізації. Бібліотека комп'ютерного зору і машинного навчання, як OpenCV [1], має інтерфейс на мовах програмування Python та C++. Це фактично означає, що можна отримати перевагу на етапі реалізації рішення на Python, а в майбутньому для оптимізації продуктивності за необхідності переписати на C++.

Основним середовищем для роботи з даними технологіями було обрано PyCharm. Основним середовищем програмування для розроблення інтерфейсу веб системи було використано Visual Studio Code.

2.2. Вибір архітектури програмного застосунку

Для реалізації поставленої задачі та згідно з висунутими вимог до програмного забезпечення було вирішено розробити мікросервісну архітектуру. Нині існує декілька підходів [2, 3] до розробки програмного забезпечення, а саме:

1. Монолітна архітектура.

2. Мікросервісна архітектура

Вибір був зумовлений на користь останньої через ряд переваг та можливостей. Нині тенденція на мікросервісну архітектуру стрімко зростає.

2.2.1. Монолітна архітектура

Монолітна архітектура вважається традиційним способом побудови додатків. Монолітний додаток будується як єдине і неподільне ціле. Зазвичай таке рішення включає користувацький інтерфейс, серверну частину та базу даних. Він уніфікований і всі функції керуються та обслуговуються в одному місці. Зазвичай монолітні програми мають одну велику базу коду та відсутність модульності.

Якщо розробники хочуть щось оновити або змінити, вони отримують доступ до тієї ж бази коду. Отже, вони вносять зміни в цілому стеку одразу. У більшості випадків розглядаються Presentation Layer, Business Logic Layer і Layer Access Data. Ідея цієї сегрегації полягає в роботі з будь-яким компонентом архітектури, незалежним від того, що знаходиться під ним або вище.

Якщо один компонент навантажений і вимагає масштабування, доводиться додавати ресурси для всієї програми. Це означає, що частина архітектури програмного забезпечення, яка погано працює, може звести всю структуру, або привиде до величезних витрат ресурсів, щоб підтримувати її та розробляти. Схема монолітної архітектури зображена на рисунку 2.1.

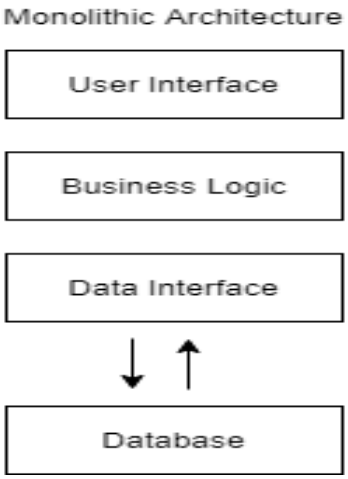

Рисунок 2.1 — Монолітна архітектура

Сильні сторони монолітної архітектури:

1. Менше наскрізних проблем. Наскрізні проблеми — це проблеми, які впливають на всю програму, таку як реєстрація, обробка, хешування та моніторинг продуктивності. У монолітній програмі ця область функціональності стосується лише одного додатка, тому з ним легше впоратися.

2. Легше налагодження та тестування. На відміну від архітектури мікросервісів, монолітні програми набагато простіше налагоджувати та перевіряти. Оскільки монолітний додаток є єдиною нероздільною одиницею, тому можна запустити тестування набагато швидше.

3. Простий у розгортанні. В монолітних додатках не доводиться обробляти багато розгортань — лише один файл або каталог.

Слабкі сторони монолітної архітектури:

1. Розуміння. Коли масштабування монолітного додатка стає занадто складним, щоб зрозуміти, то складною системою коду в межах однієї програми важко керувати.

2. Внесення змін. Важко здійснити зміни в такому великому і складному застосуванні з дуже щільним з'єднанням. Будь-яка зміна коду впливає на всю систему, тому її потрібно ретельно координувати. Це робить процес загального розвитку набагато довшим.

3. Масштабованість. Не можна самостійно масштабувати компоненти, лише всю програму.

4. Нові технологічні бар'єри. Застосувати нову технологію в монолітній програмі вкрай проблематично, оскільки тоді всю програму потрібно переписати.

2.2.2. Мікросервісна архітектура

У той час, як монолітна програма є єдиною одиницею, архітектура мікросервісів розбиває її на набір менших незалежних одиниць. Таким чином, всі служби мають свою логіку та базу даних, а також виконують конкретні функції.

В архітектурі мікросервісів вся функціональність розбита на незалежно розгорнуті модулі, які спілкуються один з одним за допомогою визначених методів, так званих API. Кожна служба охоплює власний обсяг і може бути оновлена, розгорнена та масштабована незалежно. Схема даної архітектури зображена на рисунку 2.2.

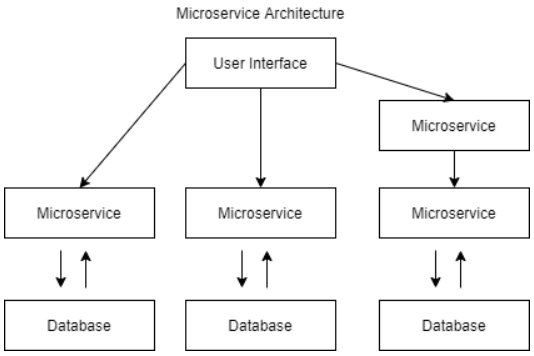

Рисунок 2.2 — Мікросервісна архітектура

Сильні сторони мікросервісної архітектури:

1. Незалежні компоненти. По-перше, всі сервіси можна розгорнути та оновити самостійно, що дає більшу гнучкість. По-друге, помилка в одному мікросервісі впливає лише на певний сервіс і не впливає на всю програму. Крім того, набагато простіше додавати нові функції до мікросервісного додатку, ніж монолітного.

2. Простіше розуміння. Розділивши на більш дрібні та прості компоненти, додаток для мікросервісу простіше зрозуміти та керувати ним, а також можна просто сконцентруватися на певній службі.

3. Краща масштабованість. Кожен елемент можна масштабувати незалежно. Таким чином, весь процес є більш економічним та економічним за часом, ніж з монолітами, коли всю програму доводиться масштабувати, навіть якщо в цьому немає потреби. Тому багато компаній закінчують перебудовувати свої монолітні архітектури.

4. Гнучкість у виборі технології. Інженерні колективи не обмежені технологією, обраною з самого початку. Вони можуть застосовувати різні технології для кожного мікросервісу.

5. Будь-яка несправність у програмі мікросервісу впливає лише на певну послугу, а не на ціле рішення. Отже, всі зміни та експерименти реалізуються з меншими ризиками та меншою кількістю помилок.

Слабкі сторони мікросервісної архітектури:

1. Додаткова складність. Оскільки архітектура мікросервісів є розподіленою системою, тому потрібно обрати та встановити з'єднання між усіма модулями та базами даних. Крім того, якщо така програма включає незалежні сервіси, всі вони повинні бути розгорнуті незалежно.

2. Розподіл системи. Архітектура мікросервісів — це складна система з декількох модулів та баз даних, тому з усіма з’єднаннями потрібно звертатися обережно.

3. Тестування. Безліч незалежно розгорнутих компонентів значно ускладнює тестування рішення на базі мікросервісів.

2.2.3. Обґрунтування вибору

Оскільки дане програмне забезпечення розраховане на розширення функціоналу в майбутньому, а саме додавання необхідних сервісів для обробки та математичного аналізу, а також згідно з поставленими вимогами до інтелектуальної веб системи доцільним рішенням було обрати мікросервісну архітектуру, що забезпечить складну та масштабовану програму. Така архітектура значно спростить масштабування та може додати нові можливості до програми.

Така архітектура є більш вигідною для складних і розвиваючих програм, тобто вона пропонує ефективні рішення для роботи зі складною системою різних функції та служб у межах однієї програми.

2.3. Інструменти для розробки мікросервісної архітектури

2.3.1. Єдина точка входу

API Gateway [4] — це компонент, який розташовується перед API та містить внутрішню мережу або брандмауер. Реалізація API шлюзу допомагає переконатися, що кожен запит ззовні повинен буде пройти через нього, перш ніж дійти до API. Наприклад, програма на базі мікросервісів може мати від 10 до 100 сервісів. API Gateway може допомогти забезпечити єдину точку входу для всіх зовнішніх користувачів, яким не важлива кількість мікросервісів, а також їх склад. Схема зображена на рисунку 2.3.

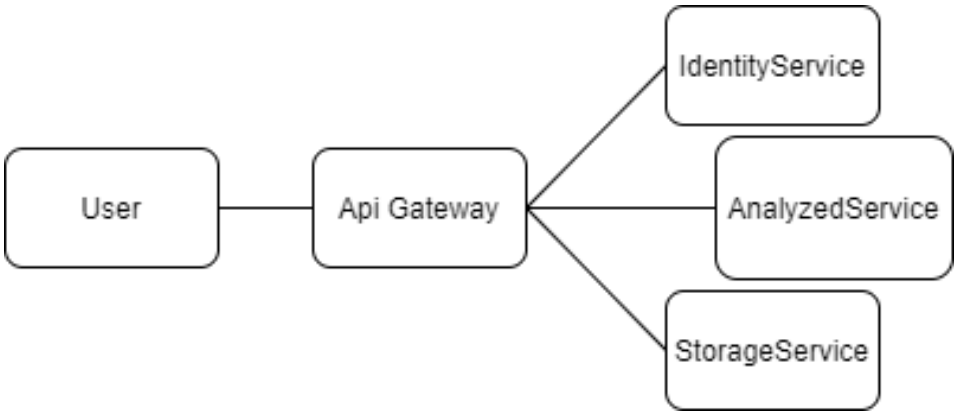

Рисунок 2.3 — API Gateway

Однак застосування API шлюзу має як переваги, так і недоліки. Найбільша перевага API шлюзу полягає в тому, що клієнтам просто потрібно звертатися лише до шлюзу, а не звертатися до певного API.

Основні функції шлюзу API:

1. Маршрутизація. Це одна з ключових функцій шлюзу API. В основному, коли API-шлюз отримує запит ззовні, він пересилає запит до відповідного внутрішнього API на основі карти маршрутизації. Ця функція ідентична функції резервного проксі, наданої веб-серверами, такими як NGINX.

2. Шлюз API реалізує деякі операції, використовуючи композицію API. Наприклад, у вашій програмі передбачена функція, що вимагає залучення декількох API, які належать до трьох різних служб. Без шлюзу API проект повинен буде звернутися до трьох API, натомість потрібно просто реалізувати API шлюз, використовуючи функцію композиції, щоб виставити лише одну кінцеву точку. Це допоможе проекту ефективно отримувати дані за допомогою одного запиту API.

3. Хешування. Зменшення кількості запитів до мікросервісів.

4. Ведення журналу. Функція ведення журналу необхідна для відстеження та налагодження.

5. Автентифікація — більшість шлюзів підтримують автентифікацію кожного запиту (або набору запитів). Згідно з правилами, заданими для кожного окремого сервісу, шлюз або направляє запит до необхідного сервісу, або повертає помилку. Більшість шлюзів доповнюють запит інформацією про автентифікації перед

відправкою конкретному сервісу, це дозволяє реалізувати специфічну логіку для конкретних користувачів в разі необхідності, тобто можна централізувати процеси автентифікації та авторизації всередині шлюзу API.

6. Обмеження швидкості. Обмеження кількості запитів в секунду для конкретного клієнта або всіх клієнтів.

Основні недоліки API Gateway

1. Збільшення складності. API Gateway — це ще одна система, яку необхідно розробити і підтримувати

2. Збільшення часу відгуку внаслідок додаткових викликів і витрат на обробку запиту

Отже, API Gateway є невід'ємною частиною в розробці мікросервісної архітектури, що надає ряд переваг та додаткових можливостей.

2.3.2. Засоби комунікації мікросервісів

Мікросервіси обслуговують конкретні функції, які є важливими для створення більших додатків. Очевидно, що мікросервіси в такому додатку потребують комунікації між собою, щоб виконати процедури. Тому одним з найважливіших аспектів розвитку мікросервісів є надання інформації від одного сервісу іншим сервісам у мірі виникнення будь-яких подій. Комунікація мікросервісів характеризуються за декількома ознаками. Перша група визначає чи є протокол синхронним або асинхронним:

1). Синхронний. Для зв'язку з веб-додатками протокол HTTP є стандартом протягом багатьох років, і він не відрізняється для мікросервісів. У синхронній комунікації клієнт відправляє запит і чекає відповіді від сервісу.

2). Асинхронний. Ключовим моментом тут є те, що клієнт не повинен блокувати потік під час очікування відповіді. У більшості випадків таке спілкування реалізується з брокерами обміну повідомленнями. Автор повідомлень зазвичай не

чекає відповіді. Він просто чекає підтвердження, що повідомлення було отримано брокером. Найпопулярніший протокол для такого типу зв'язку — це AMQP (Advanced Message Queuing Protocol), який підтримується багатьма операційними системами та хмарними провайдерами.

Кожен запит може мати різну кількість одержувачів. Дана поведінка зображена на рисунку 2.4. Запит може оброблятися одним одержувачем або службою. В разі якщо кілька одержувачів, тоді кожен запит може оброблятися різною кількістю служб. Даний тип комунікації повинен бути асинхронним.

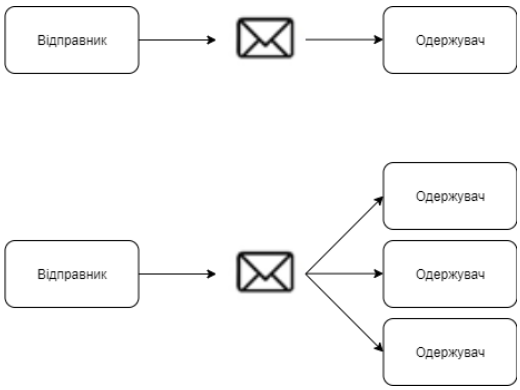

Рисунок 2.4 — Кількість одержувачів

Розроблена інтелектуальна система, що дозволяє визначати траєкторію руху автомобілів, для архітектури на основі мікросервісів використовує наступні типи зв’язку зображені на рисунку 2.5.

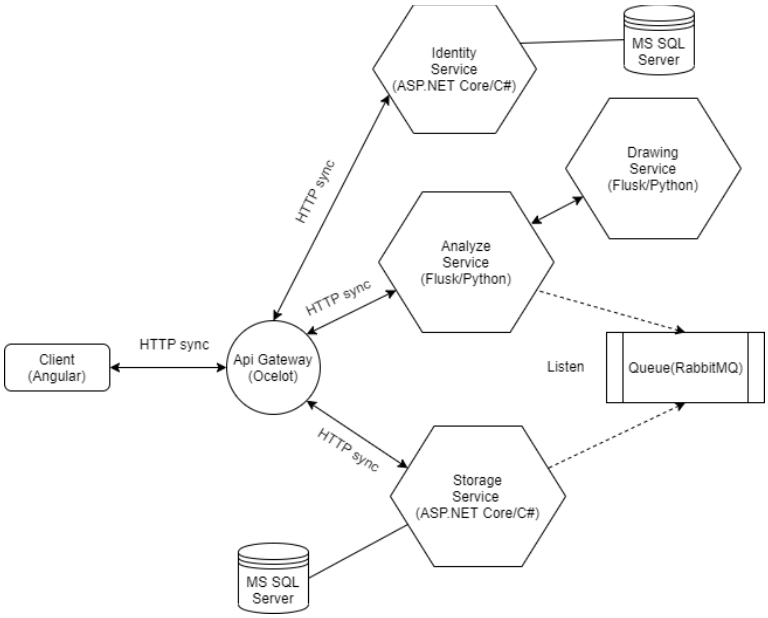

Рисунок 2.5 — Тип зв’язку архітектури на основі мікросервісах

Для реалізації комунікації мікросервісів в даному застосунку було використано RabbitMQ. В RabbitMQ існує 4 різних типи обміну (exchange), які розсилають повідомлення по-різному, використовуючи різні параметри і настройки прив’язки. Схема роботи класична:

1. Publisher публікує повідомлення в exchange
2. Exchange отримує повідомлення і тепер відповідає за маршрутизацію повідомлення
3. Необхідно встановити прив’язку між чергою і обміном.
4. Повідомлення залишаються в черзі до тих пір, поки вони не будуть оброблені клієнтами.
5. Клієнт оброблює повідомлення

Для комунікації між мікросервісами StorageService та AnalyzerService було використано тип direct exchange. Прямий обмін надсилає повідомлення в чергу на основі ключа маршрутизації.

2.3.3. Контейнеризація Docker

Docker — програмне забезпечення з відкритим вихідним кодом, що застосовується для розробки, тестування, доставки і запуску веб-додатків в середовищах з підтримкою контейнеризації. Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і перенесення в інші середовища з гарантованим збереженням стабільної роботи.

Основний принцип роботи Docker — контейнеризація додатків. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення по ізольованим середовищах — контейнерах, для запуску та розгортання програм. Кожен з цих віртуальних блоків містить всі необхідні елементи для роботи програми. Це дає можливість одночасного запуску великої кількості контейнерів на одному хості та

запуску контейнерів на будь-якій машині. Отже, у докера більше немає проблем із залежністю чи компіляцією. Все, що потрібно зробити, це запустити контейнер і програма негайно запуститься.

Переваги використання Docker:

1). Мінімальне споживання ресурсів — контейнери не віртуалізують всю операційну систему (ОС), а використовують ядро хоста і ізолюють програму на рівні процесу. Останній споживає набагато менше ресурсів локального комп'ютера, ніж віртуальна машина.

2). Швидкісне розгортання — допоміжні компоненти можна не встановлювати, а використовувати вже готові docker-образи. Наприклад, не має сенсу постійно встановлювати і налаштовувати RabbitMQ. Досить один раз інсталювати, створити образ і постійно використовувати, лише оновлюючи версію при необхідності.

3). Зручне приховування процесів — для кожного контейнера можна використовувати різні методи обробки даних, приховуючи фонові процеси.

4). Робота з небезпечним кодом — технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС.

5). Просте масштабування — будь-який проект можна розширити, запровадивши нові контейнери.

6). Зручний запуск — додаток, що знаходиться всередині контейнера, можна запустити на будь-якому docker-хості.

2.4. Архітектура клієнтського застосунку

Існує багато проблем з точки зору масштабованості, з якими стикається програміст при розробці сучасних програм. Нині клієнтські додатки не просто відображають дані та приймають користувацький ввід. Одно сторінкові додатки (SPA) надають користувачам широкі можливості взаємодії та використовують

сервер в основному в якості рівня зберігання даних. Це означає, що набагато більше відповідальності перенесено на інтерфейсну частину програмних систем. Це призводить до зростаючої складності зовнішньої логіки з якою доводиться мати справу. Згодом зростає не тільки кількість вимог, а й обсяг даних, які завантажуються у додаток. Крім того необхідно підтримувати продуктивність додатків, яка може бути дуже легко порушена.

Одним із рішень описаних вище проблем є надійна системна архітектура, проте такий вибір потребує підвищених витрат. Для розробників може бути дуже заманливо пропонувати нові функції та реалізовувати їх дуже швидко, коли система ще дуже мала. На цьому етапі все просто і зрозуміло, тому розробка плине дуже швидко. Але якщо не дбати про архітектуру, після декількох ротацій розробників, складних функцій, рефакторингів, пари нових модулів, швидкість розробки радикально сповільнюється. Нижче, на рисунку 2.6 зображена тенденція розроблення додатку при гарно сконструйованій архітектурі та у разі нехтуванням нею.

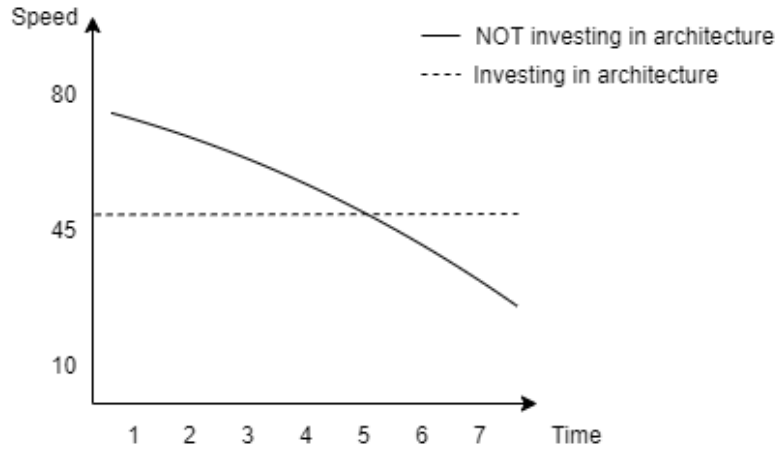

Рисунок 2.6 — Тенденція розроблення додатка

Ідея полягає в тому, щоб помістити належну відповідальність у відповідний рівень системи: ядро, абстракцію або рівень представлення. Кожен рівень розглядається незалежно один від одного. Цей поділ системи також диктує правила взаємодії між рівнями. Наприклад, рівень представлення може спілкуватися тільки через рівень абстракції, тощо. Схема даної архітектури зображена на рисунку 2.7.

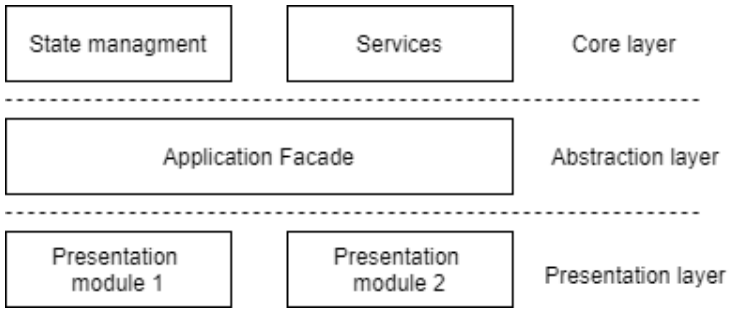

Рисунок 2.7 — Архітектура клієнтського додатку

Рівень представлення — це місце, де знаходяться Angular компоненти. Єдині обов’язки цього шару — представляти і делегувати. Цей рівень представляє користувацький інтерфейс і делегує дії користувача базовому рівню через рівень абстракції. Він знає, що відображати і що робити, але не знає, як слід обробляти взаємодії з користувачем.

Рівень абстракції — відокремлює рівень представлення від основного рівня, а також має свої власні певні обов’язки. Рівень абстракції не місце для бізнес-логіки. Основна його ідея полягає в тому, щоб зв’язати рівень представлення з бізнес-логікою.

Однією з важливих аспектів рівня абстракції є стратегія синхронізації. Незалежно від обраного рішення по управлінню станом системи, можна реалізувати оновлення користувацького інтерфейсу або оптимістично, або песимістично. Наприклад, функціонал даної веб системи дозволяє залишити відгук. Ця колекція отримана з сервера і відображена на екрані.

1. Оптимістичне оновлення спочатку змінює стан призначеного для користувача інтерфейсу і намагається оновити стан серверу. Користувач не бачить ніяких затримок через затримку в мережі. Якщо оновлення внутрішнього інтерфейсу завершується невдало, зміну призначеного для користувача інтерфейсу необхідно відкотити до минулої версії.

2. Песимістичне оновлення змінює стан серверу першим і тільки в разі його успішного оновлення, змінюється користувацький інтерфейс. Як правило, через затримку в мережі необхідно показувати якийсь лічильник або смугу завантаження під час виконання запиту на сервер.

Інколи дані, що надходять з сервера призначені тільки для читання, якими нема потреби маніпулювати і просто передаються компонентам через рівень абстракції. В цьому випадку можна застосувати хешування даних за допомогою фасаду. Рівень абстракції грає важливу роль в даній багаторівневій архітектурі. В ній чітко визначені обов’язки, які допомагають краще зрозуміти і мислити про систему. В залежності від конкретного випадку можна створити один фасад для кожного модуля Angular або для кожного об’єкта.

Основний рівень — це рівень в якому реалізована бізнес-логіка застосунку. Всі маніпуляції з даними і зв'язок з зовнішнім світом відбувається тут. На основному рівні також реалізуються HTTP запити в формі класів постачальників. Сервіси API несуть тільки одну відповідальність — це зв'язок з кінцевими точками API і нічого більше, тому необхідно уникати хешування, логіку і маніпулювання даними. На цьому рівні також розміщуються валідатори або більш складні користувацькі кейси, які потребують маніпулювання багатьма фрагментами стану користувацького інтерфейсу.

Отже, як результат, було розглянуто та спроектовано рівні абстракцій в інтелектуальній веб системі. Кожен рівень має свою чітку визначену границю і обов’язки. Також було визначено строгі правила взаємодії між рівнями. Все це допомагає краще розуміти і мислити про систему з плином часу, оскільки вона становиться все більш складною. Ці принципи, при правильному застосуванні можуть допомогти підтримувати швидкість стійкого розвитку с плином часу і дозволить з легкістю надавати нові функції.

2.5. Архітектура серверної частини

У традиційних архітектурах однакова модель даних використовується для запиту та оновлення бази даних. Це просто і добре працює для основних операцій із CRUD. Однак у більш складних програмах такий підхід може стати непростим.

Наприклад, на стороні читання програма може виконувати багато різних запитів, повертаючи об’єкти передачі даних (DTO) з різними формами. З боку запису, може реалізувати складну перевірку валідацію та бізнес-логіку. Як результат, можна створити складну модель, яка робить занадто багато. Навантаження для читання та запису часто асиметричне, з дуже різними вимогами до продуктивності та масштабу. Основні проблеми:

1. Часто виникає невідповідність між представленнями даних для читання і запису даних, таких як додаткові стовпці або властивості, які необхідно правильно оновити, навіть якщо вони не потрібні в рамках операції.

2. Конфлікт даних може виникати, коли операції виконуються паралельно на одному і тому ж наборі даних.

3. Традиційний підхід може негативно вплинути на продуктивність через навантаження на сховище даних та рівень доступу до даних та складність запитів, необхідних для отримання інформації.

Схема даної архітектури зображена на рисунку 2.8.

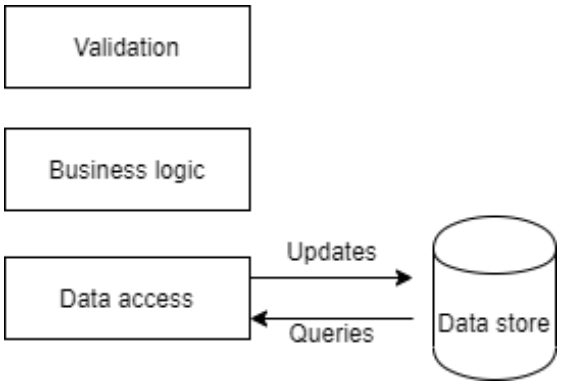

Рисунок 2.8 — Традиційна архітектура

Для вирішення цієї проблеми було використано шаблон CQRS. Він розділяє операції зчитування та запису в окремі моделі, використовуючи команди та запити для читання даних.

1. Команди повинні бути на основі завдань, а не орієнтовані на дані. Команди можуть бути розміщені в черзі для асинхронної обробки, а не оброблятися синхронно.

2. Запити ніколи не змінюють базу даних. Запит повертає DTO, який не інкапсулює будь-які знання домену.

Потім моделі можна ізолювати, як показано на наступному рисунку 2.9:

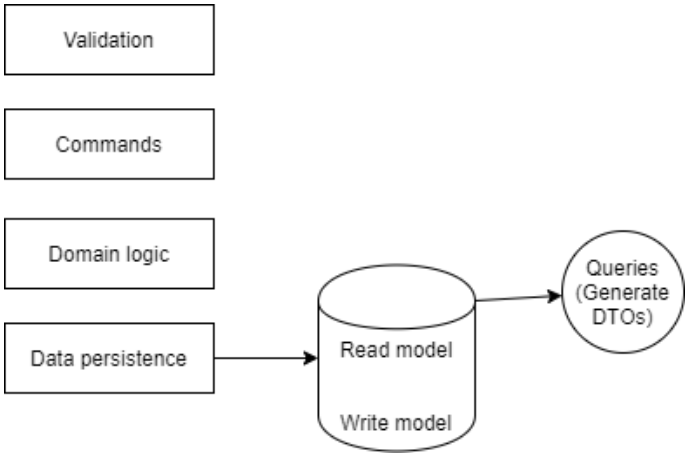

Рисунок 2.9 — CQRS архітектура

Також необхідно створити обробники та показати місце, куди слід відправити цей запит чи команду, щоб перейти до потрібного обробника для виконання своєї роботи. Всі роботи будуть виконані з MediatR.

3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ РЕАЛІЗАЦІЇ ЗАДАЧІ

В розділі наведено алгоритмічне та математичне забезпечення розв’язання задач:

1. Об’єднання деякої інформації про об’єкт, отриманої відповідно до моделі руху і вимірюваними інструментами на основі фільтрів Калмана з описом використання нормального розподілу (3.1 – 3.3).

2. Присвоєння центроїдів виявлених транспортних засобів до відповідних траєкторій з найменшими загальними витратами за допомогою Угорського алгоритму.

3.1. Нормальний розподіл

Фільтр Калмана працює при умові нормального розподілу. Нормальний розподіл є найважливішим розподілом ймовірностей у статистиці, оскільки він відповідає багатьом природним явищам. Він також відомий як розподіл Гауса. Нормальний розподіл — це функція ймовірності, яка описує розподіл значень змінної. Це симетричний розподіл, де більшість спостережень скупчуються навколо центрального піку і ймовірності значень, що знаходяться далі від середнього, звужуються рівномірно в обох напрямках. Екстремальні значення в обох хвостах розподілу однаково малоймовірні.

Параметри нормального розподілу повністю визначають його форму та ймовірності. Нормальний розподіл має два параметри, математичне очікування та стандартне відхилення. Нормальний розподіл не має лише однієї форми. Форма змінюється на основі значень параметрів, а саме:

1. Математичне очікування(середнє значення) μ — центральна тенденція розподілу. Він визначає місце піку для нормальних розподілів. Більшість значень скупчуються навколо середнього.

2. Стандартне відхилення (σ) — це міра розсіювання значень випадкової величини щодо математичного очікування. Він визначає ширину нормального розподілу. Стандартне відхилення — це квадратний корінь дисперсії (σ^2), заданий формулою (3.1):

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3.1}$$

Прогнозоване значення зосереджено навколо середнього значення, а ширина Гауса позначає невизначеність. Чим більша ширина Гауса, тим більшу невизначеність визначає.

3.2. Фільтр Калмана

Кожна фізична система має процес. Наприклад, транспортний засіб, який рухається з певною швидкістю, проходить деяку дистанцію за фіксовану кількість часу, і його швидкість змінюється залежно від його прискорення. Дана поведінка описується за допомогою добре відомих рівнянь Ньютона (3.2) та (3.3):

$$v = at; \tag{3.2}$$

$$x = \frac{1}{2}at^2 + v_0 t + x_0 \tag{3.3}$$

За відсутності даних про швидкість та прискорення, типова проблема відстеження автомобіля дозволила б вам обчислити пройдену відстань з урахуванням постійної швидкості або прискорення. Проте жодна машина не подорожує ідеальною дорогою. Є удари, пориви вітру та пагорби, які піднімають та

знижують швидкість. Ідеально моделювати систему неможливо, за винятком самих тривіальних проблем. Тому необхідно зробити спрощення. У будь-який час t справжній стан (положення нашого автомобіля) — це передбачуване значення недосконалої моделі плюс якийсь невідомий шум процесу (3.4):

$$x(t) = x_{pred}(t) + noise(t) \tag{3.4}$$

Фільтр Калмана [5] — алгоритм, який може передбачати майбутні позиції на основі поточної позиції. Алгоритм складається з двох повторюваних фаз: фаза екстраполяції та фаза корекції. Фільтр Калмана використовується для кожного знайденого транспортного засобу, викликаються функції прогнозування та оновлення. Ці функції реалізують математику фільтрів Калмана, що складаються з формул для визначення середнього значення положення та коваріації. Отже, в даній системі середнє значення це координати обмежувального вікна. Коваріація — це невизначеність щодо цього обмежувального поля, що має ці координати, адже центр області детектування відрізняється від істинного центру транспортного засобу, тобто можлива деяка помилка вимірювання.

Середнє значення (x) — це вектор положення транспортного засобу, що складається з координат центру обмежувального вікна (cx, cy) та швидкостей (vx, vy), представляється у вигляді (3.5):

$$x = [cx, cy, vx, vy] \tag{3.5}$$

Коли ініціалізується цей параметр, то швидкості заповнюються нулями, оскільки надалі вони будуть оцінені фільтром Калмана.

Коваріація (P) — матриця невизначеності в оцінці. Отже, все що необхідно оцінити стан та невизначеність. Для роботи фільтра Калмана існує два кроки: прогнозування та оновлення. Прогнозування прогнозує майбутні позиції, оновлення виправить їх та покращить спосіб, яким прогнозується, змінивши невизначеність. З часом фільтр

Калмана стає все кращим і кращим. Схема роботи фільтра представлена на рисунку 3.1.

Рисунок 3.1 — Схема роботи фільтра Калмана

Під час роботи першої фази відбувається передбачення (екстраполяція) значення змінних стану на основі оцінки стану попереднього кроку, а також їх невизначеності. Дану оцінку часто також називають апріорної через те, що вона дається до виконання будь-яких вимірювань і ґрунтується лише на математичній моделі.

Фаза прогнозування — це матричне множення, що обчислює положення обмежувального поля в момент часу t, виходячи з його положення в момент часу $t-1$. У класі KalmanFilter існує функція predict(), що реалізує математичні обчислення у вигляді (3.6) та (3.7).

$$x' = Fx + u \tag{3.6}$$

$$P' = FPF^T + Q \tag{3.7}$$

F — це матриця переходу системи з моменту часу $t-1$ в t. Вміст даної матриці особливо важливий, тому що, коли множиться x на F як показано за формулою (3.8), то змінюється x і отримується новий x, який називається x'.

$$\begin{pmatrix} cx \\ cy \\ vx \\ vy \end{pmatrix} = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} cx \\ cy \\ vx \\ vy \end{pmatrix} \tag{3.8}$$

Матриця F [4x4] містить значення часу: dt — різниця між поточним кадром та колишньою міткою кадру. Зрештою виходить $cx' = cx + dt * vx$ для першого рядка, $cy' = cy + dt * vy$ для другого і так далі.

Q — матриця шуму, що означає скільки впевненості надається системі. Її визначення дуже важливе і може змінити багато речей. Q буде доданий до коваріації, а потім визначить глобальну невизначеність. Можна поставити дуже малі значення (0,01) і змінити їх з часом.

На основі цих матриць і вимірювання можна передбачити прогноз, який дасть x' та P'. Це можна використовувати для прогнозування майбутніх чи фактичних позицій.

Друга фаза відповідає за уточнення результату екстраполяції за допомогою відповідних вимірювань, можливо отриманих з деякою погрішністю. Дана оцінка називається апостеріорної. Ці фази чергуються, тобто передбачення відбувається щодо результатів коригування з минулої ітерації, а коригування уточнює результат фази екстраполяції. Однак в деяких випадках фаза корекції може бути пропущена і передбачення буде відбуватися на основі не уточненої оцінки, що дозволяє вирішувати задачу оклюзій. Така ситуація може виникнути якщо з якої-небудь причини не має інформації з вимірювальних датчиків на даному етапі. В даному випаду вимірювальний датчик це метод виявлення YOLOv3.

Етап оновлення — це етап корекції, що включає математичні обчислення у вигляді (3.9 — 3.13). Він включає нове вимірювання z і допомагає покращити фільтр.

$$y = z - Hx' \tag{3.9}$$

$$S = HP'H^T + R \tag{3.10}$$

$$K = P'H^T S^{-1} \tag{3.11}$$

$$x = x' + Ky \tag{3.12}$$

$$P = (I - KH)P' \tag{3.13}$$

Процес оновлення починається з вимірювання помилки між вимірюванням

(z — значення від детектора YOLOv3) та прогнозованим середнім значенням.
z — вимірювання в момент часу t. Тут не вводяться швидкості, оскільки вони не вимірюються, а просто вимірюються значення у вигляді (3.14).

$$z = [cx, cy] \tag{3.14}$$

H — це матриця переходу стану. За допомогою H можна відкинути інформацію від змінної стану, яка не потрібна. Технічно H виконує ту саму роботу, що і F в передбачуваному кроці та має вигляд (3.15).

$$H = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \tag{3.15}$$

R — вимірювальний шум, тобто шум від датчика. Тут потрібно визначити шум алгоритму YOLO в перерахунку на пікселі. Шум визначається довільно, тому припустимо, що помилка вимірювання становить приблизно 2 пікселі, тому має вигляд (3.16).

$$R = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \tag{3.16}$$

В результаті отримано цикл прогнозування та оновлення. Фільтр можна використовувати для прогнозування в момент часу $t + 1$ (прогнозування без оновлення) від часу t. Для цього воно повинно бути досить хорошим і мати низьку невизначеність.

3.3. Сігма-точковий фільтр Калмана

Фільтр Калмана завжди працює з розподілом Гауса та лінійними функціями. Більшість проблем у реальному світі пов'язані з нелінійними функціями. Якщо розглядати Гауса з лінійною функцією, то вихід також є гаусівский. В той час нелінійні функції призводять до негаусівських розподілів.

Хоча KF і UKF базуються на одній і тій же схемі, проте перетворення “unscented transform” UT[6, 7], робить сігма-точковий фільтр застосовний до нелінійних систем. UKF може вирішувати всі сценарії на які здатний KF, а також бути призначений для значно більшого спектру, ніж KF, а саме для багатьох нелінійних застосувань. Крім того, UKF часто може забезпечити кращу точність оцінки, ніж KF. Іноді, навіть, якщо шум не поширюється по Гаусу, UKF все ще може працювати, тоді як KF може вийти з ладу, забезпечити неоптимальні результати оцінки змінних стану.

При використанні Extended Kalman Filter (EKF) необхідно на кожному кроці ітерацій обчислювати Якобіан через що сильно зростає обчислювальна. EKF реалізує в собі прошарок лінеаризації нелінійної динамічної системи. У цьому полягає головна причина, по якій EKF може виявитися неефективним. Мала похибка задання параметрів математичної моделі буде приводити до великих обчислювальних погрішностей. В результаті алгоритм втратить робастність (стійкість до похибок).

Unscented Kalman filter (UKF) використовує інший підхід, а саме “unscented transform” (UT). Даний підхід має на увазі вибір мінімального набору сігма-точок для даних математичного очікування і коваріації шуканого вектору з накладеними на нього випадковими шумами. За сігма-точками будуються нелінійні функції прогнозу, які потім використовуються для обчислення матриці кросс-коваріації (ядро алгоритму калмановій фільтрації). Отже, головна проблема полягає в нелінійності математичної моделі. За допомогою UT можна без лінеаризації обчислити прогноз фазового вектору і матрицю кроссковаріації. Підставивши їх

значення в стандартні вирази для фази корекції прогнозу FK можна отримати оцінки фазового вектору. Загальна складність UKF виходить не більше, ніж у EKF.

Сігма-точки також мають ваги, тож це зважені сігма-точки. Кількість сігма-точок залежить від розмірності системи. Загальна формула — $2N + 1$, де N позначає розмірність.

Обчислюються сігма-точки за наступним формулами (3.17 — 3.19):

$$\chi^{[0]} = \mu \tag{3.17}$$

$$\chi^{[i]} = \mu + \left(\sqrt{(n + \lambda)\Sigma}\right)_i \; for\; i = 1, \dots, n \tag{3.18}$$

$$\chi^{[i]} = \mu - \left(\sqrt{(n + \lambda)\Sigma}\right)_{i-n} \; for\; i = n + 1, \dots, 2n \tag{3.19}$$

де χ — матриця сігма точок, де кожен стовпчик позначає набір сігма точок, μ — середнє значення по Гаусу, n — розмірність системи, λ — коефіцієнт масштабування, який говорить про те, наскільки далеко від середнього слід вибирати сігма-точки. Математичне дослідження пропонує оптимальне значення λ рівне $3 - n$. Очевидно, що одна з сігма-точок — це середнє значення, а решта обчислюється, виходячи з вищенаведених рівнянь.

Отже, якщо працювати в 2 вимірному просторі, то розмір матриці χ буде 2 x 5

Вага для сігма-точок обчислюється за наступними формулами (3.20) та (3.21):

$$w^{[0]} = \frac{\lambda}{n + \lambda} \tag{3.20}$$

$$w^{[i]} = \frac{\lambda}{2(n + \lambda)} \; for\; i = 1, \dots, n2 \tag{3.21}$$

Обчислення ваги середнього значення має інше рівняння, ніж решта сігма-точок. Необхідна умова, що сума всіх ваг дорівнює 1.

Обчислення середнього значення та коваріації за формулами (3.22) та (3.23):

$$\mu' = \sum_{i=0}^{2n} w^{[i]} g(\chi^{[i]}) \tag{3.22}$$

$$\Sigma' = \sum_{i=0}^{2n} w^{[i]} (g(\chi^{[i]}) - \mu')(g(\chi^{[i]}) - \mu')^T \tag{3.23}$$

де μ' — прогнозоване середнє значення, Σ' — прогнозоване значення коваріації, g — нелінійна функція.

UKF працює подібно до фільтра Калмана та складається з двох повторюваних фаз: фаза прогнозування та фаза корекції.

Фаза прогнозування:

1. Обчислення сігма-точок використовуючи рівняння (3.17 — 3.19).
2. Обчислення ваги для сігма-точок використовуючи рівняння (3.20) та (3.21).
3. Трансформація сігма точок та обчислення нових значень математичного очікування та коваріації.

Щоразу, після прогнозування, невизначеність збільшується на деяку величину, тому що стан стає трохи невизначеними. Отже, потрібно враховувати шум процесу, тому математичне очікування залишиться незмінним, а коваріація обчислюється за формулою (3.24).

$$\Sigma' = \sum_{i=0}^{2n} w^{[i]} (g(\chi^{[i]}) - \mu')(g(\chi^{[i]}) - \mu')^T + R_t \tag{3.24}$$

Фаза оновлення:

Фільтри Калмана виконують оновлення в просторі вимірювання. Таким чином, необхідно перетворити сігма-точки попередньої оцінки в вимірювання за допомогою вимірювальної функції h(x) (3.25):

$$Z = h(\chi) \tag{3.25}$$

Далі обчислюється середнє значення та коваріантність цих точок, використовуючи UT. Середнє значення та коваріація сігма точок в просторі вимірювання відповідно розраховуються за формулами (3.26) та (3.27).

$$\hat{z} = \sum_{i=0}^{2n} w^{[i]} Z^{[i]} \tag{3.26}$$

$$\mathrm{S} = \sum_{i=0}^{2n} w^{[i]} (Z^{[i]} - \hat{z}\,)(Z^{[i]} - \hat{z})^T + Q \tag{3.27}$$

де $\hat{z}$ — середнє значення в просторі вимірювання, S — коваріація сігма точок в просторі вимірювання.

Далі обчислюється різниця між фактичним вимірюванням та прогнозуванням (3.28):

$$y = z - \hat{z} \tag{3.28}$$

Для обчислення коефіцієнта посилення Калмана спочатку обчислюється матриця крос-коваріації між точками стану і вимірювального простору, яка визначається за формулою (3.29):

$$T = \sum_{i=0}^{2n} w^{[i]} \left(\chi^{[i]} - \mu'\right)\left(Z^{[i]} - \hat{z}\right)^T \tag{3.29}$$

А потім коефіцієнт посилення Калмана визначається за формулою (3.30):

$$K = T.S^{-1} \tag{3.30}$$

Коефіцієнт посилення Калмана – це просте співвідношення, яке обчислюється за формулою (3.31):

$$K \approx \frac{T}{S} \tag{3.31}$$

Отже, в результаті за допомогою UT обчислюємо без лінеаризації прогноз фазового вектору і матрицю кросс-коваріації. Підставивши їх значення в стандартні виразі для фази корекції (3.32) та(3.33) отримаємо оцінку фазового вектору.

$$\mu = \mu' + K(z - \hat{z}) \tag{3.32}$$

$$\Sigma = (I - KT)\Sigma' \tag{3.33}$$

Надалі фази прогнозування та фази корекції повторюються за наведеною вище математичною моделлю, що складається з формул для визначення середнього значення та коваріації.

3.4. Угорський алгоритм

Угорський алгоритм [8] — алгоритм, що допомагає вирішувати задачу про призначення. Для даної інтелектуальної системи необхідно присвоїти n-виявлених транспортних засобів до існуючих траєкторій руху оптимальним чином. Математична модель даної задачі представлена наступним чином (3.34 – 3.37):

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij} \to (min); \tag{3.34}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = \overline{1,n}; \tag{3.35}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = \overline{1,n}; \tag{3.36}$$

$$x_j \geq 0, \quad j = \overline{1,n} \tag{3.37}$$

де c_{ij} — загальні витрати для і-го транспортного засобу та j-го треку, x_{ij} — приймає значення 0, якщо транспортний засіб не призначається для траєкторії та 1, якщо навпаки.

Даний алгоритм включає наступні кроки:

1. Необхідно виконати редукцію по рядкам, тобто знайти мінімальний елемент в кожному рядку і відняти його від кожного елемента відповідно.

2. Необхідно в отриманій матриці необхідно провести редукцію за стовпцями.

3. Наступним етапом, необхідно перевірити на наявність оптимального рішення, тобто необхідно, щоб в кожному стовпці та в кожному рядку був тільки один обраний нуль. В разі, якщо це неможливо зробити і редукція не дає потрібний результат, то необхідно перейти до наступного кроку.

4. Необхідно викреслити рядки і стовпці, які містять нульові елементи. Кількість викреслювань має бути мінімальним, тому починати викреслювати потрібно з тих стовпців та рядків, де максимальна кількість нульових елементів.

5. Серед решти елементів шукаємо мінімальний, віднімаємо його з усіх невикреслених елементів і додаємо до елементів, які розташовані на перетині викреслених рядків і стовпців. Якщо допустиме рішення отримано, то оптимальні призначення відповідають нульовим елементам. Завершити роботу. Інакше перейти до кроку під номером 4.

Повний процес Угорського алгоритму показаний на рисунку 3.2:

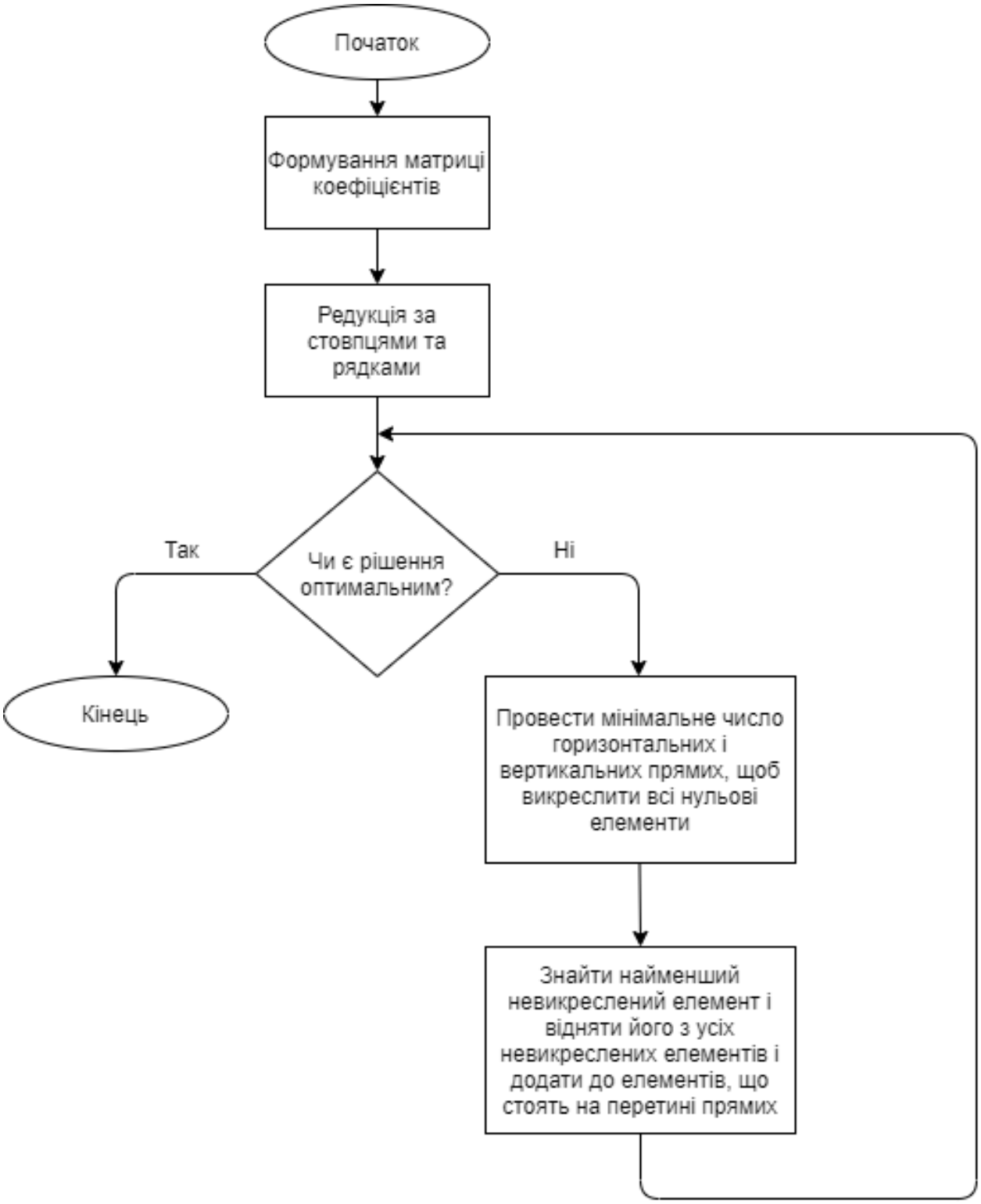

Рисунок 3.2 — Схема угорського алгоритму.

Отже, завдяки Угорському алгоритму було вирішено задачу присвоєння центроїдів виявлених транспортних засобів до відповідних траєкторій з найменшими загальними витратами.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1. Знаходження транспортних засобів

У даній роботі для виявлення транспортних засобів використовується метод YOLOv3, оскільки це дуже швидка, проста, але водночас ефективна CNN для виявлення об’єктів. Для реалізації було використано бібліотеку OpenCV, що має вбудовану підтримку Darknet Architecture. Darknet Architecture — це заздалегідь підготовлена модель для класифікації 80 різних класів.

YOLO [9] — алгоритм виявлення об'єктів. Остання версія YOLOv3 є більш потужнішою, ніж попередники YOLO та YOLOv2, і швидшою, ніж алгоритми, такі як R-CNN, і більш точна. Основна причина — це одноетапний детектор, що сканує зображення (або кадр) лише один раз, щоб зробити передбачення порівняно з іншими алгоритмами, які потребують багаторазового сканування. Саме завдяки одному етапу, YOLO є дуже швидкою моделлю, що дозволяє виявляти об’єкти навіть в режимі реального часу.

Найбільш помітною особливістю YOLOv3 є те, що він здійснює виявлення в трьох різних масштабах. YOLO ділить вхідне зображення на сітку SxS, де S залежить від масштабу. Вихідні матриці матимуть різні розмірності в залежності від розміру осередків 8, 16 та 32 пікселів. Виявлення на різних шарах допомагає вирішити питання виявлення дрібних предметів. Клітинки сітки розмірності 32 відповідають за виявлення великих об’єктів, тоді як розмір 8 виявляє більш дрібні об'єкти.

Кожна з клітинок сітки прогнозує B обмежувальних поля(в алгоритмі YOLOv3 кількість обмежувальних полів дорівнює 3) і для кожного поля модель виводить показник достовірності C. Цей показник відображає ймовірність наскільки модель підтверджує, що комірка містить певний об’єкт. Додатково до показника

достовірності модель видає 4 числа (x, y, w, h) для відображення місця розташування та розмірів передбачуваного обмежувального вікна. Координати (x, y) являють собою центр поля відносно меж комірки сітки. Ширина та висота (w, h) прогнозуються відносно всього зображення. Кожен осередок сітки представлений у вигляді вектору, розмір якого вираховується за формулою: $B\ x\ (5\ +\ C)$.

Об’єкт може знаходитись у більш ніж одній комірці, тому модель YOLO може виявити його більше одного разу(у більше ніж одній комірці), проте дана проблема вирішується за допомогою Non Maximum Suppression [10]. Це клас алгоритмів для вибору однієї сутності (обмежувальних полів) з багатьох об'єктів, що перекриваються. Після того, як детектор видає велику кількість обмежувальних вікон, необхідно вибрати найкращі. NMS — це найбільш часто використовуваний алгоритм для виконання цього завдання.

Отже, для знаходження транспортних засобів був використаний YOLOv3 та бібліотека комп’ютерного зору OpenCV. Спочатку необхідно завантажити 3 основних файли:

1. yolo.cfg — конфігураційний файл
2. yolo.weights — попередньо натреновані ваги
3. coco.name — 80 назв класів

Після цього необхідно завантажити алгоритм YOLOv3 за допомогою cv2.dnn.readNet, передаючи ваги та файл yolo.cfg. Спочатку потрібно зробити певне перетворення з вхідного зображення, яке в основному спрямоване на виявлення точок та областей зображення, які відрізняються за властивостями, такими як яскравість або колір порівняно з оточуючими. Однією з головних причин є надання додаткової інформації про регіони. Дане перетворення використовується для отримання регіонів, що представляють інтерес для подальшої обробки. Ці регіони можуть сигналізувати про наявність об'єктів або частин об'єктів у області зображення із застосуванням для розпізнавання об'єктів. Виявлення згустку зазвичай проводиться після виявлення кольору та зменшення шуму, щоб нарешті

знайти потрібний об'єкт із зображення. Тому необхідно використати cv2.dnn.blobFromImage та передати кілька параметрів.

На рисунку 4.1 показані 3 різні згустки. Незважаючи на те, що між ними не спостерігається велика різниця, але це те, що надалі використовує алгоритм YOLO.

Рисунок 4.1 — 3 згустки, що передаються далі алгоритму

Далі необхідно передати цей згусток в мережу, а потім переслати на вихідні шари. Після чого всі транспортні засоби будуть знайдені. Щоб усунути випадки, коли кілька разів один і той же об’єкт може бути виявлений, необхідно використати функцію Non Max Suppression (NMS), що дозволяє зберігати лише найбільш достовірні з усіх обмежувальних полів. Далі необхідно написати функції і класи для отримання відеоряду з камери, виведення зображення на екран і відтворення передбачених обмежувальних полів. На рисунку 4.2 наведено результат роботи методу YOLOv3.

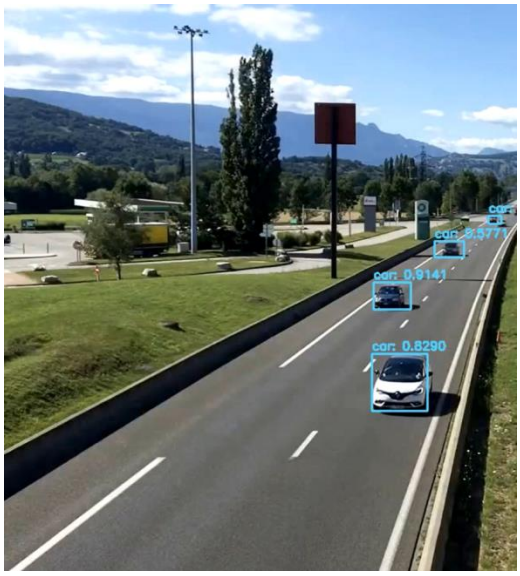

Рисунок 4.2 — Зображення оброблене за допомогою YOLOv3-416

Даний метод виявив всі транспортні засоби та розпізнав дрібні транспортні засоби, що знаходяться дуже далеко, саме завдяки третьому вихідному шару детектора.

4.2. Відстеження транспортних засобів

Відстеження декількох рухомих об’єктів для спостереження та моніторингу подій є дуже складним завданням, а саме через шум камери, стан освітлення, оклюзію об'єкта тощо. Відстеження рухомих об'єктів на основі результатів виявлення об'єктів спрямоване на оцінку оптимального шляху рухомих об'єктів для подальшого аналізу подій. Якщо є лише один об'єкт, що рухається, проблема відстеження є тривіальною. Однак якщо є кілька рухомих об’єктів, що іноді перекриваються одна з одною, проблема відстеження стає складнішою. Коли декілька об’єктів відходять один від одного після оклюзії, то потрібно знайти та узгодити правильне місце для кожного об’єкта, що дозволяє постійно виявляти та відстежувати декілька об’єктів.

Фільтр Калмана (KF) широко використовується для відстеження рухомих об’єктів, за допомогою яких можна оцінити швидкість і навіть прискорення об'єкта за допомогою вимірювання його локацій. Однак точність KF залежить від припущення лінійного руху для будь-якого об'єкта, який слід відслідковувати. Якщо об'єкт робить різкі повороти, нелінійний рух не може бути належним чином оброблений за допомогою фільтра Калмана (через припущення лінійного руху).

Щоб вирішити нелінійну проблему відстеження, було застосовано сігма-точковий фільтр Калмана (UKF) [4, 5] для відстеження кількох рухомих об'єктів. Це хороший альтернативний метод для нелінійної оцінки руху. Метод UKF добре розроблений для вирішення нелінійних задач оцінки та одночасного гальмування впливу шуму.

4.2.1. Проектування фільтра

Розробка моделей прогнозування та оновлення є ключовим аспектом теорії фільтрації Калмана. У даних експериментах рух одного транспортного засобу в двовимірному (2-D) зображенні можна розглядати як поєднання рухів по осі x і y, і ці два компоненти руху можуть бути оброблені незалежно. Якщо припустити, що виявлено M рухомих об’єктів, для відстеження потрібно оцінити $2M$ рухові компоненти (тобто, M компоненти на осі x та M компоненти на осі y). Вектор вимірювання Y можна виразити у вигляді (4.1):

$$Y = [x_1, y_1, x_2, y_2, \dots, x_M, y_M]^T \tag{4.1}$$

де $\{(x_i, y_i) \mid i = 1, \dots, M\}$ представляють інформацію про місце розташування для і-го виявленого об'єкта в напрямку осі x та y - осі.

Для кожного руху необхідно використовувати власне рівняння для відображення динаміки, тому враховується швидкість та прискорення. Таким чином, змінна стану X визначається у вигляді (4.2):

$$X = [x_1, v_{x1}, a_{x1}, y_1, v_{y1}, a_{y1}, \dots, x_M, v_{xM}, y_M, v_{yM}]^T \tag{4.2}$$

де $\{(v_{xi}, v_{yi}) \mid i = 1, \dots, M\}$ і $\{(a_{xi}, a_{yi}) \mid i = 1, \dots, M\}$ представляють швидкість та прискорення для і-го транспортного засобу в напрямку осі x та y-осі. Таким чином, рухаюча модель для і-го об'єкта визначається за допомогою функції нелінійної системи, що представлена формулами 4.3 та 4.4:

$$x_i(n) = x_i(n-1) + v_{xi}(n-1)dt + \frac{1}{2}a_{xi}(n-1)dt^2 \tag{4.3}$$

$$y_i(n) = y_i(n-1) + v_{yi}(n-1)dt + \frac{1}{2}a_{yi}(n-1)dt^2 \tag{4.4}$$

Матриця переходу представлена у вигляді (4.5):

$$F_i = \begin{matrix} 1 & dt & dt^2/2 & 0 & 0 & 0 \\ 0 & 1 & dt & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & dt & dt^2/2 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} (i = 1, \dots, M) \tag{4.5}$$

Розглядаючи форму змінної Y, матриця вимірювання H визначається як (4.6):

$$H_i = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} (i = 1, \dots, M) \tag{4.6}$$

В результаті схема UKF, описана в рівняннях (3.17) — (3.33), може бути використана для оцінки змінної стану X з нелінійної системи (рівняння 4.3 та 4.4) для відстеження багатьох транспортних засобів. На підставі вищеописаного опису алгоритм відстеження за допомогою UKF можна узагальнити наступним чином (дивитися рисунок 4.3 для блок-схеми алгоритму):

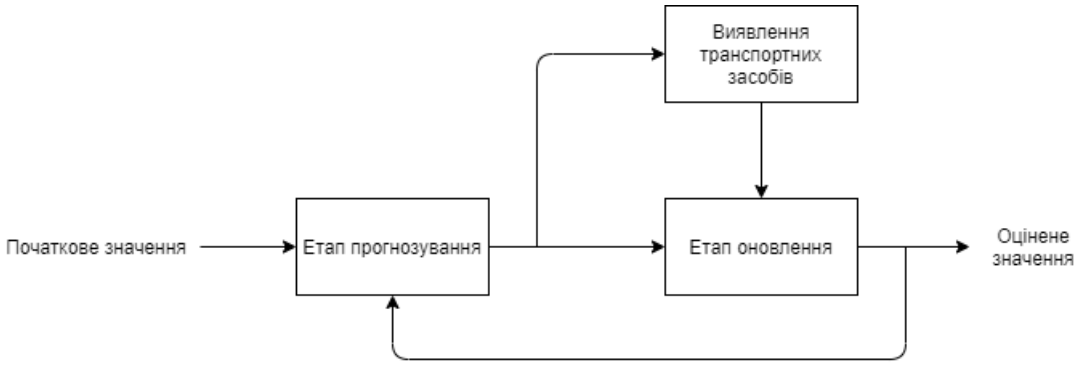

Рисунок 4.3 — Блок-схема запропонованого підходу відстеження руху за допомогою UKF

Крок 1. За результатами виявлення об'єктів можна дізнатися, скільки об’єктів виявлено і згодом їх потрібно відстежити. Перехідну матрицю F та вимірювальну матрицю H можна спочатку визначити (дивитися на 4.5 та 4.6). Після чого отримується вихідне значення вектору стану X.

Крок 2: Застосовано модель прогнозування UKF, щоб знайти наступні позиції та швидкості транспортних засобів.

Крок 3: Подаємо швидкість кожного транспортного засобу назад до алгоритму виявлення, щоб вирішити неоднозначність в виявленні об'єкта при виникненні оклюзії.

Крок 4: Оновлюємо вектор стану за допомогою значень вимірювання місця розташування кожного транспортного засобу.

Потім повертаємося до кроку 2, щоб розпочати наступну ітерацію.

4.2.2. Ідентифікація з Угорським алгоритмом

Для ідентифікації транспортних засобів був використаний Угорський алгоритм. Даний алгоритм пов’язує автомобілі від одного кадру до іншого на основі оцінок, а саме:

1. IOU(Intersection Over Union) — це означає, що якщо обмежувальне вікно перекриває попереднє, воно, ймовірно, те саме. Приклад зображено на рисунку 4.4.

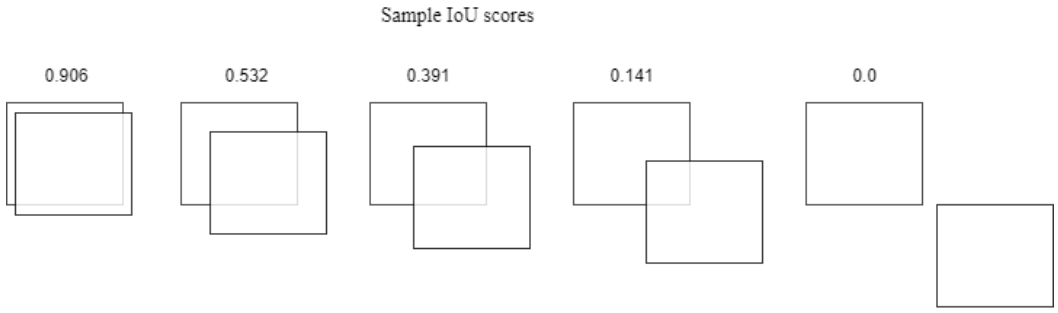

Рисунок 4.4 — Intersection Over Union

2. Shape Score — якщо форма або розмір не змінювались занадто сильно протягом двох послідовних кадрів, оцінка збільшується.

3. Евклідова відстань, або ж відстань між двома точками.

Отже, даний метод ідентифікації працює за наступною схемою зображеною на рисунку 4.5:

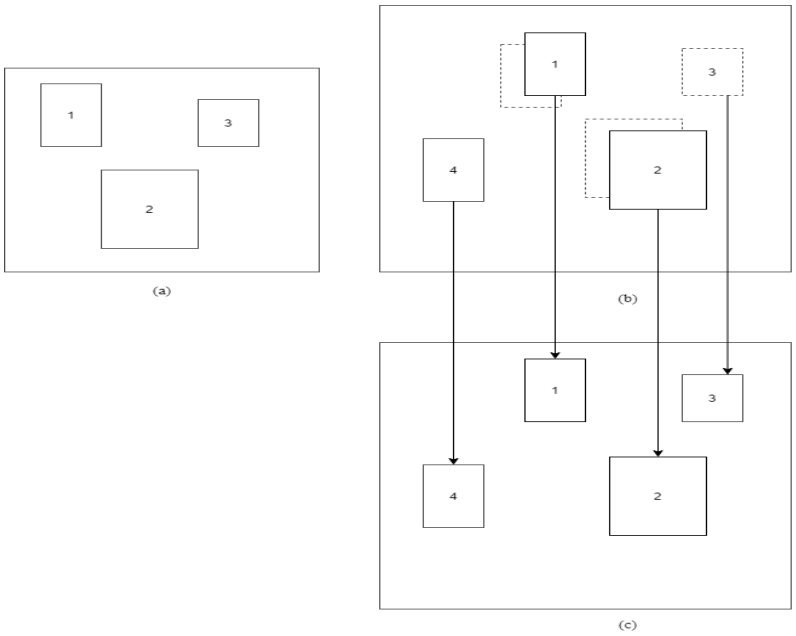

Рисунок 4.5 — Приклад IOU

У цьому прикладі від кадру a до кадру b відстежується два автомобіля (з id 1 і 2), додаючи одне нове виявлення (4) і зберігаємо виявлення (3), допускаючи, що воно хибне.

1. Існує два списки виявлених об’єктів від YOLO: список відстеження або треків $(t-1)$ та список виявлення (t).

2. Здійснюється перебір списку відстеження, виявлення та обчислюється IOU. Також існує функція витрат, що надає значення кожному балу. Результати обчислення IOU представлені в таблиці 4.1.

Таблиця 4.1. IOU у вигляді матриці

Detection/Tracking	Tracking 0	Tracking 1	Tracking 2
Detection A	IOU = 0	IOU = 0	IOU = 0
Detection B	IOU = 0.56	IOU = 0	IOU = 0
Detection C	IOU = 0	IOU = 0.77	IOU = 0

3. У деяких випадках перекриття обмежувальних рамок може мати два чи більше IOU для одного кандидата. У цьому випадку встановлюється максимальне значення IOU до 1, а всі інші — 0, як показано в таблиці 4.2.

Таблиця 4.2. Процес перетворення у вигляді матриці

Detection/Tracking	Tracking 0	Tracking 1	Tracking 2
Detection A	0	0	0
Detection B	1	0	0
Detection C	0	1	0

Дана матриця представляє відповідність між виявленням і відстеженням. Аналогічні операції необхідно виконати для обчислення Евклідової відстані. Після чого сумуються отримані бали. Евклідова відстань між точками x і y в n-вимірному просторі обчислюється за такою формулою (4.7):

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4.7}$$

Наступним кроком є виклик функції під назвою linear_assignment(), яка реалізує угорський алгоритм.

4.3.3. Загальна схема моніторингу руху транспортних засобів

Відстеження декількох об'єктів є складним завданням через змінну кількість транспортних засобів та взаємодію між ними у складних динамічних середовищах.

Після того, як YOLOv3 виявив деякий транспортний засіб та повернув координати, ширину і довжину обмежувальної рамки, алгоритм виявлення прагне визначити центроїди, що належать до одного і того ж транспортного засобу в послідовності кадрів та вивести траєкторії відповідних транспортних засобів.

Дану задачу було вирішено алгоритмом відповідності дводольного графа: вузли графа відповідають виявленим транспортним засобам та існуючим трекам, а зважені ребра графа позначаються вартістю виявлення відповідності.

Угорський алгоритм присвоює центроїди до відповідних траєкторій транспортних засобів з найменшими загальними витратами за допомогою формули 4.8, що є сумою відстані між центрами та траєкторіями, за умови, що один центр призначений виключно для однієї траєкторії.

$$S(D_i, D_j) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4.8)$$

де D_i та D_j представляють собою центроїди виявлення і відстеження відповідно, а S - евклідова відстань між двома центроїдами.

Якщо функція витрат S невелика, скоріше за два вузла мають відповідність. Порогове значення для функції витрат S становить 35. Це означає, що якщо S (Di, Dj) менше 35, то ймовірна відповідність між центроїдами виявлення та існуючим треком.

Відстеження декількох транспортних засобів може обробляти пропуски та помилкові виявлення. Кожен виявлений транспортний засіб передається до системи відстеження, і для нього створюється новий трек. Потім кожен новий трек оновлюється за допомогою фільтра Калмана в наступних кадрах, а інформація кожного треку зберігається, поки не буде видалена зі списку втрачених треків. Зберігається наступна інформація:

1). Ідентифікатор треку (ID): параметр ID ідентифікує кожен трек.

2). Обмежувальна рамка: ширина та довжина обмежувальної рамки на кожному кадрі.

3). Центроїд: зберігає центральні точки обмежувального вікна треку у кожному кадрі, а саме координати x та y.

4). Вік відстеження: показує кількість кадрів з моменту, коли трек був ініціалізований.

5). Видима довжина: це кількість кадрів, у яких був виявлений трек.

6). Невидима довжина: це кількість послідовних кадрів, у яких трек не було виявлено.

7). Коефіцієнт видимості: видима довжина / час відстеження

8). Параметри KF: зберігає параметри трекера KF

Вік відстеження, видима довжина, невидима довжина та коефіцієнт видимості (видима довжина / час відстеження) використовуються для обробки помилок та помилкових виявлень. Для запобігання коротких треків, викликаних помилковими виявленнями, видима довжина буде порівнюватися з порогом (10 кадрів). Якщо невидима довжина менша за цей поріг, його трек не зберігатиметься та не відображатимеся. Параметри встановлюються експериментально.

Також для усунення впливу помилок виявлення транспортного засобу на Угорський алгоритм використовується фільтр Калмана, який може передбачити розташування центроїда транспортного засобу у випадку його пропуску методом виявлення. Повний процес відстеження транспортних засобів показаний на рисунку 4.6 і включає наступні етапи:

1. Щойно знайдені транспортні засобі ідентифікуються і створюються треки з виявленими центроїдами. Коли відстані між виявленими центроїдами та існуючими траєкторіями перевищують заданий поріг, цей виявлений центроїд розглядається як новоявлений транспортний засіб.

2. Угорський алгоритм використовується для присвоєння решті виявлених центроїдів існуючим трекам шляхом мінімізації суми

3. Фільтр Кальмана використовується для прогнозування та виправлення виявлених центроїдів шляхом оцінки центроїда наступного кадру на основі заданого центроїда поточного кадру.

4. Видаляються існуючі траєкторії для яких не буде призначений відповідний транспортний засіб протягом п'яти кадрів (одна секунда), що свідчить про те, що транспортний засіб перемістився за зону дії камери.

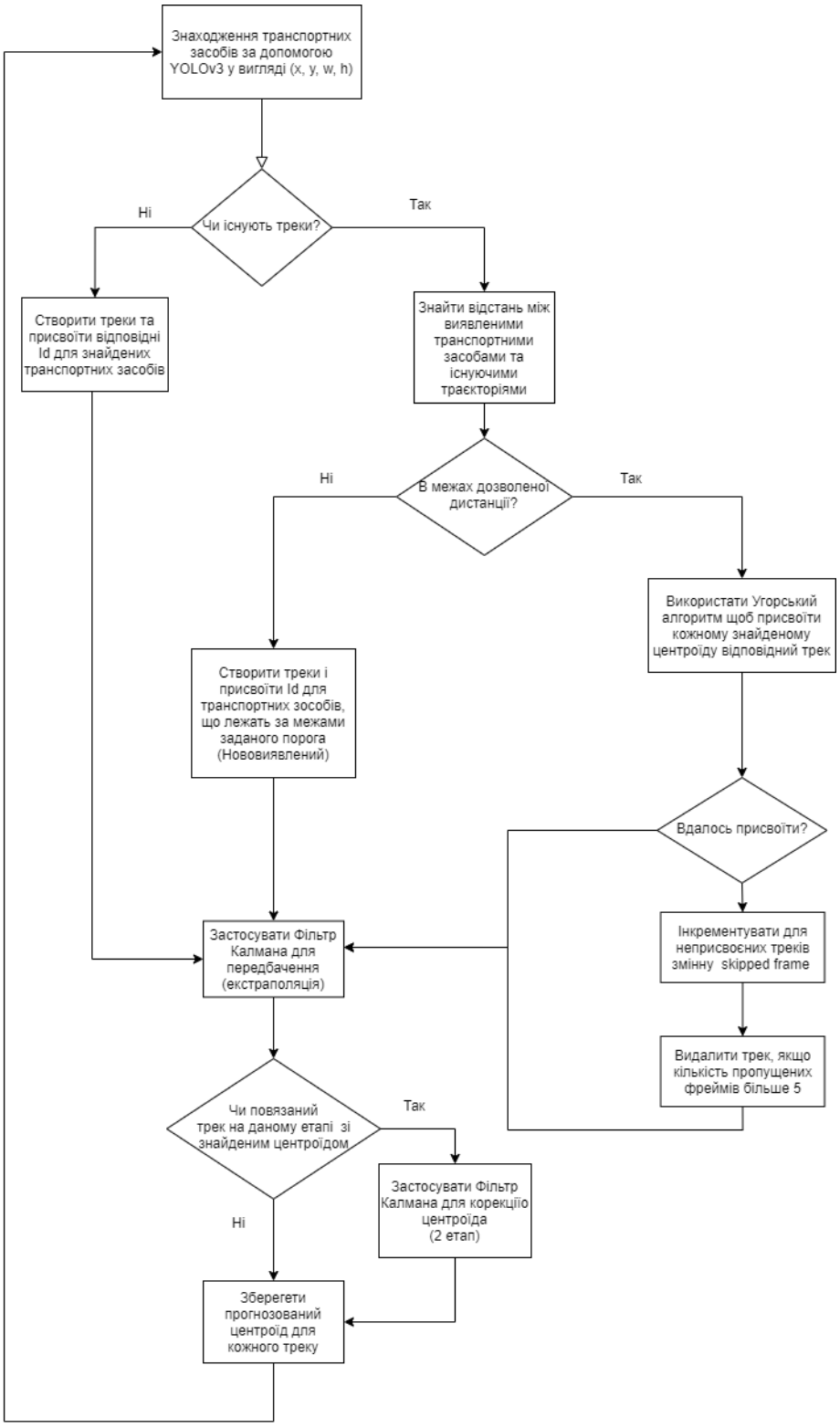

Рисунок 4.6 — Діаграма виконання модуля визначення, відстеження та ідентифікації транспортних засобів

Отже, було розглянуто схему моніторингу руху транспортних засобів, тобто алгоритм роботи методів виявлення, формування матриці коефіцієнтів, угорського алгоритму та етапи фільтрації.

4.3. Опис функціональності системи

Програмне забезпечення для визначення траєкторії руху транспортних засобів містить у собі головного актора – користувач системи;

Користувач системи має можливість авторизуватися та зареєструватися в разі відсутності облікового запису. Користувач має можливість редагувати свої персональні дані, а саме змінювати:

— пароль;

— пошту;

— фото профілю;

— ім’я, прізвище та ім’я по батькові;

Головним прецедентом користувача є можливість завантажувати відеофайл для обробки. Разом з цим, користувач може помічати необхідні ділянки на відеофайлі в разі необхідності для подальшого аналізу. Фактично, користувачу доступні наступні базові можливості:

— обирати розмір кісті для малювання;

— помічати необхідні зони;

— стирати попередні дії в процесі малювання;

Для взаємодії з проаналізованими відеофайлами користувачу надається можливість переглядати звіти про виконану роботу в персональному кабінеті, а також:

— змінювати назву звіту під власні вподобання;

— фільтрувати звіти за датою створення;

— пошук необхідних звітів за назвою;

Користувач має можливість залишати відгуки про інтелектуальну систему, що спрямований на вдосконалення системи та виправлення всіх помилок. Крім цього в разі необхідності є можливість відредагувати та видалити власні відгуки. Для запобігання спаму та некоректних відгуків існує інший актор системи, а саме адміністратор системи. Він має можливість видаляти повідомлення інших

користувачів та обмежувати їхні права, тобто блокувати можливість залишати відгуки та аналізувати відеофайли.

На рисунку 4.7 представлена діаграма прецедентів, яка описує функції та дії актора у системі.

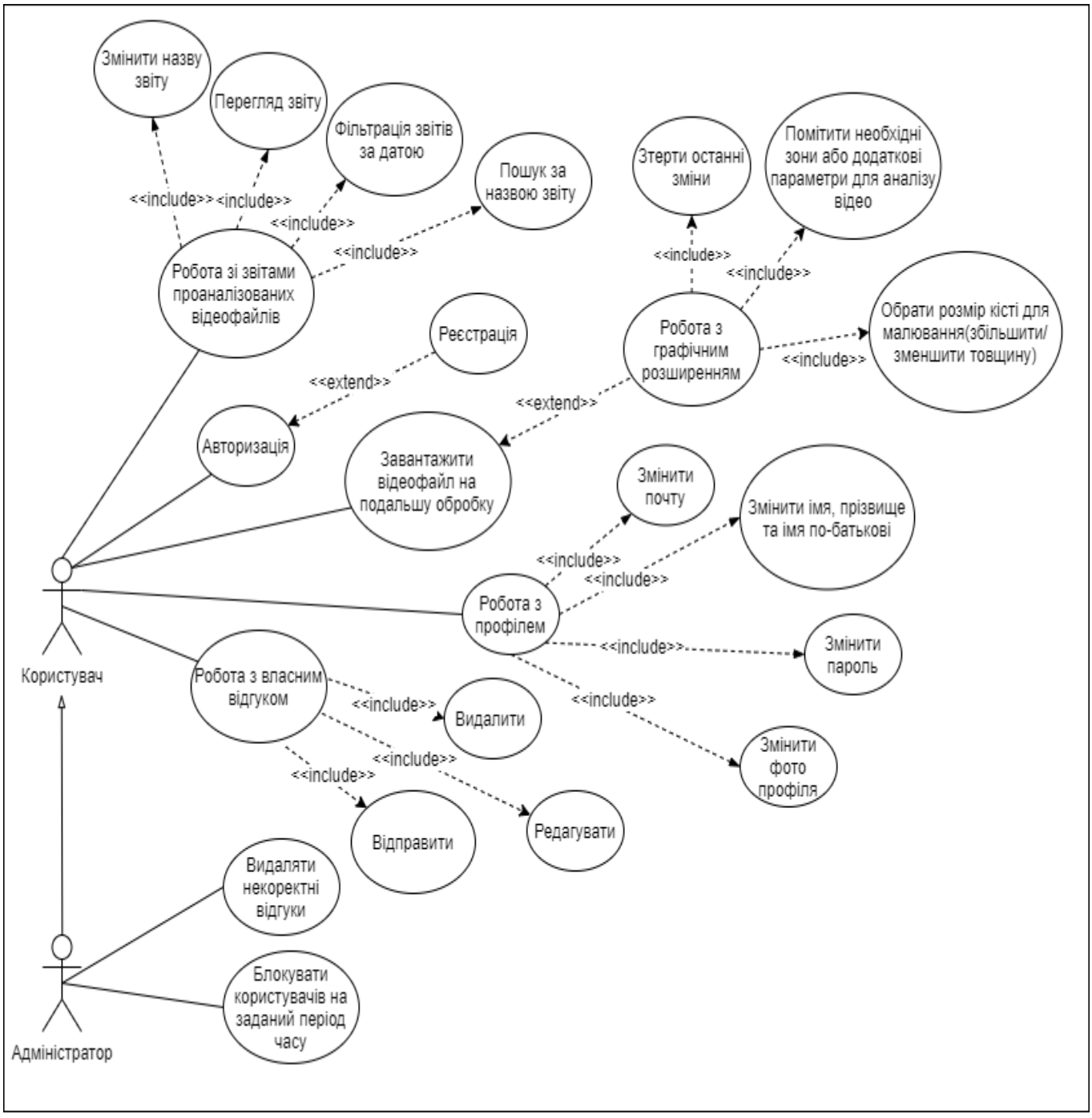

Рисунок 4.7 — Діаграма прецедентів

Більш детальний опис взаємодії користувача з системою розглянуто в розділі “МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ”.

4.4. Концептуальна модель бази даних

Концептуальна модель бази даних мікросервісу IdentityService приведена на рисунку 4.8.

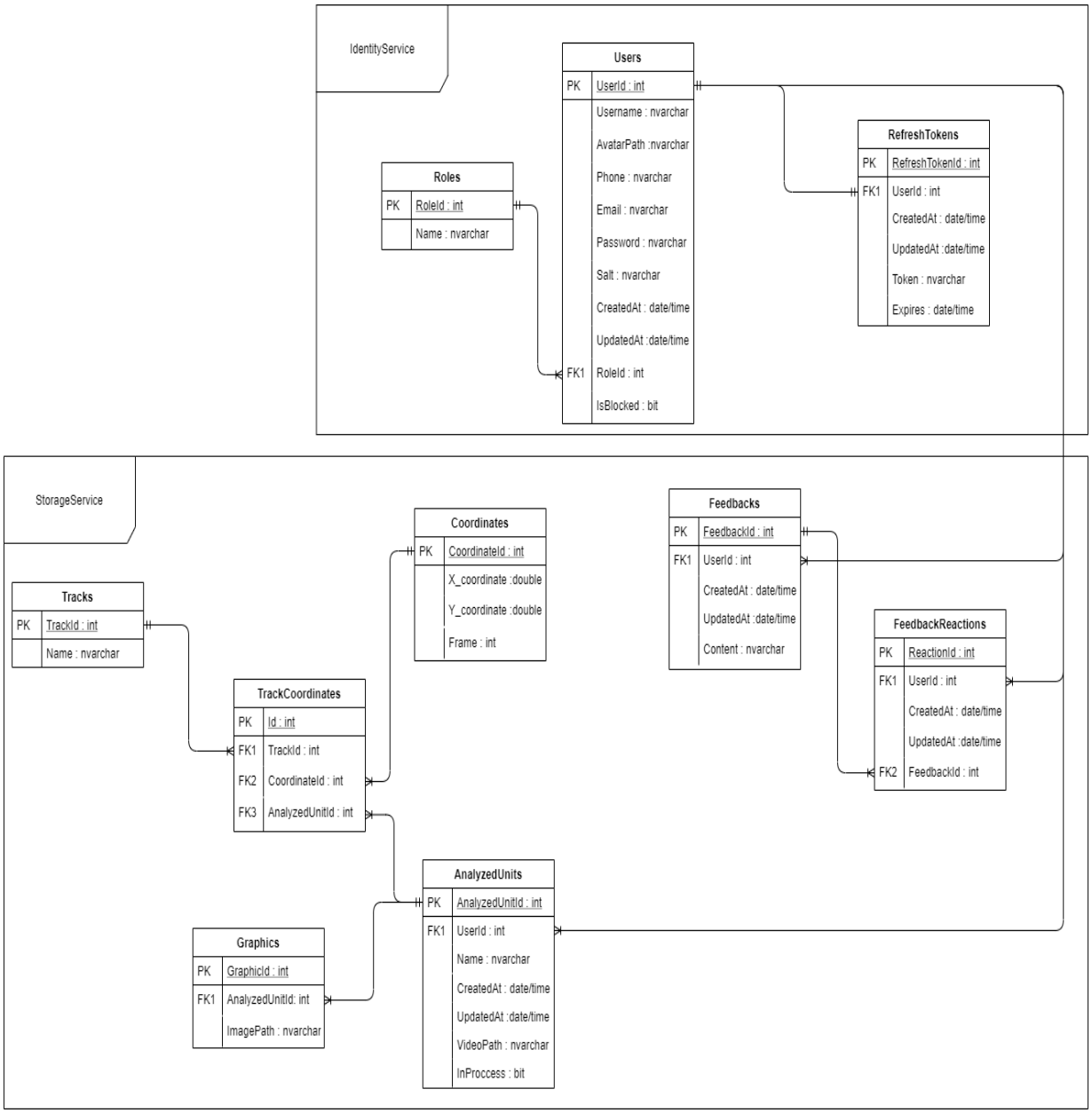

Рисунок 4.8 — Концептуальна модель бази даних в мікросервісах IdentityService та StorageService

Бази даних існують в двох мікросервісах, а саме:

1. IdentityService — що відповідає за авторизацію та реєстрацію, відповідно база даних містить інформацію про користувачів, зображену в таблицях 4.3 — 4.5.

Таблиця 4.3. Структура таблиці “Користувач”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
CreatedAt	DateTime	Дата створення
UpdatedAt	DateTime	Дата останнього оновлення
Username	nvarchar(30)	Імя користувача
AvatarPath	nvarchar(30)	Шлях до аватарки
Phone	nvarchar(30)	Номер телефона
Email	nvarchar(50)	Пошта користувача
Password	nvarchar(30)	Хеш пароля
RoleId	Int32	Посилання на первинний ключ в таблиці “Роль”
Salt	nvarchar(100)	Секретний пароль
IsBlocked	boolean	Статус блокування

Таблиця 4.4. Структура таблиці “Обновлений токен”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
CreatedAt	DateTime	Дата створення
UpdatedAt	DateTime	Дата останнього оновлення
Token	Nvarchar(100)	Токен
Expires	DateTime	Час через який токен спливає
UserId	Int32	Посилання на первинний ключ в таблиці “Користувач”

Таблиця 4.5. Структура таблиці “Роль”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
Name	nvarchar(100)	Права доступу

2. StorageService — що відповідає за розміщення даних про оброблені відеофайли, цим самим забезпечуючи історію для користувачів. Більш детальна інформація представлена в таблицях 4.6 — 4.10.

Таблиця 4.6. Структура таблиці “Трек - Координата”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
TrackId	Int32	Посилання на таблицю “Трек”
CoordinateId	Int32	Посилання на таблицю “Координата”
AnalyzedUnitId	Int32	Посилання на таблицю “Проаналізований екземпляр”

Таблиця 4.7. Структура таблиці “Трек”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
Name	nvarchar(100)	Назва треку

Таблиця 4.8. Структура таблиці “Координата”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
X	Float	Координата X
Y	Float	Координата Y
Frame	Int32	Номер кадра

Таблиця 4.9. Структура таблиці “Графік”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
ImagePath	nvarchar(100)	Шлях до графіку
AnalyzedUnitId	Int32	Посилання на таблицю “Проаналізований екземпляр”

Таблиця 4.10. Структура таблиці “Проаналізований екземпляр”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	Int32	Первинний ключ
CreatedAt	DateTime	Дата створення
UpdatedAt	DateTime	Дата останнього оновлення
Name	Nvarchar(100)	Назва даної обробки
VideoPath	Nvarchar(100)	Шлях до відеофайлу
InProccess	Boolean	Оповіщає про етап виконання аналізу, стадію виконання.
UserId	Int32	Посилання на таблицю “Користувач”

Для проектування та роботою з базою даних було використано ORM Entity Framework Core [11] та Code First підхід. EF Core взаємодіє з базами даних на більш абстрактному рівні, тобто дозволяючи розробникам працювати над класами, що представляють бізнес-об'єкти, замість того, щоб використовувати T-SQL, щоб безпосередньо керувати даними в базі даних.

Використовуючи такий постачальник, як MS SQL Server, представлено об'єкти даних у вигляді класів. EF відображає таблиці та стовпці баз даних із цими класами та властивостями. Завдяки зв'язуванню цих об'єктів та побудові моделі можна керувати даними в таблицях бази даних, використовуючи код .NET. Це позбавляє більшості потреб у користувацьких SQL-запитах та керуванні вставками та виборами вручну. Замість цього можна зробити запит за допомогою LINQ.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

5.1. Інсталяція та системні вимоги

Інтелектуальна система розроблена за допомогою веб-технологій. Однією з найважливіших переваг даного вибору є те, що користувач не потребує встановлення на свою машину великовагового програмного забезпечення. Все, що потрібно для повноцінної роботи — це браузер, що підтримує актуальні веб технології, який зазвичай поставляється з операційною системою, і доступ в Інтернет.

5.2. Інструкція з використання програмного продукту

При вході в клієнтський додаток перед користувачем з’являється головне меню зображене на рисунку 5.1 та у верхньому правому куті можливості для авторизації та реєстрації, що зображені на рисунку 5.2.

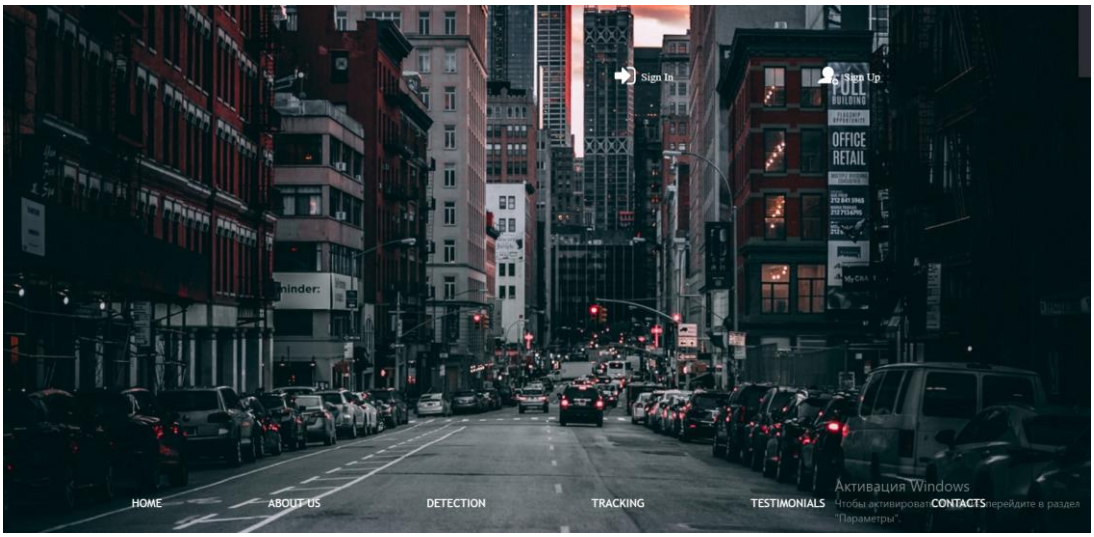

Рисунок 5.1 — Головна сторінка

Рисунок 5.2 — Кнопки авторизації та реєстрації

Для того щоб користувач міг зберігати історія своїх оброблених записів, йому необхідно завести обліковий запис. Для цього необхідно натиснути на кнопку авторизації після чого з’явиться модульне вікно з формою для заповнення даних, що зображена на рисунку 5.3.

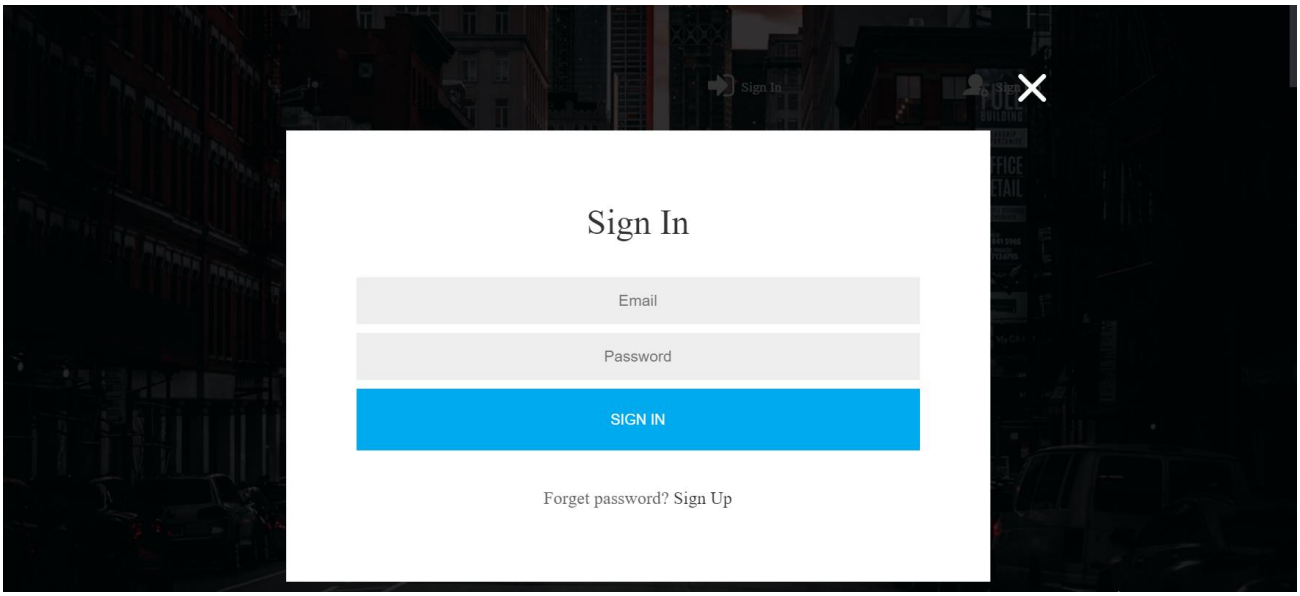

Рисунок 5.3 — Форма авторизації

Якщо користувач немає облікового запису, він має можливість його створити натиснувши на кнопку реєстрації після чого з’явиться інше модульне вікно з формою для заповнення даних, що зображена на рисунку 5.4.

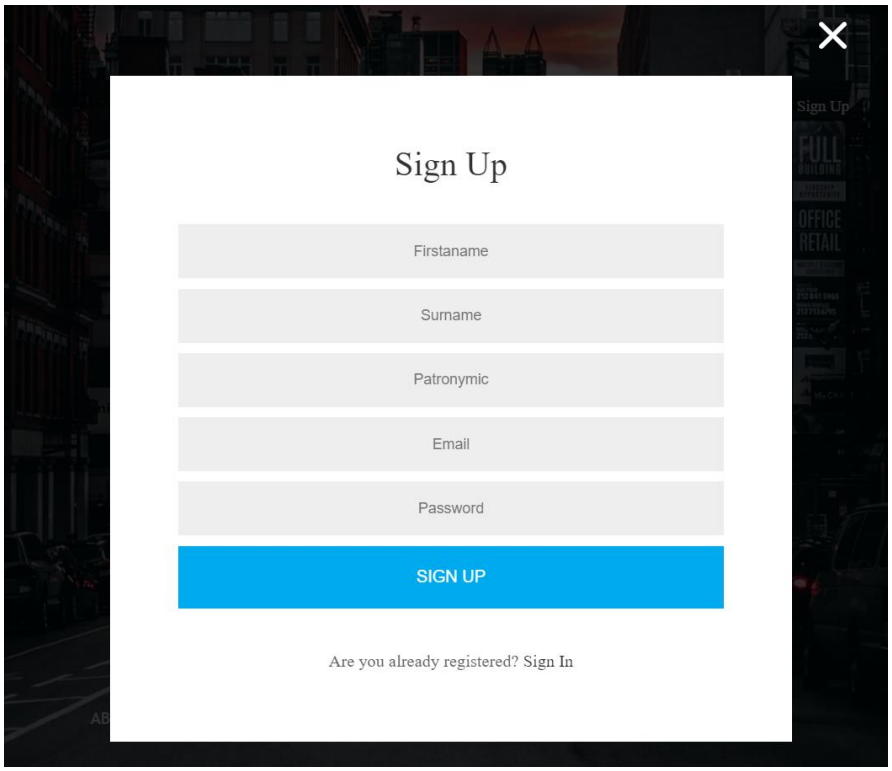

Рисунок 5.4 — Форма реєстрації

В разі якщо даний функціонал не цікавить користувача, він може просто прикріпити відеофайл, що необхідно проаналізувати, вказавши попередньо свою пошту та отримати результати аналізу на вказаний електронний адрес, що зображено на рисунку 5.5.

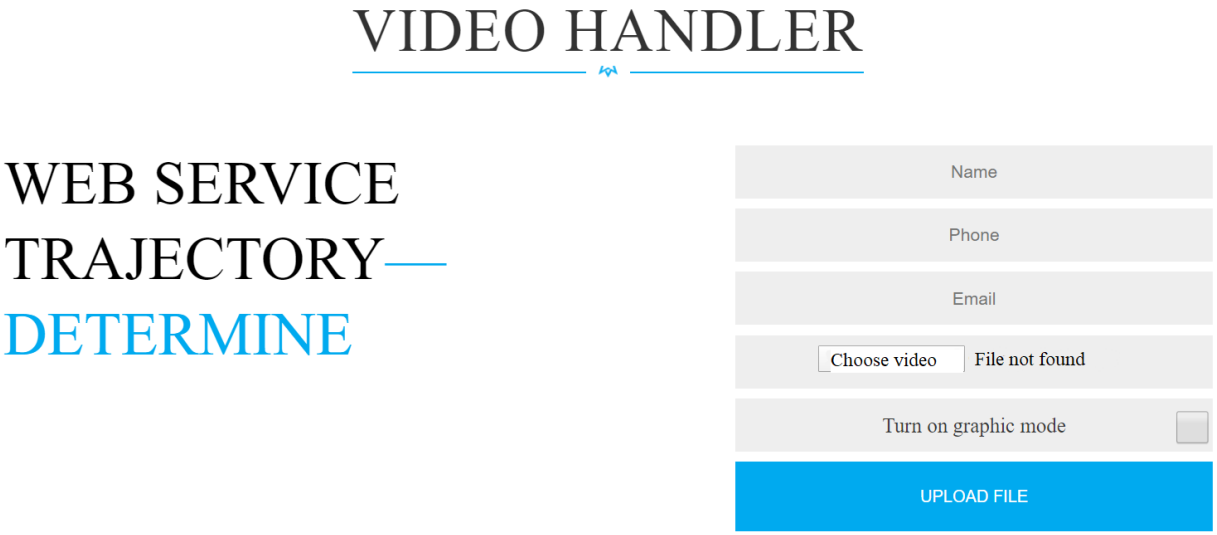

Рисунок 5.5 — Запит на аналіз користувацького відеофайлу

Після того, як користувач пройшов етап автентифікації, він має доступ до історії проаналізованих відеофайлів, що зображено на рисунку 5.6.

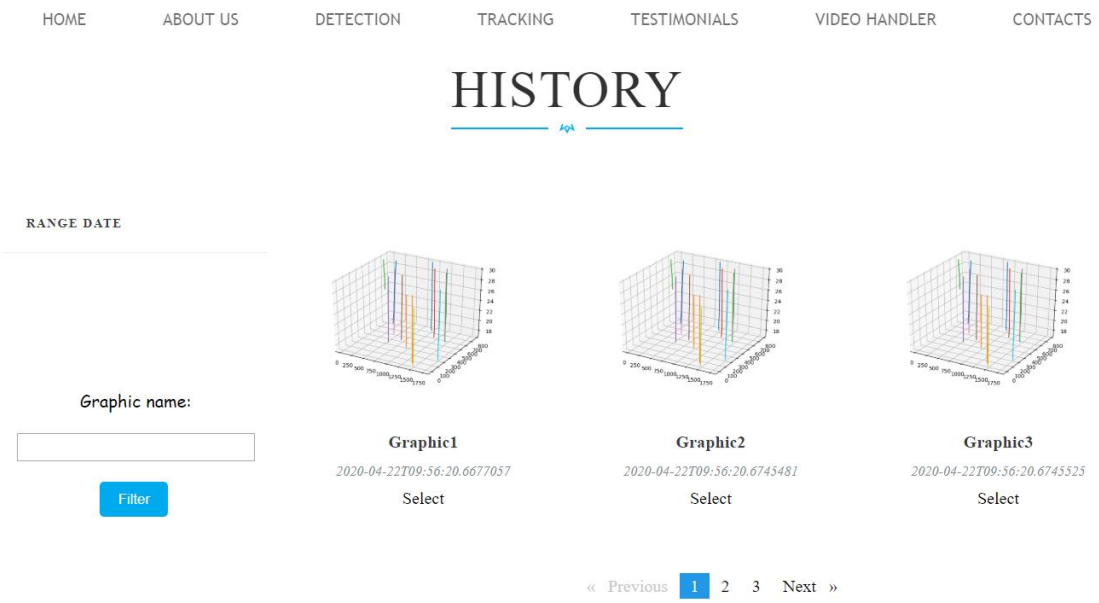

Рисунок 5.6 — Історія проаналізованих відеофайлів

Користувач має можливість переглядати звіт проведеного аналізу, де він отримує структуровані дані у вигляді наборів координат відносно відповідних кадрів, а також візуалізовані дані у виглядів графіків, що зберігаються в особистому кабінеті. Користувач може здійснювати пошук по назві проаналізованих відеофайлів, змінювати назву під власні уподобання, а також фільтрувати дані за датою створення, як зображено на рисунках 5.7 та 5.8.

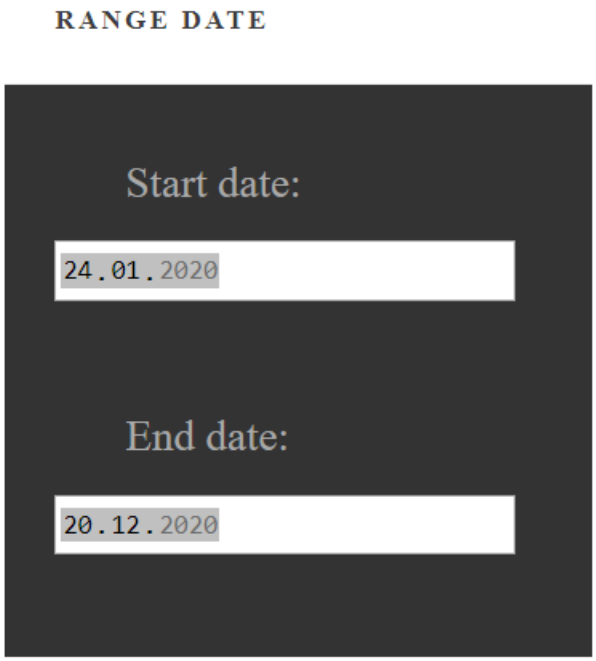

Рисунок 5.7 — Можливість вибору дати

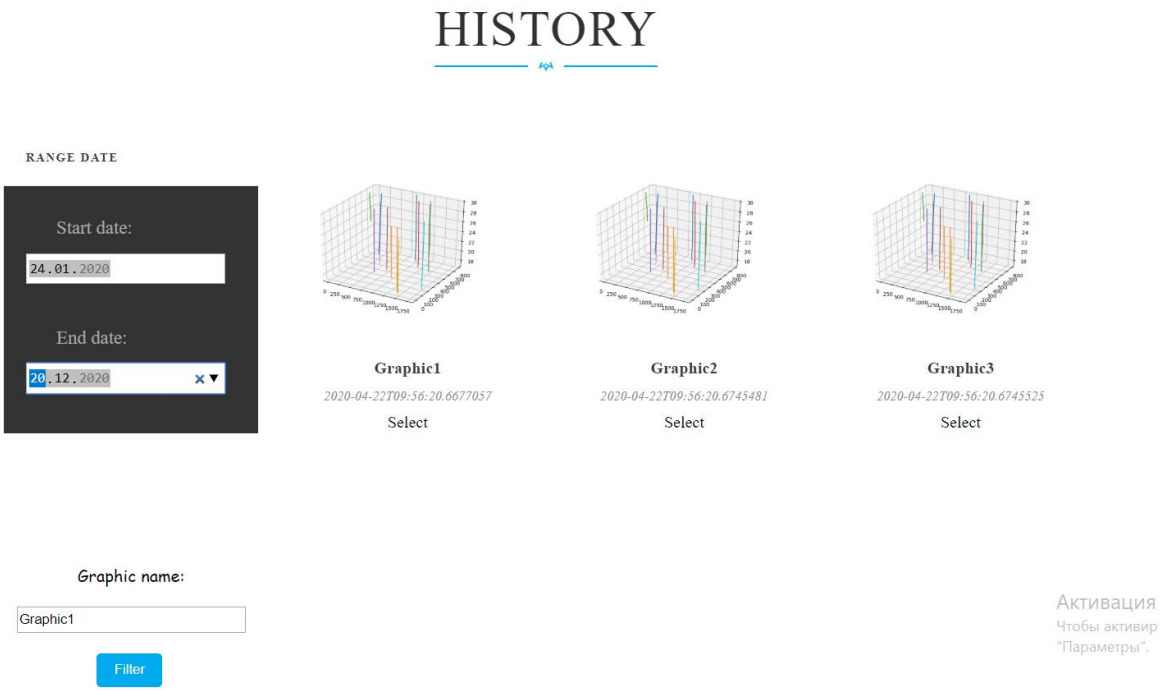

Рисунок 5.8 — Фільтрація та пошук даних

Також користувач має інтерактивний кабінет, що зображено на рисунку 5.9, що випливає з правого боку, де він може переглядати та редагувати свої персональні дані, бачити список проаналізованих відеофайлів та бути оповіщеним в разі завершення поточного аналізу відеофайлу.

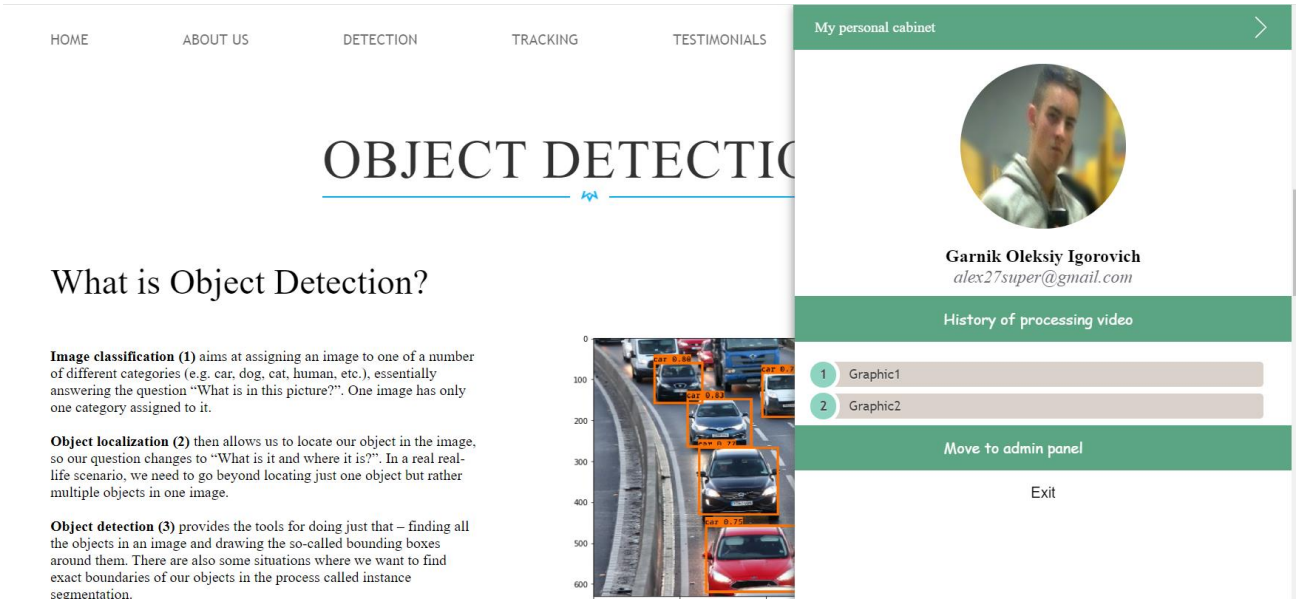

Рисунок 5.9 — Інтерактивний кабінет

5.3. Експерименти та результати

На рисунку 5.10 зображені результати виявлення рухомого транспортного засобу щойно з’явившись на зображені. Фон зображення досить складний, оскільки заважають дерева, що додатково впливають на точність виявлення.

Рисунок 5.10 — Виявлений транспортний засіб

Це означає, що метод виявлення YOLOv3, не завжди зможе розпізнати транспортний засіб на кожному кадрі відеофайлу, як показано на рисунку 5.11.

Рисунок 5.11 — Недосконалість метода виявлення

Проте застосувавши UKF для відстеження транспортного засобу до цього випадку можна знехтувати неточністю, а результати показані на рисунку 5.12. Транспортний

засіб з треком №7, що на деякий момент не був розпізнаний методом виявлення, успішно був збережений за допомогою UKF.

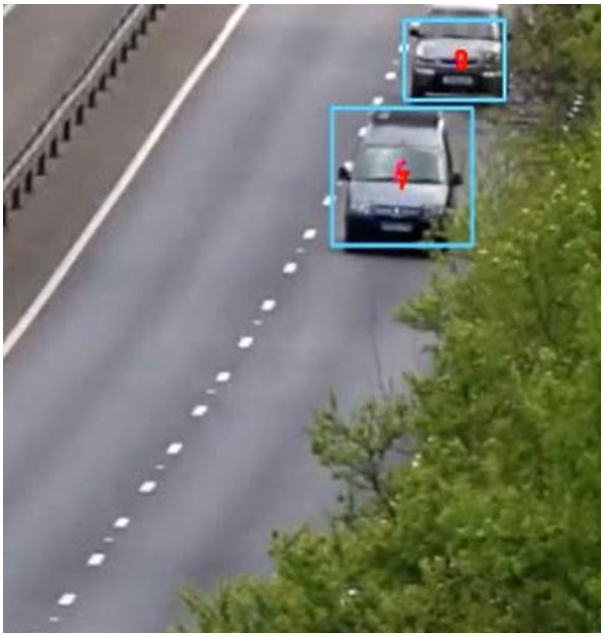

Рисунок 5.12 — Відновлення треку для транспортного засобу

За допомогою інформації про швидкість, оціненої UKF для кожного транспортного засобу на попередньому зображенні, алгоритм виявлення може визначити правильну машину навіть з оклюзією в поточному зображенні .

Далі застосовано метод до подібного, але більш складного випадку, як показано на рисунку 5.13. На даному рисунку зображені результати виявлення та відстеження, коли три транспортних засоби зближуються і відбувається часткова оклюзія.

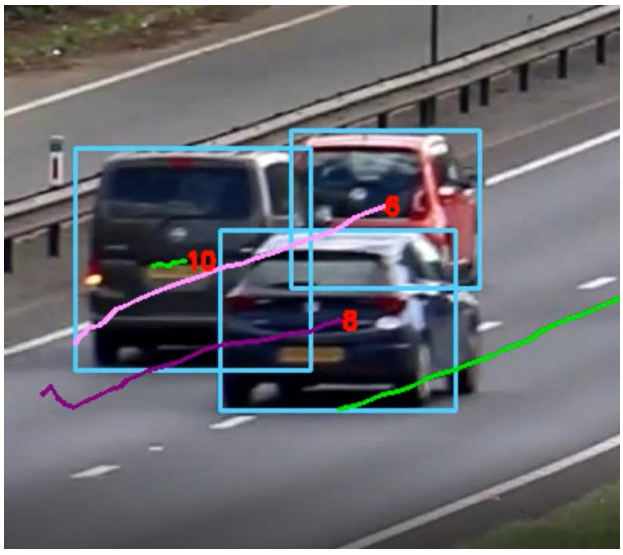

Рисунок 5.13 — Скупчення машин

У цьому сценарії у нас є три машини, і кожна з них перетинається з іншими під час рухомого процесу. Тому коли два та більше транспортних засобів знаходяться досить близько, продуктивність виявлення погіршується, як видно на рисунку 5.14.

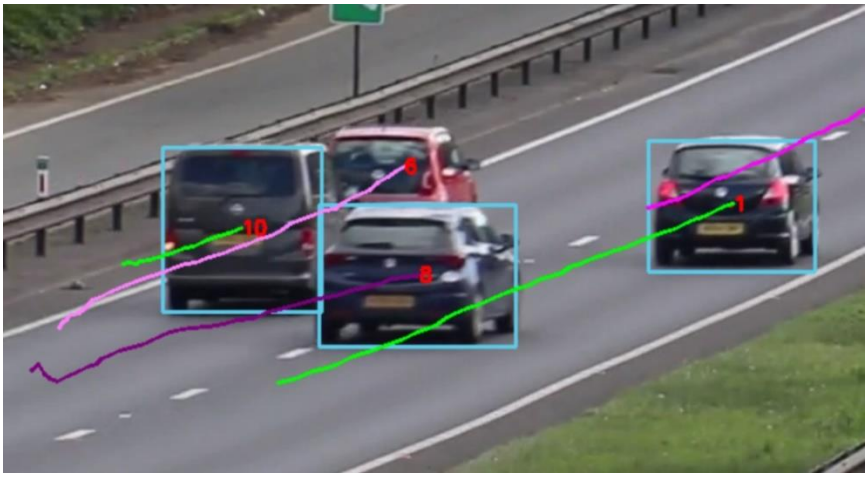

Рисунок 5.14 — Часткова оклюзія

Методу виявлення не вдалося розпізнати автомобіль з треком 6. Крім того, якщо базуватися тільки на основі результатів виявлення (інформації про місцезнаходження), то тоді неможливо відокремити правильні об'єкти після оклюзії. Проте застосувавши метод відстеження до цього випадку, трек успішно зберігається. Отже, алгоритм виявлення надає інформацію про місцезнаходження алгоритму відстеження (тобто, UKF); UKF додатково покращує точність оцінки та отримує остаточне розташування кожного транспортного засобу на поточному зображенні. З рисунка 5.15 чітко видно, що схема відстеження (як реалізована UKF) може добре працювати в такому складному випадку, показуючи детальний рух кожного транспортного засобу до та після оклюзії.

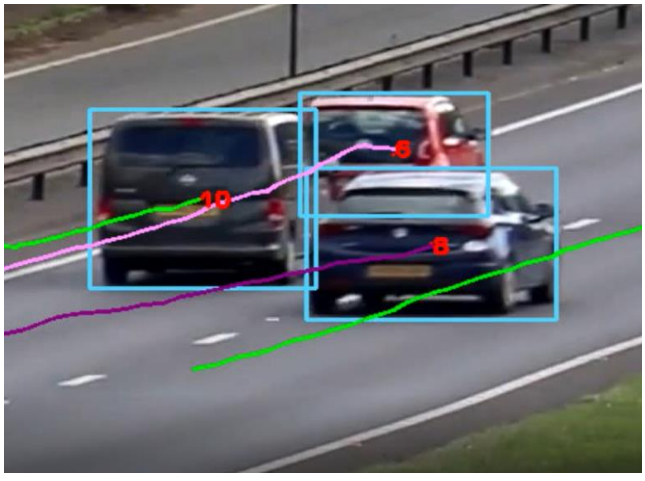

Рисунок 5.15 — Збереження треку №6 та присвоєння автомобілю після оклюзії

Навіть якщо метод виявлення може виявити рухомі об’єкти на кожному зображенні, неоднозначність, викликана оклюзією, є більш складною. На рисунках 5.16 та 5.17 зображені результати виявлення та відстеження, коли два рухомих об’єкта відмічені треками 4 та 5 зближуються і відбувається повна оклюзія, тобто транспортний засіб повністю пропадає з огляду.

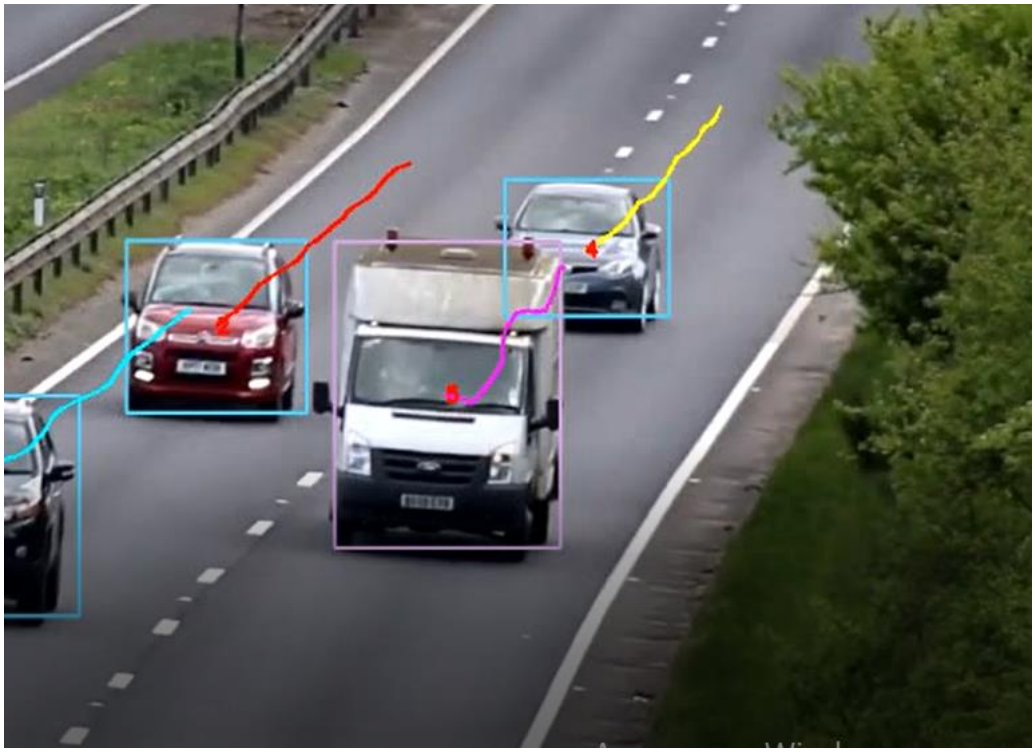

Рисунок 5.16 — Транспортні засоби з треками 4 та 5 до повної оклюзії

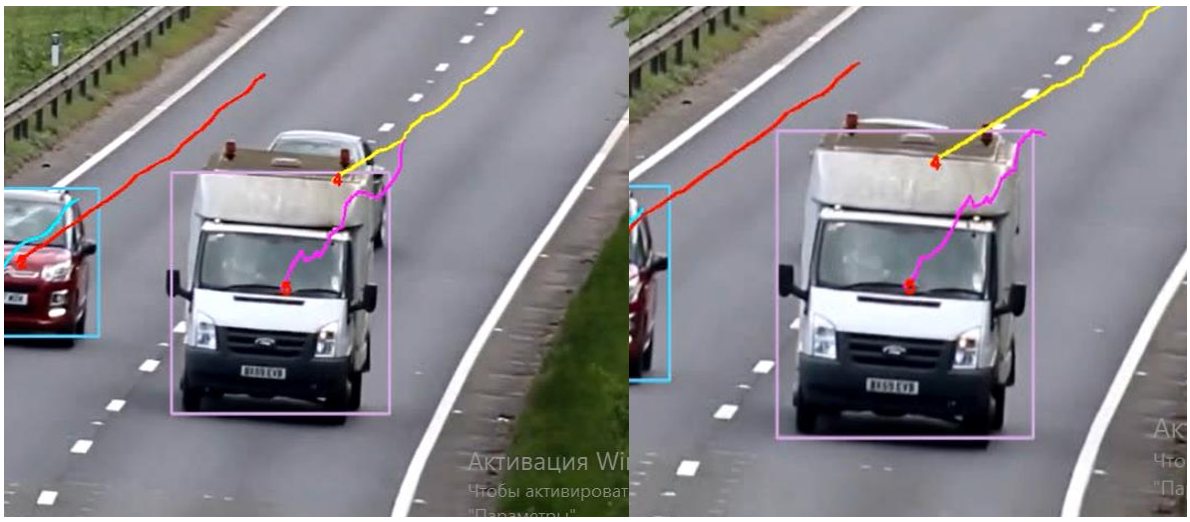

Рисунок 5.17 — Транспортні засоби під час повної оклюзії

На основі підходу відстеження, тобто в даному випадку використання UKF, можна отримати більш точну інформацію про місце розташування для кожного транспортного засобу, що перебуває в русі та відокремити правильні транспортні

засоби після оклюзії. Розглядаючи безперервність швидкості рухомого об’єкта, можна співставити виявлений результат після оклюзії з об’єктом до оклюзії, як показано на рисунку 5.18.

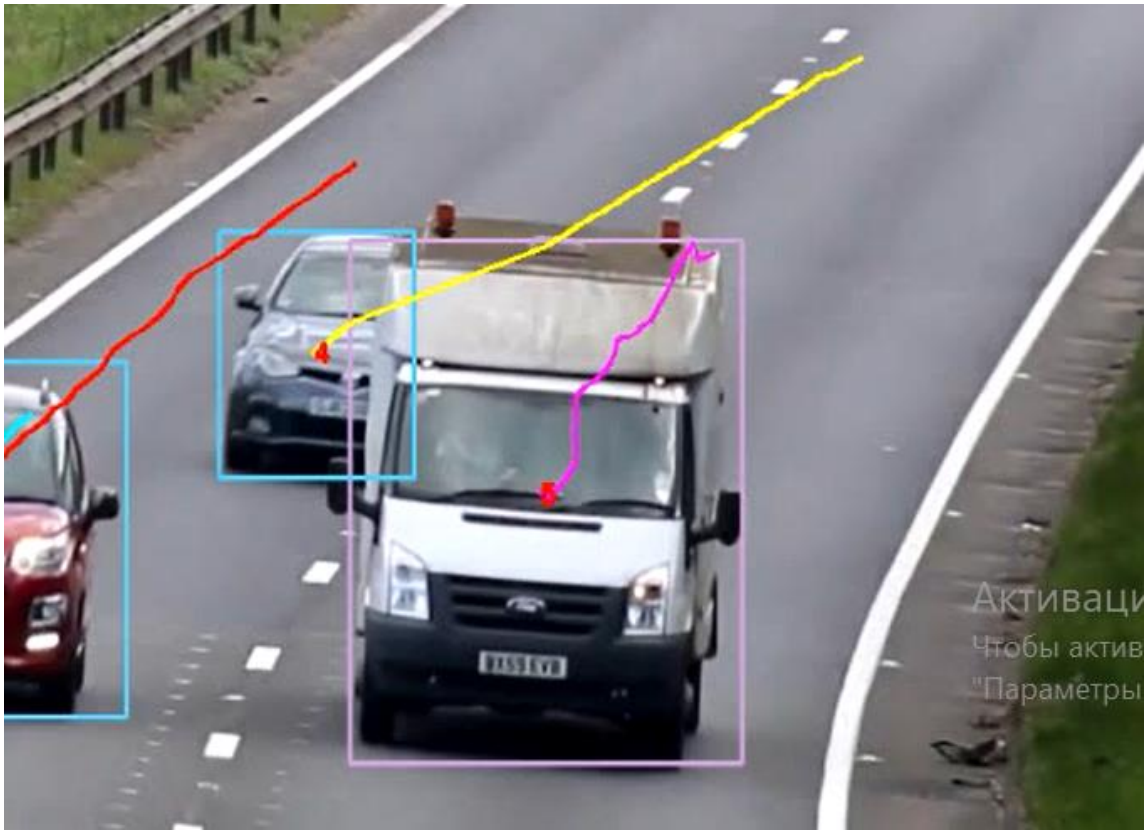

Рисунок 5.18 — Транспортні засоби після повної оклюзії

Це досить важливо для усунення неоднозначності, що існує в більшості методів виявлення об'єктів.

У цій роботі було реалізовано ефективний підхід для відстеження декількох рухомих об'єктів. Зокрема, алгоритм відстеження на основі UKF розроблений для оптимізації нелінійних шляхів переміщення транспортних засобів з оклюзією. UKF оцінює як інформацію про місцезнаходження, так і про швидкість переміщення об'єктів і, нарешті, надає більш точну інформацію про місцеположення. За допомогою зворотного зв’язку алгоритму відстеження, особливо інформації про швидкість, безперервність руху об'єкта може допомогти алгоритму виявлення виявити правильні об'єкти при виникненні оклюзії, що значною мірою вирішує неоднозначність у виявленні та відстеженні декількох рухомих об'єктів. Результати експериментів демонструють, що запропонований метод може правильно виявляти та відстежувати декілька рухомих об'єктів та добре працювати в сценаріях оклюзії.

ВИСНОВКИ

Під час дипломної роботи було проведено огляд існуючих методів, технологій і архітектур програмного забезпечення для чіткого виконання поставленої задачі та відповідно розробки програмного забезпечення, що задовольняє затребуваним вимогам. Наведено перелік переваг щодо створення інтелектуальної системи на прикладі веб застосунку. Обґрунтовано вибір мікросервісної архітектури, що дає значний приріст в продуктивності розробки в майбутньому, а також забезпечує працездатність, гнучкість та чимало інших переваг. Спроектована архітектура клієнтського додатку та серверної частини, що робить розробку програмного забезпечення інтуїтивно зрозумілим та заощаджує ресурси для супроводу та масштабування.

Отже, як результат було розроблене програмне забезпечення, що дозволяє визначати траєкторію руху транспортних засобів за допомогою методів виявлення, алгоритмів відстеження та ідентифікації об’єктів, що в майбутньому стануть базою для обробки та математичного аналізу. Дана робота розширила ряд можливостей, знань в проектуванні та розробці подібних сферах застосування. Інтелектуальна система відповідає висунутим вимогам та виконує поставлені задачі, а саме:

1. Реалізовано особистий кабінет, де користувач може завантажувати відео для аналізу та переглядати звіти про виконану роботу.

2. Можливістю довільно позначати необхідні зони, що надалі допоможуть при подальшому аналізі.

3. Реалізовано мікросервісну архітектуру, що забезпечує ефективність, працездатність та масштабованість системи.

4. Розроблено ефективний підхід для відстеження декількох рухомих транспортних засобів.

5. Розроблено інтуїтивно зрозумілий і зручним користувацький інтерфейс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bradski G. Learning OpenCV. Computer Vision with the OpenCV Library / G. Bradski, A. Kaehler. — O'Reilly Media, Inc., 2008. — 580 с.

2. R.Chen, S.Li, and Z.Li, “From Monolith to Microservices: a Dataflow-Driven Approach”, in 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017. — pp. 466 - 475.

3. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга / Крис Ричардсон. — Питер, 2019. — 544 с.

4. Montesi F. Circuit breakers, discovery, and API gateways in microservices [Електронний ресурс] / F. Montesi, J. Weber // arXiv preprint arXiv:1609.05830. — 2016. — Режим доступу: https://arxiv.org/pdf/1609.05830.pdf.

5. Kálmán R.E. A New Approach to Linear Filtering and Prediction Problems / Rudolf Emil Kálmán — Journal of Basic Engineering, 1960. — pp. 35 - 45.

6. Simon, J. A New Approach for Filtering Nonlinear Systems / J. Simon // Proc. of the American Control Conference, Seattle, USA, Jun, 1997 — pp. 808 - 815.

7. Simon, J. The Scaled Unscented Transformation / J. Simon // Proc. of the American Control Conference, Anchorage USA, May, 2002 — pp. 555 - 559.

8. S.Chopra. A Distributed Version of the Hungarian Method for Multi-Robot Assignment [Електронний ресурс] / S. Chopra, G. Notarstefano, M. Rice, M. Egerstedt // arXiv:1805.08712v1. — 2018. — Режим доступу: https://arxiv.org/pdf/1805.08712.pdf.

9. Redmon J. You only look once: Unified, real-time object detection / J. Redmon, J. Divvala, S. Girshick, R. Farhadi // In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June - 1 July 2016. — pp. 779 - 788.

10. Blaschko, M.B. Non Maximal Suppression in Cascaded Ranking Models / M. B. Blaschko, J. Kannala, E. Rahtu // Scandinavian Conference on Image Analysis (SCIA), 2013.

11. Фримен А. Entity Framework Core 2 для ASP.NET Core MVC для профессионалов / Адам Фримен. — Диалектика, 2019. — 624 с.

12. Гарник О.І. Визначення траєкторії руху автомобілів на основі показників відеокамер / О.І. Гарник, С.І. Шаповалова // Сучасні проблеми наукового забезпечення енергетики: матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів, Київ, 21-24 квітня 2020 р. У 2 т. - К: КПІ ім. Ігоря Сікорського, 2020.- Т.2.- С. 103.

ДОДАТОК А

Інтелектуальна система визначення траєкторії руху транспортних засобів

Специфікація

УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б_81	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б_81	tracker.py	Основний компонент
УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б_81	analyzer.py	Основний компонент

ДОДАТОК Б

Інтелектуальна система визначення траєкторії руху транспортних засобів

Текст програми

УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б

Аркушів 7

Київ 2020

```
from track import Track
import numpy as np
from scipy.optimize import linear_sum_track_detection_assign
from scipy.linalg import block_diag

# клас трекер, що відповідає за всі етапи відстеження транспортних засобів
class Tracker(object):
    def __init__(self, distance_thresh, maximum_count_frames_to_skip, maximum_length_of_path):
        # дозволений поріг для присвоєння
        self.distance_thresh = distance_thresh
        # максимальна кількість кадрів дозволена для пропуску детектора
        self.maximum_count_frames_to_skip = maximum_count_frames_to_skip
        # довжина шляху, яка буде відображатися на відео
        self.maximum_length_of_path = maximum_length_of_path
        # кількість треків, які відповідають за транспортні засобі
        self.tracks = []
        # відповідний ідентифікатор для трекінгу
        self.track_id_count = 1

    # метод, що відповідає за фази фільтру Калмана
    def update(self, detection_vehicles):
        # якщо, кількість треків дорівнює 0, то необхідно проініалізувати систему
        if len(self.tracks) == 0 :
            for i in range(len(detection_vehicles)):
                track = Track(detection_vehicles[i], self.track_id_count)
                self.track_id_count += 1
                self.tracks.append(track)

        N = len(self.tracks)
        M = len(detection_vehicles)
        distance_matrix = np.zeros(shape=(N, M))  # ініціалізуємо матрицю, що містить евклідову
довжину між детекціями та існуючими траєкторіями
        for i in range(len(self.tracks)):
            for j in range(len(detection_vehicles)):
                try:
```

```
            # обчислюється Евклідова відстань
            diff = self.tracks[i].prediction - detection_vehicles[j]
            distance = np.sqrt(diff[0][0]*diff[0][0] + diff[1][0]*diff[1][0])
            distance_matrix[i][j] = distance
        except:
            pass

distance_matrix = (0.5) * distance_matrix
# створюємо список, що відповідає за присвоєні треки та ініціалізуємо
track_detection_assign = []
for _ in range(N):
    track_detection_assign.append(-1)

# результати роботи угорського алгоритму, отримуємо пару трек - детекція
row_index, column_index = linear_sum_track_detection_assign(distance_matrix)

for i in range(len(row_index)):
    track_detection_assign[row_index[i]] = column_index[i]

# знаходимо траєкторії, що не були присвоєні до детекції
trajectory_not_assigned = []
for i in range(len(track_detection_assign)):
    if (track_detection_assign[i] != -1):
        if (distance_matrix[i][track_detection_assign[i]] > self.distance_thresh):
            track_detection_assign[i] = -1
            trajectory_not_assigned.append(i)
            pass
    else:
        self.tracks[i].skipped_frames += 1

# треки які необхідно видалити, оскільки кількість пропущених кадрів перевищує норму
tracks_for_delete = []
for i in range(len(self.tracks)):
    if (self.tracks[i].skipped_frames > self.maximum_count_frames_to_skip):
```

```
            tracks_for_delete.append(i)

    if len(tracks_for_delete) > 0:
        for id in tracks_for_delete:
            if id< len(self.tracks):
                del self.tracks[id]
                del track_detection_assign[id]
            else:
                print("Errors")
    # Якщо детекції не було присвоєну відповідну існуючу траєкторію, то тоді
    # ініціалізуємо її треком та вважаємо даний транспортний засіб нововиявленим
    detections_not_assigned = []
    for i in range(len(detection_vehicles)):
        if i not in track_detection_assign:
            detections_not_assigned.append(i)

    if(len(detections_not_assigned) != 0):
        for i in range(len(detections_not_assigned)):
            track = Track(detection_vehicles[detections_not_assigned[i]], self.track_id_count)
            self.track_id_count += 1
            self.tracks.append(track)

    for i in range(len(track_detection_assign)):
        # виконуємо першу фазу фільтра Калмана, екстраполяцію
        self.tracks[i].KF.KF.predict()

        if(track_detection_assign[i] != -1):
            self.tracks[i].skipped_frames = 0
            print("detection_vehicles: ", np.array(detection_vehicles[track_detection_assign[i]]))
            print("detection_vehicles: ", detection_vehicles[track_detection_assign[i]][0][0])
            coordinate_xy = np.array([ detection_vehicles[track_detection_assign[i]][0][0],
            detection_vehicles[track_detection_assign[i]][1][0]])
            print(coordinate_xy)
            # виконуємо другу фазу фільтра Калмана, фазу корекції
            self.tracks[i].KF.KF.update(coordinate_xy)
```

```
            # зберігаємо прогнозоване значення
             self.tracks[i].prediction = np.array([[self.tracks[i].KF.KF.x[0]], [self.tracks[i].KF.KF.x[1]]])
             self.tracks[i].trace.append(np.array([self.tracks[i].KF.KF.x[0], self.tracks[i].KF.KF.x[1]]))
```

```
# клас, що відповідає за повідомлення для мікросервісу StorageService, сигналізуючи про початок виконання аналізу для відповідного користувача
class StartVideoProcessingEvent(object):
    def __init__(self, user_id):
    self.userId = user_id

# клас, що відповідає за повідомлення для мікросервісу StorageService, сигналізуючи про закінчення виконання аналізу для відповідного користувача та відправлення результатів
class FinishVideoProcessingEvent(object):
    def __init__(self, video_path, tracks_dictionary, images_path, user_id):
    self.UserId = user_id
    self.VideoPath = video_path
    self.ImagePaths = images_path
    self.tracksDictionary = tracks_dictionary

class Point(object):
    def __init__(self, x, y, radius):
        self.x = x
        self.y = y
        self.radius = radius

    def clone(self):
        return type(self)(
        self.x,
        self.y,
        self.radius)

    def __str__(self):
        return f"X: {self.x}, Y: {self.y}"

import numpy as np
```

```
import pylab as plt
from mpl_toolkits.mplot3d import Axes3D
import os
import datetime

class Graphic(object):
    def __init__(self, tracks_dictionary):
        self.tracks_dictionary = tracks_dictionary

class GraphicBuilder(object):
    def __init__(self, tracks_dictionary):
        self.tracks_dictionary = tracks_dictionary
        self.x = []
        self.y = []
        self.frame = []
        self.default_path = "src\\assets\\"
        self.imagePaths = []

    # метод, що за результатами аналізатора відеофайлів будує всі необхідні графіки
    # візуалізуючи результати
    def build(self):
        fig_all = plt.figure()
        ax_all = fig_all.add_subplot(111, projection='3d')

        full_path = self.generate_path()
        for track_id, value in self.tracks_dictionary.items():

            fig_track = plt.figure()
            ax_track = fig_track.add_subplot(111, projection='3d')

            print("Track_id:", track_id);
            for frame, coordinate in value.items():
                x = coordinate[0]
                y = coordinate[1]
```

```
            print("Frame: ", frame, ", Coordinate: ", coordinate[0], coordinate[1])
            self.x.append(coordinate[0])
            self.y.append(coordinate[1])
            self.frame.append(frame)

        ax_all.plot(self.x, self.y, self.frame, label="Track_id: " + str(track_id))
        ax_track.plot(self.x, self.y, self.frame, label="Track_id: " + str(track_id))

        self.reset_configuration()

        image_path = full_path + '\\track' + str(track_id)
        self.imagePaths.append(image_path)
        fig_track.savefig(image_path)

    fig_all.savefig(full_path + '\\all_tracks')

    self.imagePaths.append(full_path + '\\all_tracks')

# метод, що ініціалізує параметри
def reset_configuration(self):
    self.x = []
    self.y = []
    self.frame = []
# метод, що генерує шлях для зберігання файлів
def generate_path(self):
    now = datetime.datetime.now()
    folder_name = str(now.date()) + str(now.time()).replace(":", "_")
    full_path = self.default_path + folder_name

    try:
        os.mkdir(full_path)
    except OSError:
        print("Can't open file" % full_path)
    else:
        print("Successfully create %s " % full_path)
```

```
        return full_path

from filterpy.common import Q_discrete_white_noise
from filterpy.kalman import KalmanFilter
from filterpy.kalman import UnscentedKalmanFilter
from filterpy.kalman import MerweScaledSigmaPoints
import numpy as np

# класс трек, що відповідає за збереження всіх необхідних параметрів:
# Ідентифікатор треку, параметр, що ідентифікує кожен трек
# параметр, що зберігає ширину, довжину обмежувального вікна, а також центроїд у вигляді
# координат x та y
# кількість пропущених кадрів
# видиму довжину, кількість кадрів у яких був виявлений трек
# парамери фільтра Калмана
class Track(object):
    def __init__(self, prediction, trackId):
        self.track_id = trackId
        self.KF = self.filter('UKF')
        self.prediction = prediction
        self.skipped_frames = 0
        self.trace = []
    # метод, що обирає в залежності від параметра, тип фільтру Калмана
    def filter(self, type):
        switcher = {
            'KF': KalmanFilterCustom(),
            'UKF': UnscentedKalmanFilterCustom()
        }
        return switcher.get(type, 'Invalid filter')

from args_parser import ArgsParser
import cv2
import copy
from detector import DetectorFactory
```

```
from tracker import Tracker
import numpy as np
from graphic import GraphicBuilder
from events import FinishVideoProcessingEvent
from event_bus import EventBus
import pika
import pickle
import json

# класс аналізатор, що компонує всі необхідні етапи в один та представляє загальну схему моніторингу транспортних засобів
class Analyzer(object):
    # містить в собі об’єкти, що відповідають за виявлення та відстеження транспортних засобів
    def __init__(self, vehicles_detector, vehicles_tracker):
        self.vehicles_detector = vehicles_detector
        self.vehicles_tracker = vehicles_tracker

    # метод, що віповідає за процес аналізу
    def analyze(self, video_path, output_video_path, user_id):
        # захоплюємо кадри з відеофайлу
        cap = cv2.VideoCapture(video_path)
        writer = None
        # ініціалізуємо параметр, що віпдповідає за кількість пропущених кадрів
        skip_frame_age = 0
        color_for_vehicles_detection = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0),
                    (0, 255, 255), (255, 0, 255), (255, 127, 255),
                    (127, 0, 255), (127, 0, 127)]
        # структура, що відповідає за збереження даних з проаналізованого відеофайлу
        tracks_dictionary = {}

        frame_count = 0
        #цикл виконання відеофайлу
        while (True):
            # захоплюється кадр один за одним
            ret, frame = cap.read()
```

```
        if frame is None:
            break

        # запускаємо метод виявлення обєктів та отримуємо список координат у вигляді x, y та w, h, де x, y - координати центроїда та w, h – ширина та висота обмежувального вікна відповідно
        vehicles_center = self.vehicles_detector.detect(frame)
        # якщо на кадрі було знайдено транспортні засоби, то запускаємо алгоритм відстеження
        if (len(vehicles_center) > 0):
            self.vehicles_tracker.update(vehicles_center)

            for i in range(len(self.vehicles_tracker.tracks)):
                #перевірити чи треки вже є в структурі, чи їх потрібно додати
                if self.vehicles_tracker.tracks[i].track_id not in tracks_dictionary:
                    tracks_dictionary[self.vehicles_tracker.tracks[i].track_id] = {}

                if (len(self.vehicles_tracker.tracks[i].path) > 1):
                    # додати координати та відповідні дані
                    # Актуально для Unscended Kalman Filter
                    tracks_dictionary[self.vehicles_tracker.tracks[i].track_id][frame_count] = [
                        self.vehicles_tracker.tracks[i].path[len(self.vehicles_tracker.tracks[i].path) - 1][0],
                        self.vehicles_tracker.tracks[i].path[len(self.vehicles_tracker.tracks[i].path) - 1][1]]

                    for j in range(len(self.vehicles_tracker.tracks[i].path) - 1, 1, -1):
                        # Unscented Kalman Filter
                        x1 = self.vehicles_tracker.tracks[i].path[j][0]
                        y1 = self.vehicles_tracker.tracks[i].path[j][1]
                        x2 = self.vehicles_tracker.tracks[i].path[j - 1][0]
                        y2 = self.vehicles_tracker.tracks[i].path[j - 1][1]
                        clr = self.vehicles_tracker.tracks[i].track_id % 9
                        cv2.line(frame, (int(x1), int(y1)), (int(x2), int(y2)), color_for_vehicles_detection[clr], 2)

                        text = "{}".format(self.vehicles_tracker.tracks[i].track_id)
                        cv2.putText(frame, text)
```

```
            frame = cv2.resize(frame, (1250, 750))
            cv2.imshow('Tracking', frame)

            writer.write(frame)

        # після виконання всіх етапів виявлення, запускаємо GraphicBuilder щоб створити всі необхідні графіки
        graphic_builder = GraphicBuilder(tracks_dictionary)
        graphic_builder.build()
        # завантажуємо до брокеру повідомлень, повідомлення про закінчення оброки відеофайлу
        finish_video_processing_event = FinishVideoProcessingEvent(output_video_path, graphic_builder.tracks_dictionary, graphic_builder.imagePaths, user_id)

        event_bus = EventBus(finish_video_processing_event)
        event_bus.publish()

        cap.release()
        writer.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    args_parser = ArgsParser()
    args_parser.parse()

    # створення методу виявлення
    detectorFactory = DetectorFactory()
    vehicles_detector = detectorFactory.create_detector('yolo', args_parser)

    # Створення методу відстеження
    vehicles_tracker = Tracker(100, 5, 5)
```

ДОДАТОК В

Інтелектуальна система визначення траєкторії руху транспортних засобів

Опис програми

УКР.КПІ_ім_Ігоря_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126_20Б

Аркушів 10

Київ 2020

АНОТАЦІЯ

Додаток містить опис основних компонентів програмної системи, що дозволяє визначати траєкторію руху транспортних засобів за допомогою методів виявлення, алгоритмів відстеження та ідентифікації, що в майбутньому стануть базою для обробки та математичного аналізу, а саме реалізовано:

1. Відстеження транспортних засобів на основі детектора YOLOv3.

2. Ефективний підхід для відстеження декількох рухомих об'єктів на основі сігма-точкового фільтру Калмана та Угорського алгоритму, що пов’язує автомобілі від одного кадру до іншого на основі оцінок.

Інтелектуальна веб система розроблена за допомогою різноманітних технологій, а саме деякі сервіси створені на мові програмування C#, фреймворку ASP.NET Core та системи управління базами даних MS SQL Server:

1. IdentityService, що відповідає за задачі авторизації та реєстрації.

2. StorageService, що відповідає за збереження даних, а також розсилає результати проаналізованих відео файлів на пошту користувачам.

Для розроблення наступних модулів було використано мову програмування Python, фреймворк Flusk та бібліотеку OpenCV:

1. DrawingService, що дозволяє позначати необхідні додаткові зони для аналізу.

2. AnalyzerService, що виконує задачі виявлення, відстеження та ідентифікації транспортних засобів.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку опис основних компонентів інтелектуальної веб системи, що відповідають за визначення, відстеження та ідентифікації транспортних засобів. У другому додатку містить програмний код згаданих вище компонентів.

Розроблена система працює у вигляді веб додатку, тому немає ніяких вимог до апаратної платформи. Для того, щоб розгорнути додаток локально, необхідно встановити наступні компоненти:

1. Для роботи AnalyzeService та DrawingService: Python 3.6, фреймворк Flusk та бібліотеку комп'ютерного зору OpenCV. Мікросервіси розроблені у середовищі програмування PyCharm.

2. Для роботи StorageService та IdentityService: C# не нижче 7 версії, фрейворк ASP.NET Core та провайдер баз даних MS SQL Server. Мікросервіси розроблені у середовищі програмування Visual Studio 2019.

3. Для роботи з клієнтської частини: необхідно встановити сервер Node.js і пакетний менеджер npm. Середовище програмування: Visual Studio Code.

4. Для взаємодії між мікросервісами: RabbitMQ 3.8.4

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Інтелектуальна веб система надає наступний функціонал:

1. Автентифікація користувачів.

2. Реєстрація користувачів.

3. Можливість завантажувати відеофайли для подальшої обробки.

4. Можливість переглядати звіти про виконаний аналіз у вигляді наборів координат відносно відповідних кадрів, а також візуалізовані дані у виглядів графіків, що зберігаються в особистому кабінеті.

5. Можливість довільно позначати необхідні зони, що допоможуть при подальшому аналізі.

Розроблений програмний додаток рекомендований для використання в сфері цифрового виробництва, що широко застосовується в багатьох галузях економіки, таких як автономні автомобілі, системи допомоги водію, системи контролю тощо.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Інтелектуальна система, що дозволяє визначати траєкторію руху автомобілів розроблена на основі мікросервісної архітектури. Для комунікації між мікросервісами використовуються два протоколи — запити і відповіді HTTP з вихідними API і легкі асинхронні повідомлення при передачі оновлень, що реалізується за допомогою брокеру повідомлень RabbitMQ. Реалізована єдина точка входу за допомогою Ocelot, що дозволяє ізолювати клієнтів від структури сервісів, позбавляє від необхідності самостійно визначати місце розташування сервісу та надає уніфікований і спрощений API.

Для поділу операцій зчитування та запису в окремі моделі, використовуючи команди та запити для читання даних, було використано шаблон CQRS.

Веб служби реалізовані за допомогою REST архітектури, що є простим способом взаємодії між системами.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Оскільки інтелектуальна система розроблена на прикладі веб системи, то користувач не потребує встановлення на свою машину великовагового програмного забезпечення. Все, що потрібно для повноцінної роботи — це браузер, що підтримує актуальні веб-технології і доступ в Інтернет. Також даний програмний комплекс не залежить від операційної системи та обсягу оперативної пам’яті.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблений програмний продукт не потребує інсталяції та для запуску необхідно відкрити браузер на сторінці додатку за посиланням http://localhost:4200/. Після чого перед користувачем з’явиться головна сторінка, де надалі користувач зможе обрати необхідний функціонал.

ВХІДНІ ДАНІ

Вхідна інформація для інтелектуальної веб системи:

1. Дані для створення облікового запису, а саме:
   — пароль;
   — пошту;
   — фото профілю;
   — ім’я, прізвище та ім’я по батькові;
2. Відео для обробки аналізу.

ВИХІДНІ ДАНІ

Вихідна інформація для інтелектуальної веб системи:

1. Оброблений відеофайл.

2. Інформація по обробленому відеофайлі, а саме користувач отримує структуровані дані у вигляді наборів координат відносно відповідних кадрів, а також візуалізовані дані у виглядів графіків, що зберігаються в особистому кабінеті.