МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інженерії програмного забезпечення, канд. техн. наук, доцент,_____Є.О. Давиденко
«____»_____2023 року

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

НАВІГАЦІЯ ТРАНСПОРТНИХ ЗАСОБІВ НА ОСНОВІ АНАЛІЗУ ТРАФІКУ З ВИКОРИСТАННЯМ КОМП'ЮТЕРНОГО ЗОРУ

Спеціальність «Інженерія програмного забезпечення»

121 – КРМ.1 – 608. 21710806

Студент _____ М. О. Губарєв
«__»_____2023 р.

Керівник канд. техн. наук, доцент _____ А. П. Бойко
«__»_____2023 р.

Консультант д-р. біол. наук, професор _____ Л. І. Григор'єва
«__»_____2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії програмного забезпечення, канд. техн. наук, доцент,
_____Є.О. Давиденко
«____»_____2023 року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

Видано студенту групи 608 факультету комп'ютерних наук
_____Губарєву Михайлу Олександровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи: «Навігація транспортних засобів на основі аналізу трафіку з використанням комп'ютерного зору»
Затверджено наказом по ЧНУ ім. Петра Могили від _____№ ____
2. Строк подання студентом роботи ____ _____ 20__ р.
3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Вихідні (початкові) дані до роботи: відео з вебкамер, програмні бібліотеки для розпізнавання образів. Результати роботи: алгоритм розпізнавання автомобілів, алгоритм швидкої обробки даних про стан дорожнього трафіку, алгоритм вибору найкращого маршруту руху містом, діаграми класів, програмний код.
4. Перелік питань, що підлягають розробці: титульний аркуш, завдання на КРМ, анотація українською та англійською мовами, зміст, перелік умовних скорочень, символів, одиниць та термінів. Вступ, перший, другий, третій, четвертий розділи та висновки до них. Спеціальна частина з охорони праці. Висновки. Перелік джерел посилання. Додатки.
5. Перелік графічних матеріалів: діаграми діяльності, діаграми класів, діаграма варіантів використання застосунку, презентація.
6. Завдання до спеціальної частини: два підрозділи присвячені питанням охорони праці та цивільному захисту.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Бойко А. П., канд.техн.наук, доцент	Кафедра комп'ютерної інженерії	Основна частина
Григор'єва Л. І., д-р. біол. наук, професор	Кафедра екології	Спеціальна частина з охорони праці та безпеки в надзвичайних ситуаціях

Керівник роботи _канд.техн.наук, доцент_Бойко А. П_____
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання __Губарєв М.О._____
(прізвище, ім'я, по батькові студента)

_____
(підпис)

Дата видачі завдання « ___ » _____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Навігація транспортних засобів на основі аналізу трафіку з використанням комп'ютерного зору»

	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРМ	05.11.2022	05.11.2022	виконано
2	Огляд літератури за темою роботи	07.11.2022	07.11.2022	виконано
3	Складання календарного плану КРМ	15.11.2022	25.11.2022	виконано
4	Аналіз предметної області	26.11. 2022	30.11. 2022	виконано
5	Розробка проєктних рішень	02.12. 2022	07.12. 2022	виконано
6	Моделювання та конструювання ПЗ	11.12. 2022	21.12. 2022	виконано
7	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	23.12. 2022	18.01.2023	виконано
8	Розробка спеціальної частини з охорони праці	19.01.2023	25.01.2023	виконано
9	Відгук керівника КРМ	27.01.2023	31.01.2023	виконано
10	Оформлення КРМ та презентації	01.02.2023	04.02.2023	виконано
11	Попередній захист	13.02.2023	13.02.2023	виконано
12	Рецензування	10.02.2023	17.02.2023	виконано
13	Завершення оформлення КРМ та презентації	13.02.2023	13.02.2023	виконано
14	Захист кваліфікаційної роботи	24.02.2023	24.02.2023	виконано

Розробив студент _Губарєв М.О._____ _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 20__ р.

Керівник роботи канд.техн.наук, доцент_Бойко А. П_____ _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
«Навігація транспортних засобів на основі аналізу трафіку з використанням комп'ютерного зору»
Студент 608м гр.: Губарєв Михайло Олександрович
Керівник: канд. техн. наук, доцент Бойко А. П.

Кваліфікаційна робота магістра присвячена розробці вебсервісу моніторингу транспортних засобів за допомогою комп'ютерного зору. Метою роботи є автоматизація процесу моніторингу стану завантаженості автомобільних доріг за рахунок створення мережі динамічних відеокамер спостереження та розроблення адаптивного вебзастосунку для водіїв, що покращить умови для керування транспортними засобами.

Кваліфікаційна робота магістра складається зі вступу, чотирьох розділів, спеціальної частини з охорони праці, висновків та додатків. У вступі визначається актуальність теми, що приймається за мету та невеликий огляд поставленої задачі, предмет дослідження та об'єкт дослідження.

Об'єктом дослідження роботи є процес аналізу дорожнього трафіку з використанням технологій комп'ютерного зору.

Предметом дослідження є технології аналізу даних для навігації транспортних засобів в умовах високої завантаженості дорожніх шляхів міст у часи пік.

У першому розділі описується огляд існуючих аналогів навігаційних систем. Також, описується специфікація вимог до вебсервісу. У другому розділі описується моделювання вебсервісу аналізу відео за допомогою комп'ютерного зору. У третьому розділі описуються основні технології розробки та проєктування вебсервісу навігації. У четвертому розділі описується розробка прототипів серверної частини та адаптивного інтерфейсу вебзастосунку, його тестування та інструкції користувача. У висновках проводиться аналіз роботи та отриманих результатів. За результатами дослідження було розроблено вебсервіс для навігації водіїв на основі моніторингу автомобільного трафіку за допомогою комп'ютерного зору.

Дипломна робота містить 99 стор. основної частини, 77 рис., 8 табл., 57 посилань.

Ключові слова: *навігаційна система, комп'ютерний зір, відеоспостереження, каскади Хаара,* алгоритм Дейкстри.

ABSTRACT

of the Master's Thesis

«Vehicle navigation based on traffic analysis using computer vision»

Student of group 608m: Gubarev Mykhailo Oleksandrovych

Supervisor: Ph.D., associate professor Boiko A.P.

The master's thesis is devoted to the development of a web-based vehicle monitoring service using computer vision. The work aims: to automate the monitoring process of the road's state congestion by creating a network of dynamic surveillance cameras and developing an adaptive web application for drivers that will improve the conditions for driving.

The master's thesis consists of an introduction, four chapters, the special part on labor protection, conclusions, and appendices. The introduction defines the relevance of the topic, what is taken as the goal and a brief overview of the task, the subject and the object of research.

The object of research is the process of road traffic analysis using computer vision technologies.

The subject of the research is data analysis technologies for vehicle navigation in conditions of high congestion on city roads during peak hours.

The first chapter describes an overview of existing analogues of navigation systems. It also describes the specification of requirements for the web service. The second chapter describes the modeling of a web service for video analysis using computer vision. The third chapter describes the main technologies for developing and designing a web-based navigation service. The fourth chapter describes the development of the servers' prototype and adaptive interface of the web application, it`s testing, and user manuals. The conclusions analyze the work and the results obtained. As the results of the research, the web service was developed for driver navigation based on the cars` traffic monitoring using computer vision.

The thesis contains 99 pages of the main part, 75 figures, 8 tables, and 57 references.

Keywords: *navigation system, computer vision, video surveillance, Haar cascades, Dijkstra algorithm.*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ	–	Програмне забезпечення
ТЗ	–	Транспортний засіб
CBOC	–	Composite Binary Offset Carrier
СКБД	–	Система керування базами дани
AQI	–	Air quality index
DVR	–	Digital Video Recorder
GPS	–	Global Positioning System
GNSS	–	Global Navigation Satellite System
NVR	–	Network Video Recorder
OTA	–	Over The Air
POI	–	Point of interest
SAN	–	Storage Area Network
UI	–	User Interface
UML	–	Unified Modeling Language

ВСТУП

Сучасні виклики та посилена урбанізація призводять до необхідності пошуку технологічних рішень щодо покращення руху на міських вулицях та підвищення їх безпеки. Створити інтелектуальне управління дорожнім рухом можна за допомогою технологічних рішень, таких як інформаційний вебзастосунок, який буде аналізувати трафік та розраховувати відстань руху автомобіля у залежності від поточної дорожньої ситуації. Таким чином, регулювати транспортний потік у містах можна буде за поточною дорожньою ситуацією щодо завантаженості шляхів руху за допомогою системи, яка швидко оброблятиме дані, які до неї надходитимуть, і забезпечуватиме зворотній зв'язок, наприклад, у вигляді інформації на панелі керування.

Для модернізації інфраструктури міст є дуже важливим створення ефективних шляхів руху. Завдяки цим технологіям навігаційна система буде збирати й аналізувати дані, що сприятиме адаптації до довгострокових транспортних потреб та покращенню управлінням щоденним дорожнім рухом. Аналізувати дані можна практично у реальному часі за допомогою датчиків, камер і радарів, які використовуються для розчищення людних вулиць та покращення руху транспорту. У хмарне сховище можна завантажити дані для довгострокового аналізу, наприклад, щодо покращення дорожніх умов.

Таким чином, у транспортних системах міст інтелектуальний підрахунок транспортних засобів має важливе значення. За допомогою відео зі звичайних камер спостереження у режимі реального часу на шосе та міських дорогах алгоритми глибокого навчання можуть класифікувати типи транспортних засобів та підраховувати їх у різних погодних та транспортних умовах, що є ключовою стратегією аналізу трафіку. За допомогою вже встановлених або нещодавно встановлених камер спостереження дана система підрахунку транспортних засобів отримує дані для аналізу.

Основними перевагами підрахунку транспортних засобів є: надійні алгоритми глибокого навчання, що забезпечують високу точність у задачах виявлення та класифікації об'єктів, автоматичне та безконтактне розпізнавання декількох типів транспортних засобів (автомобіль, автобус, вантажівка, велосипед), можливість відстеження кількох виявлених об'єктів та їх підрахунок у режимі реального часу, коли вони проходять певну область ділянки дороги, конфіденційність та надійність даних.

Отже, **актуальність** даного дослідження полягає в тому, що підрахунок транспортних засобів за допомогою технологій комп'ютерного зору забезпечує ефективний моніторинг транспортних засобів у кількох географічно розподілених місцях та повністю автоматизований аналіз трафіку у режимі реального часу, що дозволяє виявляти аномалії, небезпечні події, здійснювати моніторинг годин пік, вузьких місць за допомогою відео звичайних камер відеоспостереження. Таким чином, можна здійснити оцінку ефективності впливу запроваджених заходів, націлених на зменшення пробок на дорогах чи зниження міського трафіку, та відстежувати зміни у транспортному потоці.

Метою кваліфікаційної роботи магістра є автоматизація процесу моніторингу стану завантаженості автомобільних доріг за рахунок створення мережі динамічних відеокамер спостереження та розроблення адаптивного вебзастосунку для водіїв, що покращить умови для керування транспортними засобами.

Об'єктом дослідження роботи є процес аналізу дорожнього трафіку з використанням технологій комп'ютерного зору.

Предметом дослідження є технології аналізу даних для навігації транспортних засобів в умовах високої завантаженості дорожніх шляхів міст у часи пік.

Методи дослідження: методи об'єктно-орієнтованого аналізу та проектування для створення програмного забезпечення розпізнавання транспортних об'єктів, методи

забезпечення надійності програмного забезпечення, гібридний вид методології гнучкої розробки (agile methodology) [1].

Для досягнення поставленої мети необхідно розв'язати наступні завдання:

- проаналізувати існуючі аналоги навігаційних систем, визначити їх основні переваги та недоліки;
- розробити інформаційну модель інформаційно-аналітичної системи;
- нормалізувати відносини сутностей у системі керування базами даних;
- розробити програмні засоби для динамічної мережі відеомоніторингу;
- розробити програмні засоби для асистування водієві під час руху;
- проаналізувати отримані результати та виробити рекомендації для їх практичного застосування.

Для досягнення мети необхідно проаналізувати та реалізувати наступні алгоритми:

- алгоритм розпізнавання автомобілів;
- алгоритм вибору найкращого маршруту руху містом.

Практичне значення роботи: розроблено алгоритм для передачі даних та сповіщення про завантаженість. За допомогою реалізованої технології розпізнавання завантаженості автомобільного трафіку дозволятиме навігаційній системі збирати та аналізувати дані, що, у свою чергу, призведе до поліпшення повсякденного керування дорожнім рухом та адаптації до довгострокових потреб у транспорті.

Апробація результатів дослідження. Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія - 2023».

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ НАВІГАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ НА ОСНОВІ АНАЛІЗУ ТРАФІКУ

1.1 Загальний аналіз існуючих систем навігації

Навігаційна система – це обчислювальна система, яка допомагає у навігації. Навігаційні системи можуть бути вбудовані у бортовий комп'ютер або мультимедійну систему транспортного засобу чи бути автономними, тобто розташованими в іншому місці авто. Дані системи можуть використовувати супутникові, радіо або інші канали передачі сигналу для керування транспортним засобом. У деяких випадках використовують комбінацію цих методів [2]. Існують такі типи навігаційних систем: радіонавігація, супутникова система навігації, пульсарна та інерційна навігації (використовується переважно у кораблях, літаках та ракетах).

Автомобільна навігаційна система – це комп'ютерний картографічний пристрій, призначений для того, щоб допомогти водіям знайти місце призначення. Деякі з цих пристроїв вбудовані в автомобіль, а інші можна придбати окремо і встановити в автомобіль. Також можна інтегрувати смартфон в автомобіль за допомогою адаптерів Apple CarPlay або Android Auto чи скористатися Automirroring-адаптером. Тобто всі можливості iPhone або Android-пристрою доступні на екрані мультимедійної системи машини: музика, відео, навігація, інтернет-браузер тощо (окрім дзвінків) [3].

Переважна більшість автомобільних навігаційних систем покладаються на супутникові сигнали, безконтактні маяки забезпечують ще одну форму радіопозиціонування. Використання для позиціонування радіомаяків є доцільним у якості резервної системи навігації у випадку коли інші системи недоступні або відмовили, оскільки дана система має чисельні похибки.

Ще один вид навігації – пульсарна навігація, яка також не може замінити супутникову навігацію. Дана технологія, так само як інерційна, має ряд недоліків і може застосовуватись лише для визначення положення літаків і морських суден.

Супутникова навігація, або глобальна навігаційна супутникова система (англ. GNSS – Global Navigation Satellite System) – це система супутників, що передає дані про глобальне позиціонування та точний час (рис. 1.1).

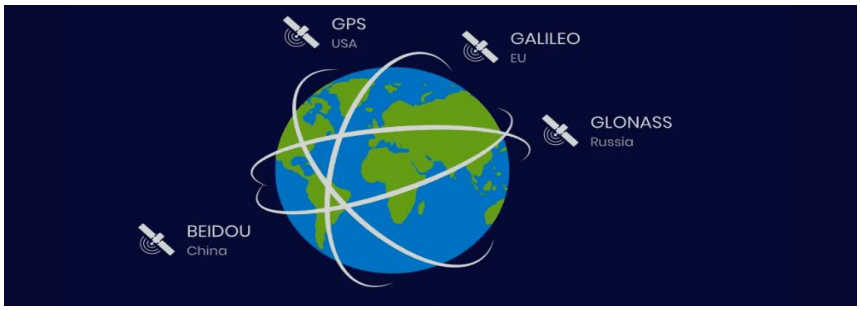

Рисунок 1.1 – Система супутників GNSS

Найвідоміша супутникова навігаційна система Global Positioning System (GPS) є частиною комплексу NAVSTAR, спочатку була розроблена для збройних сил США. Повністю доступною широкому загалу стала в 2000 році. Американські військові підтримують від 24 до 31 супутника, що обертаються навколо Землі будь-якої миті часу, які використовуються для системи GPS. Приймач GPS вимірює відстань між собою та супутником, знаходячи час, витрачений на те, щоб радіосигнал передався від супутника до пристрою [4].

GPS – не єдина на даний час система супутникової навігації. Альтернативною до GPS є глобальна навігаційна супутникова система (GNSS) Galileo, запущена в 2016 році, створена Європейським союзом з метою створення незалежної європейської високоточної системи позиціонування. Використання основних послуг Galileo є безкоштовним та відкритим для всіх авторизованих урядом користувачів. Очікується, що Galileo буде сумісним із модернізованою системою GPS. Приймачі зможуть комбінувати сигнали із супутників Galileo та GPS, щоб значно підвищити точність [5].

BeiDou (BDS, ран. Compass) – це глобальна навігаційна система, розгорнута та керована Китаєм та призначена для використання тільки у даній країні. Характерною особливістю системи є невелика кількість супутників, перебуваючих на геостаціонарній орбіті. З 16 навігаційних супутників виведених на орбіту Землі лише 11 використовується за призначенням. З грудня 2012 року система пропонує послуги клієнтам в Азіатсько-Тихоокеанському регіоні [6].

GLONASS – глобальна навігаційна система, розгорнута та експлуатована Росією та належить міністерству оборони даної країни. У 2011 році була

повністю поновлена та до 2025 року планується повна модернізація. Характерною особливістю системи є невелика розповсюдженість клієнтського обладнання [7].

Michibiki (QZSS) – регіональна навігаційна система, розгорнута та експлуатована Японією, поширена у регіонах Азії та Океанії з акцентом на Японію. Створена і розроблена японським урядом для поліпшення керованої США Глобальної системи позиціонування (GPS) у даному регіоні. Метою QZSS є надання високоточних і стабільних послуг позиціонування в регіоні Азія-Океанія, сумісних із GPS [8].

NavIC (або IRNSS) – регіональна навігаційна система, розгорнута та експлуатована Індією, поширена у регіонах Південної та Західної Азії, що знаходиться у стані розробки. NavIC надає два рівні обслуговування: «стандартну службу позиціонування», яка відкрита для цивільного використання, та «обмежена службова» (шифрована) для авторизованих користувачів (включаючи військових). Трекери на основі NavIC є обов'язковими для комерційних автомобілів в Індії та для деяких мобільних телефонів [9].

System.NET – мережа референтних станцій у складі наземної складової навігаційної системи GNSS, створеної в Україні у 2012 році. Може приймати досить широкий спектр сигналів глобальних супутникових навігаційних систем та регіональних систем SBAS. Дані, що отримуються з мережі System.NET можуть використовуватись для високоточної навігації транспорту та інших задач [10].

Січ-2-30 (розроблявся під назвою – Січ-2-1) – український супутник оптико-електронного спостереження Землі. Запущений ракетою-носієм Falcon 9 компанії SpaceX з мису Канаверал у Флориді (США) у 2022 році. Знімає поверхню Землі у видимому й інфрачервоному діапазонах. Через невисоку роздільну здатність отримуваних знімків супутник не може бути запорукою повної інформаційної незалежності країни. Таким чином, Україна є залежною від інших постачальників картографічних знімків та на сьогоднішній день існує гостра необхідність розвитку власної української навігаційної системи для отримання цифрових зображень поверхні Землі для проведення постійного безперервного моніторингу різних явищ [11].

На території України для корекції GPS-даних з наземних базових станцій використовуються методи коригування DGPS і RTK. У високоякісному обладнанні і професійному програмному забезпеченні є досить ефективним використання методів корекції отриманих даних DGPS і RTK [12]. Але отримання поправок у реальному часі є платною послугою, що є головним недоліком у їх використанні.

Проаналізувавши типи навігаційних систем можна зробити висновок, що у наземних транспортних засобах для знаходження місця розташування об'єктів використовується

переважно супутникова навігаційна система як більш точна, рідше – у поєднанні з радіонавігацією у якості резервної системи навігації. Варто зауважити, що супутникові навігаційні системи мають свої переваги та недоліки. До переваг можна віднести – високу точність, доступність, простоту використання, до недоліків – чутливість сигналу до природних та штучних перешкод, необхідність оновлювати ПЗ, залежність від підключення до мережі Інтернет.

Отже, система навігації може поєднувати в собі декілька різних типів технологій від яких отримує інформацію про стан дорожнього руху в режимі реального часу для позиціонування транспортного засобу в просторі та в часі, а також для визначення параметрів руху (швидкості, напрямку та ін.). Використовуючи цю інформацію, система може порівняти кілька маршрутів та знайти найшвидший. Потім вона може продовжити моніторинг, щоб проаналізувати, чи стане один із маршрутів швидше на основі трафіку, і відповідним чином налаштуватися. Системи навігації забезпечують контроль та діагностику стану інфраструктурних мереж міста, що забезпечує ефективне управління транспортним засобом та зменшує завантаженість доріг.

1.2 Аналіз програмних засобів для асистування водієві під час руху

У автомобілях використовується широкий спектр навігаційних систем, але незалежно від їх особливостей вони діляться на дві категорії: стаціонарні та інтегровані. Інтегровані системи навігації – це автомобільні комп’ютери, а також мультимедійні центри, мобільні пристрої, що мають функцію навігації. Стаціонарні навігаційні системи – це пристрої, які вбудовані в автомобіль. До них відносяться навігаційні системи на основі бортового комп’ютера автомобіля, які зустрічаються переважно в автомобілях преміум-класу. Хоча ці два типи навігаційних систем існують для надання одних і тих самих послуг, вони різняться за своїми особливостями, і у кожної з них є свої плюси та мінуси. Розглянемо їх нижче.

Автономність. Стаціонарна навігаційна система у автомобілі вже має всі необхідні карти. Звичайно, інтегрована навігаційна система також може мати автономні карти, але це відбувається шляхом зберігання і завантаження карти певної країни або регіону, що буде займати місце на смартфоні або у хмарному сховищі, а також це буде коштувати певну суму за передачу даних при завантаженні.

Ціна встановлення. Водієві доведеться заплатити додаткові гроші виробнику, щоб вони встановили систему навігації у його автомобілі, яка потенційно включає додаткові функції, які насправді не завжди потрібні. З іншого боку, можна придбати автономну навігаційну систему за набагато нижчою ціною та встановити її самостійно.

Вартість використання. Стаціонарні навігаційні системи безкоштовні, за них не потрібно платити абонентську плату, щоб продовжувати користуватись навігаційною системою. На інтегровану навігацію теж необхідно витрачати певні кошти, оскільки вони покладаються на стільникові дані для отримання інформації.

Конфіденційність. Більшу конфіденційність забезпечують стаціонарні навігаційні системи. Інтегровані навігаційні системи використовують кілька сторонніх сервісів для

покращення своїх послуг, і ці сторонні сервіси можуть записувати та збирати дані для різних цілей.

Акумулятор. Стаціонарні навігаційні системи споживають енергію безпосередньо з автомобіля. Більшість портативних навігаційних систем використовують батареї для живлення. Їм потрібно більше енергії, оскільки вони часто використовують стільникові дані для завантаження карт і навігаційної інформації. Стаціонарні системи залишаються включеними до тих пір, поки в баку є паливо, і навіть якщо воно закінчиться, навігаційна система працюватиме доти, доки працює акумулятор автомобіля.

Зручність. У деяких нових автомобілях у мультимедійні пристрої встановлені системи голосового керування навігацією, але більшість машин мають статичні карти. У портативних навігаційних системах наявні всі необхідні функції та специфікації. Крім того, більшість навігаційних систем універсальні, і їх можна демонтувати з одного автомобіля і помістити в інший.

Оновлення та старіння карт. Стаціонарні навігаційні системи важко оновити, оскільки вони зазвичай постачаються з SD-картою, що містить карти та інші дані. Також виробник може припинити оновлення навігаційних систем даної моделі автомобіля через кілька років. Портативні навігаційні системи майже завжди підключені до Інтернету, а це означає, що вони завжди є актуальними. Також у більшості з них є операційна система, яка полегшує процес оновлення, тому у авто завжди є актуальні карти та навігація [13].

Таким чином, стаціонарні та інтегровані навігаційні системи мають свої переваги та недоліки. Стаціонарні навігаційні системи дорожчі, але вони надійніші з погляду апаратного забезпечення. Інтегровані системи дешевші в установці і завжди актуальні, але вимагають доступ мережі Інтернет та споживають заряд батареї [14]. Однак, із розвитком технологій з'явилась можливість інтегрувати мобільні пристрої у мультимедійну систему транспортного засобу, що в режимі реального часу надає карту поточного місцезнаходження користувача та покрокові вказівки як дістатися до заданого місця призначення. Частіше за все водії використовують такі програми як Google Maps, Waze та Apple Maps, які підключаються через мобільний телефон у мультимедійну систему автомобіля за допомогою застосунків для відображення функцій пристрою – CarPlay та Android Auto (рис. 1.2) [15].

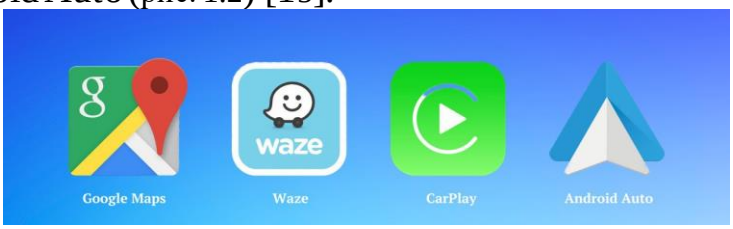

Рисунок 1.2 – Іконки застосунків для навігації

Навігаційні програми, які конкурують або перевершують якість автомобільних навігаційних систем, є стандартними на багатьох смартфонах. Apple представила CarPlay у 2014 році. CarPlay дозволяє відображати, серед іншого, навігацію з підключеного iPhone та взаємодіяти за допомогою дисплея та елементів керування автомобіля (рис. 1.3).

Рисунок 1.3 – Вигляд головного екрана Apple Carplay під час взаємодії з мультимедійною системою автомобіля

Android представив Android Auto (рис. 1.4) для смартфонів Android як прямий конкурент Apple CarPlay. Обидві системи використовують супутникові сигнали та датчики підключеного телефону для навігаційних даних, які потім передаються до системи автомобіля для відображення. Дані про трафік у режимі реального часу можуть бути завантажені за допомогою стільникових даних телефону залежно від програми, яка використовується для навігації.

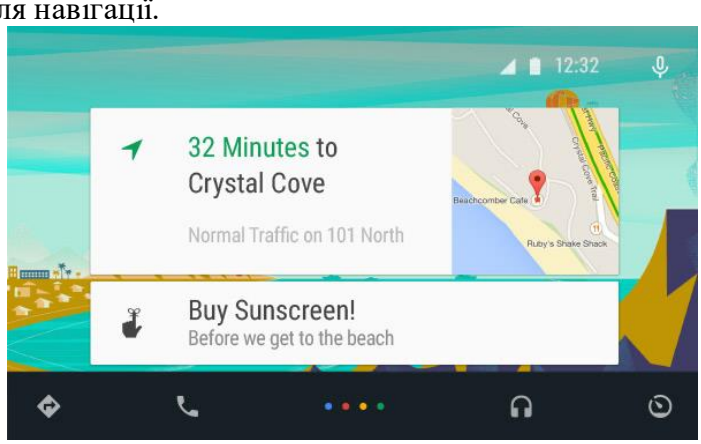

Рисунок 1.4 – Вигляд головного екрана Android Auto

Щоб відповідати інтересам клієнтів та високій якості інтегрованих навігаційних програм від Google, Apple та Microsoft, автовиробники повинні пропонувати таку ж розширену функціональність та мобільність, як і свої вбудовані навігаційні рішення [16]. Таким чином, майбутнє навігації залежатиме від її інтеграції з іншими автомобільними системами та подальшого розвитку різноманітних технологій, реалізованих у транспортних засобах наступного покоління таких як штучний інтелект, комп’ютерний зір, когнітивні обчислення, доповнена реальність, IoT, Big Data та інші пов’язані з ними інновації, які незабаром призведуть до появи безпілотних автомобілів як основного засобу пересування.

Отже, для навігації транспортних засобів існують вбудовані та портативні системи, які мають свої переваги та недоліки. Стаціонарні навігаційні системи дорогі, але надійні. Інтегровані системи дешевші, завжди актуальні, але вимагають доступ мережі Інтернет і споживають заряд батареї. Можливість інтегрувати мобільний пристрій у мультимедійну систему транспортного засобу вирішує проблему навігації транспортного засобу в режимі реального часу. Сучасні програмні засоби для асистування водієві під час руху змінюються відповідно до нових тенденцій. Впровадження інноваційних технологій змінює всю навігаційну концепцію, покращуючи керівництво та інтеграцію з іншими інноваційними функціями, реалізованими в автомобілі.

1.3 Детальний огляд систем навігації транспортних засобів на основі аналізу трафіку

Існує багато програм для навігації, доступних для вільного користування. Більшість навігаційних програм діють однаково, але кожна з них має функції, які відрізняють її від конкурентів. Отже, розглянемо деякі з найкращих навігаційних програм, щоб порівняти їх плюси та мінуси, а саме Google Maps, Waze, Apple Maps.

Google Maps – це картографічна та навігаційна програма для настільних та мобільних пристроїв від Google. Пропонує інформацію про трафік у режимі реального часу, потокову передачу музики та інтеграцію з Google Assistant, обмін ходом поїздок та вкладку «explore» на основі штучного інтелекту. Завдяки багатьом функціям (рис. 1.5) Google Maps є найпопулярнішою серед безкоштовних картографічних програм. На картах Google Maps не можна додати більше 10 зупинок до свого маршруту.

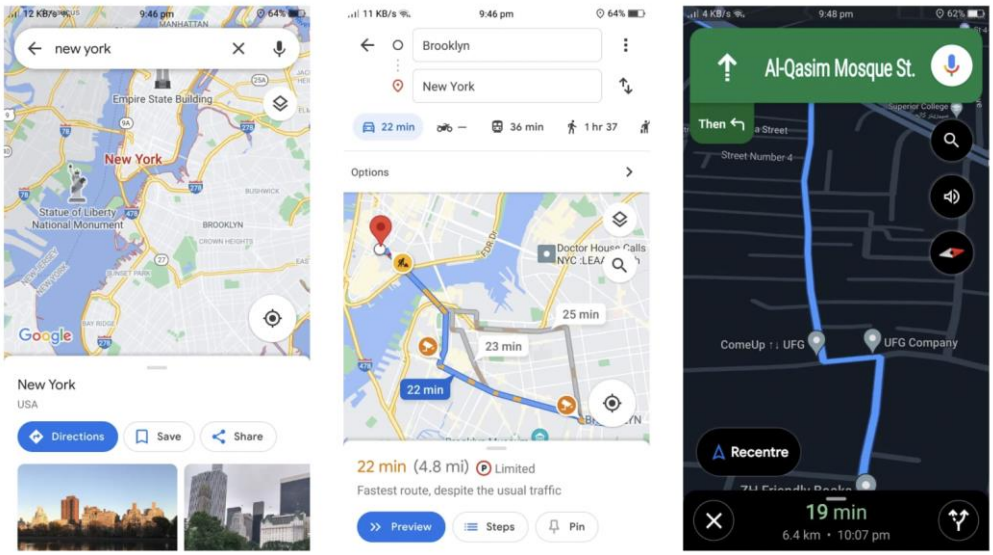

Рисунок 1.5 – Інтерфейс застосунку Google Maps

Плюси: надійні карти з опціями Google Earth та Street View, дані про трафік у режимі реального часу, дозволяє завантажувати карти для автономного використання, будує найбільш ефективний маршрут для пересування пішки, велосипедної та автомобільної їзди, пропонує внутрішнє планування будівель, має веб-версію.

Мінуси: GPS швидко розряджає батарею телефону, Google Maps може створити маршрут лише з 10 зупинками, включаючи початкову точку, збирає дані користувача, немає функції обміну соціальними мережами.

Використовуючи машинне навчання, Google Maps зберігає свою базову карту оновленою та відносно безпомилковою. За допомогою штучного інтелекту аналізуються дані про інтенсивний трафік в режимі реального часу. Ще однією ключовою особливістю даної програми є Google Street View, який показує панорамний вид на вулицю. Таким чином, Google Maps є найкращою оновлюваною програмою, яка аналізує дані про трафік у режимі реального часу [17].

Waze – картографічна та навігаційна програма для настільних та мобільних пристроїв від Google. Waze пропонує оновлення трафіку в режимі реального часу. Програма надсилає поточні дані про трафік у хмарне сховище, щоб повільний трафік та пробки враховувалися під час розрахунку маршруту для інших користувачів Waze, що подорожують у схожому напрямку. Має більш просунуті дані в режимі реального часу про стан дорожнього руху, закриття доріг і т.д. Waze пропонує простий та мінімалістичний інтерфейс з високим рівнем можливостей для налаштування (рис. 1.6). Waze також не дозволяє здійснювати необмежені доповнення маршруту.

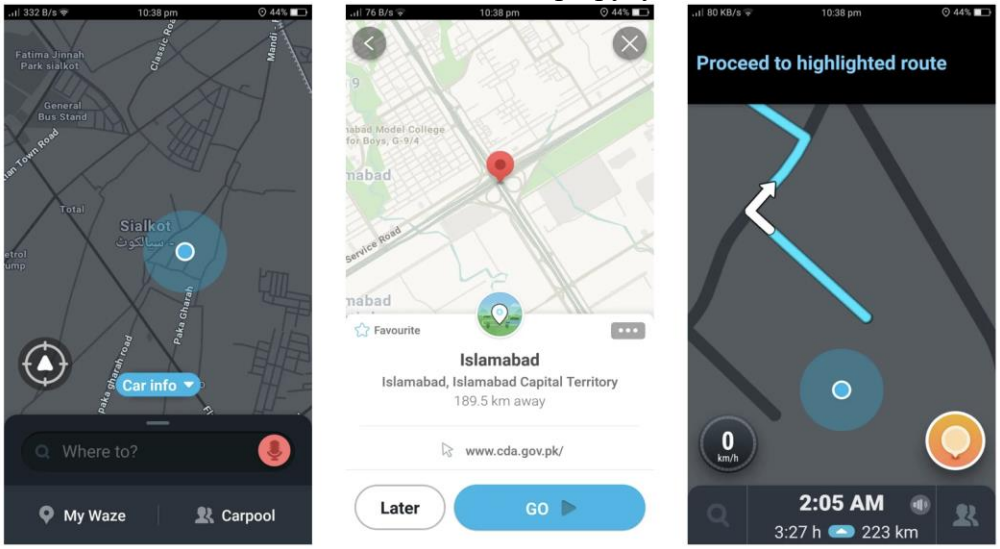

Рисунок 1.6 – Інтерфейс застосунку Waze

Плюси: база спільноти користувачів – найсильніша функція Waze, Waze дозволяє користувачам ділитися поточними цінами на паливо, додаток також дозволяє користувачам підключатися до Facebook та Google Calendar, допомагає знайти адреси подій та створити маршрути, має вебверсію.

Мінуси: потужність батареї та ресурси даних, які споживає Waze, високі завдяки численним іконкам та великому обміну даними, важко знайти напрямки з місць, які не є поточним розташуванням користувача, у дизайні Waze є візуальна невпорядкованість, що ускладнює сприйняття та розуміння функціоналу тих чи інших кнопок, відсутній рельєф місцевості, супутник або 3D-картографічні зображення, немає автономних опцій.

Waze – це спільнота користувачів-водіїв, які допомагають оновлювати інформацію у додатку в реальному часі, підтримують один одного у випадку аварії та точно вказують на карті місце, де вона відбулась. Використовуючи Waze, можна уникнути пробок, тому що

програма миттєво змінює маршрут у режимі реального часу [18]. Таким чином, програма має найактуальніші дані для оповіщень інших користувачів про аварії, будівництво і поліцейських та підходить для тривалих поїздок.

Apple Maps – це картографічний вебсервіс, розроблений Apple Inc. Дана картографічна система будує маршрути та розраховує час прибуття до місця призначення для різних способів пересування: ходьби, їзди на велосипеді, автомобілі, у громадському транспорті.

Функції Apple Maps включають покрокові голосові підказки, інтерактивні 3D-карти і попереджувальні повідомлення щодо вибору смуги руху. Apple Maps може автоматично запам’ятовувати місце паркування, на якому користувач залишив свій автомобіль, а також обирати на карті громадський транспорт, автосервіси і пішохідні прогулянки. Apple Maps пропонує фотографічний, концептуальний вигляд, який може зробити його використання більш привабливим та захоплюючим (рис. 1.7).

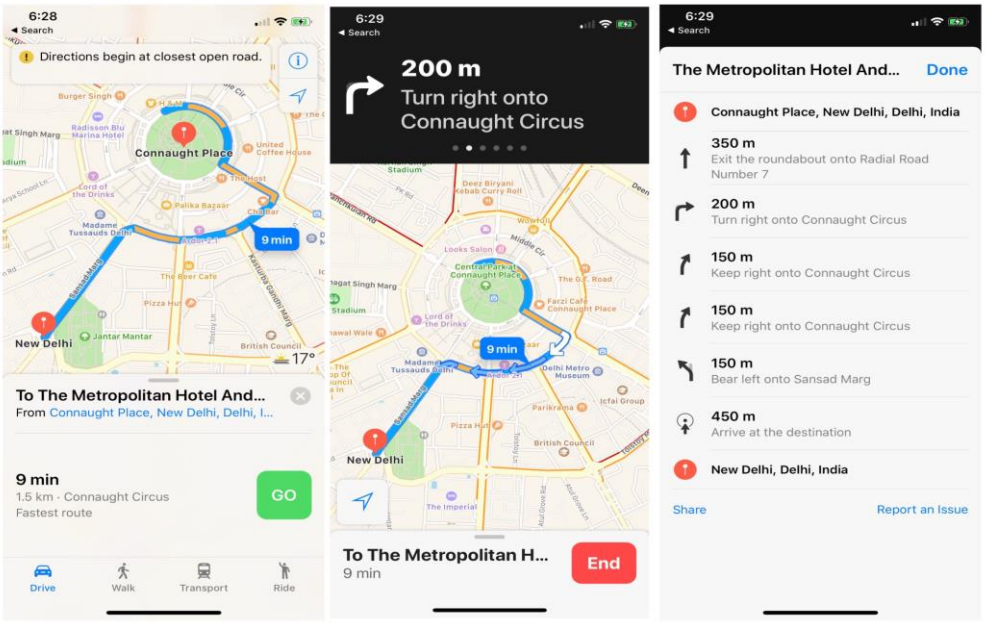

Рисунок 1.7 – Інтерфейс застосунку Apple Maps

Плюси: відмінний дизайн та візуальний стиль, віртуальний гід: наявність інструменту Flyover для віртуального туру місцем призначення, надає такі функції, як покроковий маршрут та голосова навігація, Apple Maps показує погоду та AQI (англ. Air quality index – індекс якості повітря) для вибраного місця розташування.

Мінуси: існують неточності у інформації про об’єкти, культурні пам’ятки на картах можуть бути відсутніми в порівнянні з іншими програмами, немає можливості автономного використання карт, функція Look Around Street View доступна лише у 5 країнах: Канаді, Ірландії, Японії, Сполучених Штатах та Великобританії – всього з кількома зонами покриття в кожній країні, немає вебверсії, немає автономних опцій, доступність використання тільки для iOS.

На відміну від Google Maps та Waze, орієнтованого на водія, Apple Maps не має браузерної версії. Проте Apple випустила свій JS-код MapKit, який дозволяє веброзробникам додавати сервіси Apple Maps на вебсайти. У цілому, Apple Maps, як і раніше, відстає від Google Maps через кілька відсутніх функцій, але компанія робить все можливе, щоб наздогнати і перевершити Google [19]. Отже, Apple Maps є найкращою з точки зору конфіденційності та найкраще реалізує свої функції у пристроях компанії Apple. Кожна з розглянутих програм має свої сильні та слабкі сторони (таблиця 1.1).

Таблиця 1.1 – Порівняння навігаційних програм

ПЗ	Google Maps	Apple Maps	Waze
Логотип			
Основні переваги	-зручний; -надійний; -часто оновлюваний	-інтеграція з пристроями Apple; -конфіденційність; -простий дизайн	-оновлювані дані в режимі реального часу; -контрольшвидкості; -сповіщення про паркування
Доступність на платформах:	PC, Mac, Android iOS	iOS	PC, Mac, Android, iOS
Трафік у реальному часі	+	+	+
Панорами вулиць	+	+	–
3D-зображення	+	+	–
Завантаження карт для автономної роботи	+	–	–
Вебверсія	+	–	+

Функціональні особливості	-планування маршруту; -помічник Google; -оновлення трафіку в реальному часі	-карти Apple з підтримкою GPS; -інтеграція з пристроєм Apple; -функція Look Around	-навігація по карті; -оновлення в режимі реального часу; -сповіщення про ситуацію на дорозі

Google Maps та Apple Maps можуть попереджати водіїв про аварії та пробки, але дані повіщення загальніші, ніж Waze, тому що ці програми не мають такого високого ступеня соціальної інтеграції. Тим не менш, Google Maps та Apple Maps мають більш широкий та інформативний вид карт, які часто чіткіше показують маршрути та навколишні сервіси. Google Maps та Apple Maps мають функції доповненої реальності, які найкраще підходять для пішоходів. Waze – чудовий інструмент навігації для будь-якого водія, який шукає поліцейські пастки, оновлення дорожнього руху та звіти про аварії [20].

Таким чином можна зробити висновок, що основна різниця між програмами полягає у тому, що Google Maps та Apple Maps використовують GPS-навігацію, тобто отримують дані тільки з супутників, у той час як Waze прокладає маршрути використовуючи інформацію, надану спільнотою Waze. Але головними недоліками Apple Maps та Waze, на відміну від Google Maps, є те, що вони не мають функції завантаження карт для автономної роботи.

1.4 Специфікації вимог до програмного забезпечення

Призначення та межі проєкту. Призначення ПЗ: даний вебсервіс створено для автоматизації процесу моніторингу стану завантаженості автомобільних доріг за рахунок створення мережі динамічних відеокамер спостереження та розроблення адаптивного вебзастосунку для водіїв, що покращить умови для керування транспортними засобами

Погодження, що ухвалені в програмній документації. Було погоджено, що для створення загального програмного забезпечення та його злагодженої роботи будуть використовуватися допоміжні фреймворки – Flask (Python), Jinja2 (Python), OpenCV (Python), NumPy(Python), JQuery (JavaScript), Bootstrap (JavaScript, CSS3)

Межі проєкту ПЗ. Крайня дата завершення роботи над вебсервісом – 12.02.2023р.

Загальний опис

Сфера застосування. Даний вебсервіс не має особливих обмежень у сферах та місцях його застосування, але рекомендовано використовувати за призначенням, а саме для навігації транспортних засобів.

Характеристики користувачів. Основні характеристики користувачів: водії (наявність транспортного засобу, смартфону (девайсу з камерою) та Інтернет) та адміністратори (вебкамери під'єднані до серверу відеоспостереження, смартфон або девайсу з камерою та Інтернет).

Загальна структура і склад системи. Основні частини для створення програмного забезпечення: мережа вебкамер, сервер; база даних; вебзастосунок.

Загальні обмеження. Обмеженням для роботи з ПЗ є: наявність Інтернету, роздільна здатність вебкамери, протоколи передачі відеоданих, обчислювальні потужності вебсерверу, обчислювальні потужності смартфону, продуктивність роботи програмних фреймворків, швидкість роботи СКБД.

Функції вебсервісу навігації транспортних засобів.

Функція прокладання маршруту. Опис функції: функція прокладання маршруту показує найкращий маршрут від місця знаходження водія до заданої кінцевого місця прямування транспортного засобу з огляду на завантаженість доріг.

Вхідна і вихідна інформація. Вхідна інформація: обраний пункт прямування транспортного засобу, дані про відеомоніторинг доріг. Вихідна інформація – прокладений найкращий маршрут з огляду на завантаженість доріг до місця прямування, яке обрав водій.

Функціональні вимоги. База даних з інформацією з вебкамер про завантаженість доріг та мережа Інтернет.

Вимоги до програмного забезпечення.

Джерела і зміст вхідної інформації (даних): у даному програмному забезпеченні джерелом вхідної інформації є користувач та мережа вебкамер. Користувач вручну задає необхідні дані, в даному випадку – це кінцева точка прямування транспортного засобу. Джерелом отримання даних про автомобільний рух вулицями міста є мережа вебкамер.

Нормативно-довідкова інформація (класифікатори, довідники тощо): інструкція з використання.

Вимоги до способів організації, збереження та введення інформації: обмін даними повинен відбуватися використовуючи REST API, у якості СКБД для програмного забезпечення обрати MySQL.

Вимоги до технічного забезпечення: комп'ютер чи ноутбук з процесором вище INTEL Core i3 6100, цього процесору достатньо як для підтримки роботи ПЗ, так і для розробки ПЗ.

Вимоги до програмного забезпечення.

Архітектура програмного забезпечення є трирівневою і складається з клієнтської частини, серверної частини та СКБД. Обмін даними повинен відбуватися використовуючи REST API.

Системне програмне забезпечення. Серверна частина програмної системи побудована за допомогою Flask. Для написання фронтенду обрано Javascript-фреймворк JQuery.

Мережеве програмне забезпечення. Вебінтерфейс Pythonanywhere для керування вебхостингом Flask застосунку. Браузер – для зручного перегляду вебсторінок.

Програмне забезпечення ведення інформаційної бази. Все ведення інформаційної бази має відбуватися через SQLAlchemy. В якості СКБД для програмного забезпечення обрано MySQL.

Мова і технологія розробки ПЗ. Програмне забезпечення повинно бути розроблене за допомогою фреймворків: Flask (Python), Jinja2 (Python), OpenCV (Python), NumPy(Python), JQuery (JavaScript), Bootstrap (JavaScript, CSS3).

Вимоги до зовнішніх інтерфейсів.

Інтерфейс користувача. Вебінтерфейс повинен задовольняти усім вимогам UX/UI, що дозволить користувачу затратити найменше часу на розуміння роботи системи. Шаблон вебзастосунка повинен бути розділений на дві частини, де перша частина це пошукова панель, де розміщені фільтри для пошуку, вона розміщена зліва, друга частина є основний контент який змінюється відповідно до заданих фільтрів.

Апаратний інтерфейс. Апаратний інтерфейс мережевих вебкамер, за якими камери мають взаємодіяти з хостом для трансферу відео до серверної частини.

Програмний інтерфейс. Зовнішні API не використовується, але серверна частина може мати RESTful API для взаємодії з клієнтськими вебсервісами.

Комунікаційний протокол. Серверна частина може взаємодіяти за REST протоколом з клієнтськими вебсервісами. Клієнтські вебсервіси та користувачі для використання функцій застосунку мають використовувати URL на базі HTTP протоколу.

Властивості програмного забезпечення.

Доступність. Будь-який користувач, який бажає користуватися програмним забезпеченням може використати URL вебзастосунку.

Супроводжуваність. Супроводжується системним адміністратором.

Переносимість. Програмне забезпечення може працювати на будь-якій операційній системі та будь-якому смартфоні, де присутні браузери.

Продуктивність. Продуктивність залежить від швидкості: мережі Інтернет, конвертації кадрів відео для опрацювання, розпізнавання об’єктів, роботи SQL запитів у СКБД, алгоритмів побудови маршруту.

Продуктивність застосунка повинна контролюватися часом виконання запитів, що не повинні перевищувати 2 секунд.

Надійність. Усі дані, які вказуються користувачем повинні бути приватними і виключити можливості їх отримання будь якими способам для інших користувачів. Кожний користувач може отримати доступ до своїх даних лише авторизуватися у системі.

Безпека. На сервері можлива генерація SSL-сертифікату для встановлення захищеного з’єднання між браузером клієнта та сервером.

Інші вимоги. Усі вимоги описані вище.

Висновки до розділу 1

Отже, в першому розділі проведено огляд існуючих систем навігації, розглянуто типи систем навігації, їх переваги та недоліки. На даний момент найпоширенішим засобом орієнтування на місцевості є GPS-навігатори, вбудовані майже в кожен пристрій. Але якість позиціонування за певних умов у даних системах недостатня. Крім недостатньої точності існуючі системи навігації змушують водіїв відволікатись від управління транспортним засобом. Більш точні та функціональні системи навігації розробляються за допомогою сучасних технологій отримання та обробки інформації. Тому актуальним є створення нових більш функціональних та точних методів навігації.

Також було сформульовано задачі досліджень дипломної роботи та основних вимог щодо створення програмного забезпечення. Вибір було зроблено у сторону ПЗ на основі аналізу міського трафіку, що матиме трирівневу архітектуру та складатиметься з клієнтської, серверної частин та СКБД (обрано MySQL). Обмін даними у даній систем буде здійснюватись з використанням REST API, а ведення інформаційної бази відбуватиметься через SQLAlchemy. Серверна частина програмної системи буде побудована за допомогою Flask, фронтенд ПЗ буде написано з використанням Javascript-фреймворку JQuery.

2 МОДЕЛЮВАННЯ ВЕБСЕРВІСУ АНАЛІЗУ ВІДЕО ЗА ДОПОМОГОЮ КОМП'ЮТЕРНОГО ЗОРУ

2.1 Алгоритм накопичення даних розподіленої мережі вебкамер

З розвитком технологій вимоги до розгортання систем відеоспостереження стали більш високими. Тенденція розвитку систем відеоспостереження рухається до збільшення кількості пристроїв моніторингу, якість відео постійно покращується, а час зберігання відео збільшується. Усі ці зміни швидко збільшують обсяги даних, які створюються системою відеоспостереження [21]. Зазвичай система відеоспостереження – це сукупність програмних та апаратних засобів, каналів комунікації, що передають візуальну інформацію про стан об'єкта у реальному часі. Дані системи здатні обробляти відеопотік з його подальшим перетворенням, накопиченням, зберіганням. При необхідності ці дані можна архівувати. Таким чином можна робити інтелектуальний аналіз зображень, ідентифікуючи об'єкти у зазначений час та у заданому місці.

У загальному вигляді вебсервіс аналізу трафіку з використанням комп'ютерного зору складається з декількох апаратних і програмних модулів. До апаратних пристроїв відтворення відеоінформації відноситься будь-яке обладнання з екраном – смартфони, монітори персональних комп'ютерів і т.д. Сервери, хмарні сховища і сервіси використовуються для накопичення отриманої інформації та проведення її аналізу. Основні структурні компоненти системи (рис. 2.1): відеокамера, що захоплює зображення, сервер для зберігання інформації записаної з камер, на якому також розгорнуто вебсервер та БД зображень, пристрої для відображення інформації (наприклад, ПК, телефон), що забезпечують доступ до системи через браузер з доступом до вебсервера.

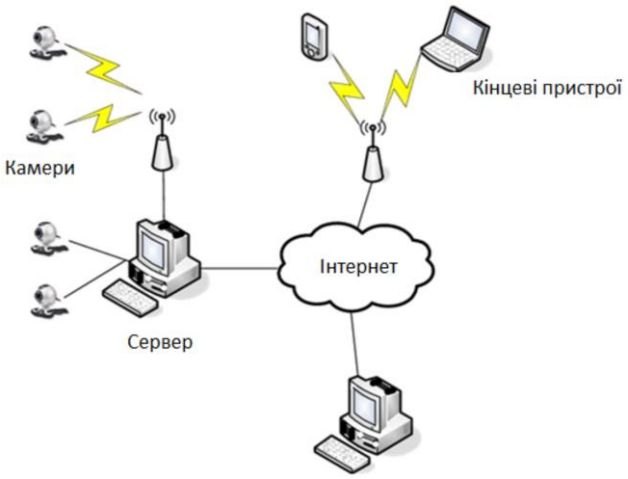

Рисунок 2.1 – Загальна схема передачі даних розподіленої мережі вебкамер

Дана архітектура дозволяє отримувати доступ до відеоконтенту у реальному часі та дає можливість розпізнавати контури об'єктів. На вхід модуля розпізнавання об'єктів з вебкамери надходить зображення на якому програмний детектор знаходить автомобілі, що рухаються. Далі на дорозі визначається положення машин, тобто область, де вони перебувають. ПЗ обробляє знайдені об'єкти та надсилає інформацію у СКБД, яка забезпечує їх зберігання, потім інформація з СКБД обробляється для аналізу трафіку на дорогах. Тобто основними завданнями системи відеоспостереження є: фіксація об'єкту, часу і дати появи машини в куті огляду відеокамери, запис стоп-кадру або фрагменту відео з авто, відправка зображення в СКБД [22].

Тим не менш, відеодані спостереження за дорогами мають свої унікальні характеристики. Запис відео є послідовним. Відеодані – це потокові медіадані з великим розміром файлу, тому швидкість запису відео висока. Зміст даних регулярно змінюється відповідно до потоку автомобільного руху. Операція запиту відносно мала, але робоче навантаження досить велике. Система повинна записувати та зіставляти великі обсяги збережених даних. Відповідно до характеристик відеоданих спостереження пропонується система зберігання відео SAN (англ. Storage Area Network) на основі IP-SAN. Кожен відеокадр зберігається в області даних фіксованого розміру [23].

Розвиток системи відеоспостереження пройшов шлях від аналогової передачі сигналу до цифрової. NVR і DVR використовуються з різними типами камер спостереження, що показує найбільш істотну різницю між цими двома типами відеореєстраторів. Якщо порівнювати DVR та NVR, то основновною відмінністю є те, що NVR використовує програмне забезпечення. NVR має функцію отримання даних IPC (IP-камери), відеокодека, сховища, дисплея в режимі реального часу тощо. Спочатку відео кодується та обробляється мережевою відеокамерою. Далі до NVR відео надсилається для зберігання чи віддаленого перегляду [24]. Він також може пересилати збережені відео в інші системи зберігання через мережу (рис. 2.2)

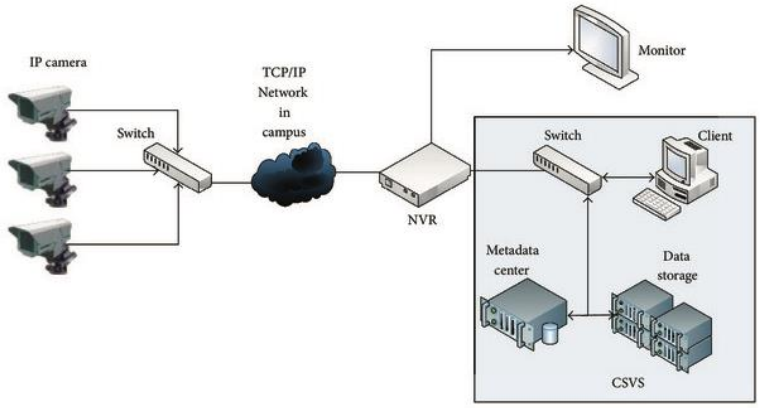

Рисунок 2.2 – Розгортання системи відеоспостереження

Система відеоспостереження має бути забезпечена великим сховищем, що складається з трьох частин: клієнта, кластера метаданих та кластера зберігання даних. Кластер метаданих включає центр індексації ключових кадрів відео. Структура системи показано на рис. 2.3.

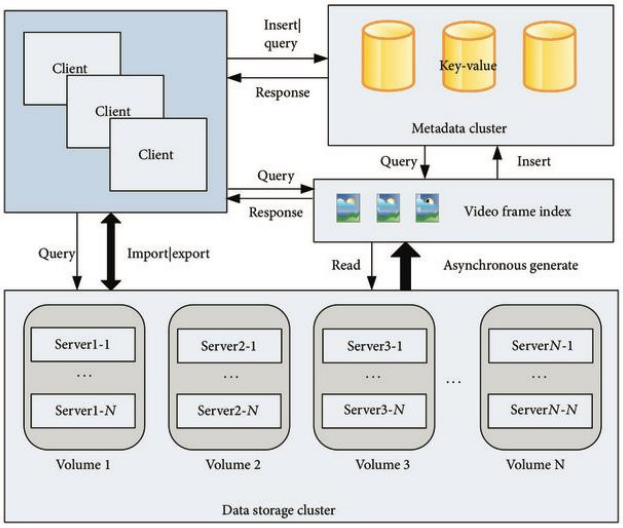

Рисунок 2.3 – Структура системи зберігання даних

Клієнт системи відповідає за ініціювання завдань та збирання операційних результатів. Клієнт надає внутрішні та зовнішні API, а також функції імпорту та експорту відеоданих на основі запитів метаданих та відеоконтенту та інших операцій. Усі ці операції запускаються клієнтом.

Кластер метаданих відповідає за зберігання метаданих, що описує відео відправлене клієнтом. Дані зображення, згенеровані функцією вилучення ключового кадру, пов’язані з метаданими. Він також надає клієнту функцію пошукового запиту. Кластер досягає балансу навантаження, щоб змусити всі сервери в кластері виконувати завдання разом. Кластер зберігання даних відповідає за читання та запис відеоданих. Реплікація даних зберігається у вигляді томів на різних вузлах сервера для забезпечення резервного копіювання та автоматичного відновлення у разі ненормального зберігання даних.

Весь кластер складається з декількох томів зберігання, тому зберігання складається з декількох вузлів служби зберігання. Коли сервісний вузол виходить з ладу, інші, як і раніше, гарантують обслуговування. Лінійне розширення простору можна досягти шляхом збільшення обсягів зберігання.

Відеоспостереження відрізняється від інших видів відеозйомки. Кут зйомки відео стабілізовано. Таким чином, фон відео не дуже змінюється. Відповідно, різниця між кадрами відео порівнюється за допомогою розрахунку відстані гістограми [25]. Оскільки на дорозі існує певний графік, зміст відео також є регулярним. Вранці та ввечері під час робочого перевезення відеоконтент є найрізноманітнішим, але в нічний час кількість навантаження зменшується, оскільки трафік знижується. Якщо ми зіставляємо контент годин пікової завантаженості доріг, де в кожному кадрі відео з рухом машин, то системі необхідно робити велику кількість обчислень. Щоб вирішити цю проблему в поєднанні з вищезазначеними характеристиками, а також витягти ключовий кадр відео [26] та побудувати індекс вмісту відео, необхідно розрахувати різницю у гістограмі.

В операціях вилучення ключових кадрів для розрахунку формули відстані гістограми існує чотири методи: CORREL, Chi-Square, Bhattacharyya та INTERSECT. Наша стратегія вилучення ключових кадрів полягає в тому, щоб використовувати якнайменше кадрів для підсумовування всього відеоконтенту. Для підвищення обчислювальної ефективності використовуємо метод INTERSECT, який є найкращим методом для розрахунку відстані між гістограмами. Метод INTERSECT має таку формулу:

$$d_{\text{intersect}}\left(H_1, H_2\right) = \sum_i \min\left(H_1\left(i\right), H_2\left(i\right)\right). \tag{2.1}$$

де H_1 та H_2 – це дві гістограми.

Поріг – це змінна з плаваючою комою від 0 до 1. Чим ближчий поріг до 1, тим більше ключових кадрів записується. Найменування ключових кадрів – це формат довгого цілого числа + розширення.

Довге ціле число – це порядковий номер кадрів у відео. Порядковий номер кадру можна використовувати для визначення часової точки, де кадр з’являється у відео.

$$\text{Time} = \frac{\text{num}}{\text{frame\_rate}} + 1. \tag{2.2}$$

Наприклад, якщо тривалість відео становить 1000 секунд та 25 кадрів на секунду (frame_rate), відео складається з 25000 кадрів. Якщо ключовий кадр, який потрібно запитувати, є 1024 кадром (довге ціле число), кадр може бути зіставлений за 41 секунду відео відповідно до формули.

Запис даних. Запис відео ініціюється клієнтом. Клієнт отримує відео з NVR, що генерує метадані на основі відео і починає операцію запису. Процедура запису виглядає так. З NVR отримується відео. Далі файл відео зберігається, записується час початку та час закінчення відео, обчислюється тривалість відео, аналізується розмір відеофайлу, генерується шлях до відеофайлу, вилучаються ключові кадри та генерується шлях

зберігання ключових кадрів. Потім дані записуються в базу метаданих, а відеозаписи зберігаються відповідно до черги та продовжується обробка наступних відеозаписів. Далі запускається функція асинхронного вилучення ключового кадру та перевіряється чи дорівнює він 0. Вилучаються ключові кадри та записуються у шляху зберігання, присвоюється 1, операцію запису завершено.

Отримання відеоданих. Отримання відео має працювати лише з базою метаданих. Отримання відеоконтенту ґрунтується на індексі ключових кадрів. *Процес отримання.* Відповідно до умов пошуку, таких як час початку, час закінчення, тривалість, точка зйомки або комбінація цих умов, може бути ініційовано запит пошуку до бази метаданих. База метаданих виводить всі записи даних, які відповідають умовам. Далі генерується шлях зберігання відео з усіх записів та вилучається ключовий кадр. Реалізація алгоритму отримання ключових кадрів зображено на рис. 2.4.

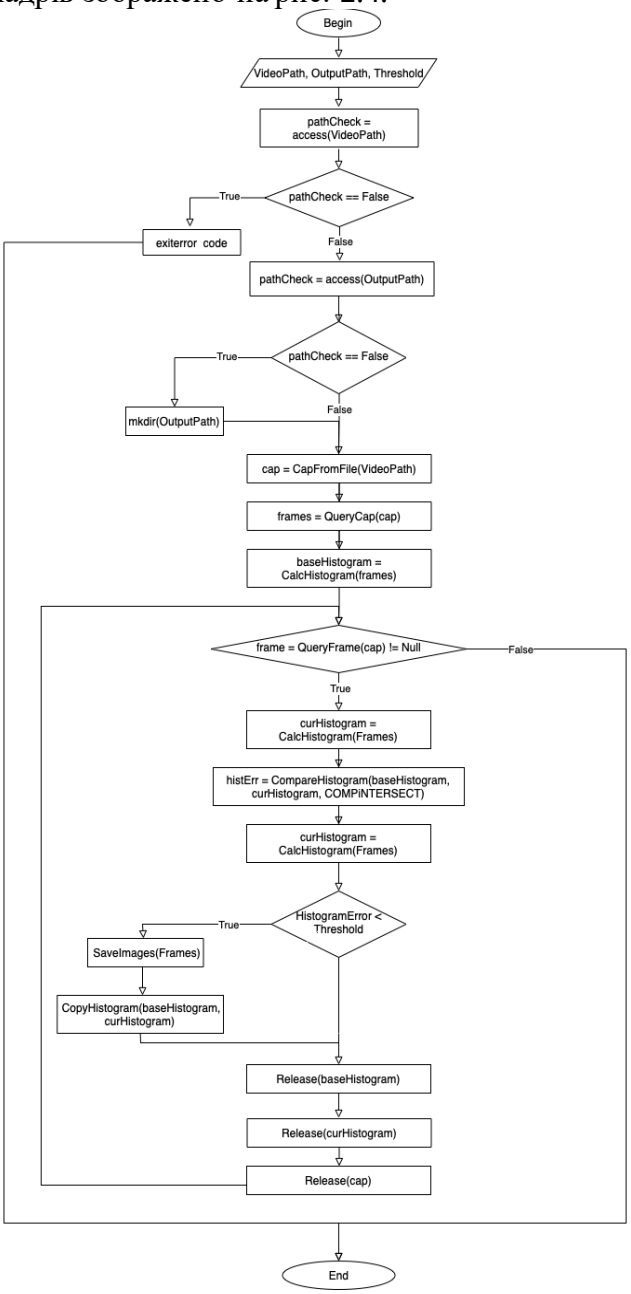

Рисунок 2.4 – Алгоритм отримання ключових кадрів зображення

Отже, було розглянуто алгоритм накопичення даних у системі моніторингу трафіку для навігації транспортних засобів. Досягнувши масштабування простору зберігання та високої доступності даних, технологія вилучення ключових кадрів застосовується до функції індексу зберігання системи моніторингу трафіку. Таким чином система з індексом ключових кадрів може підвищити ефективність запиту на основі відеоконтенту. Алгоритм вилучення ключових кадрів зображення було покращено на основі розрахунку відстані гістограми та знайдемо більш точну техніку вилучення ключових кадрів. Також було оптимізовано алгоритм процесу зіставлення запиту ключового кадру для підвищення ефективності пошуку на основі відеоконтенту.

2.2 Алгоритм роботи вебсервісу обробки відео з вебкамер

Для реалізації алгоритму розпізнавання об’єктів було використано бібліотеку комп’ютерного зору OpenCV. OpenCV (англ. Open Computer Vision) – це бібліотека комп’ютерного зору з відкритим вихідним кодом, що надає ряд числових алгоритмів і типів даних, що обробляють зображення, використовуючи алгоритми комп’ютерного зору [27]. Алгоритми застосовуються для ідентифікації об’єктів, відстеження рухомих об’єктів, відстеження рухів камери, класифікації дій людей у відео, з’єднання зображень для отримання високої роздільної здатності зображень всієї сцени, пошук подібних зображень у базах даних зображень, виявлення та розпізнавання облич, отримання 3D-хмарних точок зі стереокамер, визначення пейзажу, вилучення 3D-моделей об’єктів тощо [28].

У ряді випадків виділення об’єктів на зображеннях використовується як основний етап обробки зображення для вирішення складних завдань, а саме для розпізнавання машин. У трекінгових системах також можна використовувати даний підхід, якщо колір об’єкта відрізняється від фону. Рішення проблеми ділиться на кілька етапів. Спочатку відбувається виділення пікселів, що відносяться до заданого об’єкту. Далі починається виділення за контурами знайдених об’єктів та знаходження контуру предмета. Накінець будується прямокутник, що містить усі точки контуру об’єкта.

Колірна модель HSV використовується для виділення та пошуку об’єктів по кольору. Дана модель описує колірний простір на основі трьох колірних ознак: колірний тон (Hue), насиченість (Saturation) і яскравість (Brightness, Value). Даний колірний простір є нелінійним. Ідеальним інструментом для побудови алгоритмів обробки зображень вважається простір HSV (HSI), так як він базується на природному і інтуїтивно зрозумілому описі кольору (рис. 2.5).

```
import cv2
import numpy as np
import os
##########################################################################
#############################
def main():
 capWebcam = cv2.VideoCapture(0) #початок захвату відеозображення об'єкту та зв’язок з вебкамерою
 print «defaultresolution = « + str(capWebcam.get(cv2.CAP_PROP_FRAME_WIDTH)) + «x» + str(capWebcam.get(cv2.CAP_PROP_FRAME_HEIGHT)) #відображення оригінального розширення камери
 capWebcam.set(cv2.CAP_PROP_FRAME_WIDTH, 320.0) #трансформація розширення камери
 capWebcam.set(cv2.CAP_PROP_FRAME_HEIGHT, 240.0) #зменшення значення розширення камери для кращої швидкодії
 if capWebcam.isOpened() == False: #перевірка підключення камери
 print «error: cap Webcam not accessed successfully\n\n»
 os.system(«pause»)
 return
 # endif
 while cv2.waitKey(1) != 27 andcapWebcam.isOpened(): #програма працює доти, доки не blnFrameReadSuccessfully, imgOriginal = capWebcam.read() #нажати кнопку ESC або не вимкнути камеру
 if not blnFrameReadSuccessfully or imgOriginal is None: #перевірка на наявність кадру з камери
 print «error: frame not read from webcam\n»
 os.system(«pause»)
 break
 # endif
```

Рисунок 2.5 – Фрагмент коду програми для побудови алгоритмів обробки зображень

З простору RGB можна отримати відтінок для моделі HSV. Програмний код для реалізації алгоритму розпізнавання об’єктів за кольором наведено нижче (рис. 2.6):

```
        imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)
        #встановлення діапазону заданого кольору на зображені
        imgThreshLow = cv2.inRange(imgHSV, np.array([0, 170, 120]), np.array([20, 240, 255]))
imgThreshHigh = cv2.inRange(imgHSV, np.array([20, 70, 170]), np.array([40, 170, 255]))

        imgThresh = cv2.add(imgThreshLow, imgThreshHigh)
#створення маски зображення заданого кольору
        imgThresh = cv2.cvtColor(imgThresh, cv2.COLOR_GRAY2RGB)
        #згладжування об'єкта методом Гаусса
        imgThresh = cv2.GaussianBlur(imgThresh, (5, 5), 2)
        #морфологічна операція по розширенню об`єкту для заповнення випадкового
простору в об'єкті
        imgThresh = cv2.dilate(imgThresh, np.ones((5,5),np.uint8))
        #трансформація картинки в сірі відтінки
        imgThresh = cv2.cvtColor(imgThresh, cv2.COLOR_BGR2GRAY)
        #бінарізація зображення
        _, imgThresh = cv2.threshold(imgThresh, 130, 255, cv2.THRESH_BINARY)
        #морфологічні операції для покращення цільності об'єкта на зображені
        st1 = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 21), (10, 10))
        st2 = cv2.getStructuringElement(cv2.MORPH_RECT, (11, 11), (5, 5))
        imgThresh = cv2.morphologyEx(imgThresh, cv2.MORPH_CLOSE, st1)
        imgThresh = cv2.morphologyEx(imgThresh, cv2.MORPH_OPEN, st2)
        #знаходження контурів об'єкта
        img, contours,hierarchy = cv2.findContours(imgThresh, 2, 4)
        #формування прямокутника (зеленого кольору) навколо об'єкта
        cnt = contours[0]
        a,b,w,h = cv2.boundingRect(cnt)
        cv2.rectangle(imgOriginal,(a,b),(a+w,b+h),(0,255,0),2)
        #знаходження центру об'єкта
        x = (a+w)/2
        y = (b+h)/2
        #вивід картинки на екран монітору
        cv2.namedWindow(«imgOriginal», cv2.WINDOW_AUTOSIZE)
        cv2.namedWindow(«imgThresh», cv2.WINDOW_AUTOSIZE)
        cv2.imshow(«imgOriginal», imgOriginal)
        cv2.imshow(«imgThresh», imgThresh)
        # endwhile
        cv2.destroyAllWindows() #очищення пам'яті від картинки
        return #знову на початок циклу
##################################################################
##########################
        if__name__ == «__main__»:
        main()
```

Рисунок 2.6 – Фрагмент коду програми для реалізації алгоритму розпізнавання об'єктів за кольором

Існує кілька способів виявлення, відстеження і підрахунку транспортних засобів. Один із способів виявити транспортні засоби – використати каскади Хаара. Каскад Хаара – це алгоритм для виявлення об'єктів у відео та зображеннях, що застосовується для ідентифікації об'єктів і заснований на концепції ознак, запропонованих Полом Віолою та Майклом Джонсом у 2001 році. Підхід базується на машинному навчанні. Каскадна функція навчається на наборі безлічі позитивних і негативних шаблонів, що використовуються, щоб виявляти об'єкти на інших зображеннях [29]. Алгоритм складається з чотирьох етапів: вибір функції Хаара, створення інтегральних зображень, навчання Adaboost, каскадні класифікатори. Даний алгоритм здатний розпізнавати частини тіла та обличчя на зображенні, але також він може навчитися ідентифікації майже будь-який об'єкту.

Алгоритм роботи функції. Зібираємо ознаки Хаара [30]. Функція Хаара у вікні виявлення враховує сусідні прямокутні області. Далі інтенсивність пікселів підсумовується у кожній області та обчислюється різниця між даними сумами (рис. 2.7).

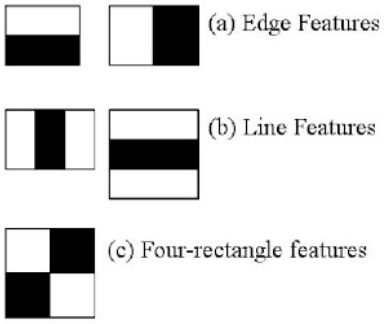

Рисунок 2.7 – Ознаки Хаара

Для швидкого обчислення використовуються інтегральні зображення. Але більшість із них не мають значення. Найкраща функція підбирається за допомогою концепції Adaboost. Вона навчає класифікатори та вибирає найкращі функції для використання. Алгоритм за допомогою лінійної комбінації зважених простих «слабких» класифікаторів створює «сильний» класифікатор. Даний процес виглядає наступним чином [31].

Характеристики Хаара розраховуються для кожної частини зображення у процесі фази виявлення вікна відображення цільового розміру, яке рухається над вхідним зображенням. Велика кількість характеристик Хаара потрібна для опису об’єкта з достатньою точністю, оскільки кожна з характеристик Хаара [32] є тільки «слабким класифікатором» (трохи вища якість виявлення, ніж випадкові припущення). Таким чином, дані характеристики для формування сильного класифікатора організовуються в каскадні класифікатори.

Каскадний класифікатор складається з набору етапів, у якому кожен з етапів є ансамблем слабких учнів. Пнями називаються прості класифікатори, які слабко навчаються. За допомогою техніки прискорення тренується кожний етап. Шляхом прийняття середньозваженого рішення, прийнятого слабкими учнями, підготувати дуже точний класифікатор дає можливість стимулювання [33].

Поточним розташуванням ковзного вікна визначено область, яка позначається кожним етапом класифікатора або як позитивну, або як негативну. Якщо визначається як позитивна, то означає що об’єкти знайдено, негативний, навпаки – об’єкти не знайдено. Якщо мітка є негативною, класифікація цієї області завершена [34], і детектор ковзає до наступного місця. Якщо мітка позитивна, класифікатор переміщує область на наступний етап. Коли останній етап класифікує область як позитивну, то детектор повідомляє про об’єкт, знайдений у поточному місці вікна.

Етапи призначені для відкидання найбільш негативних шаблонів. Припускається, що переважна більшість вікон не містить потрібного об’єкта. Низький рівень хибних негативних результатів на кожному етапі каскаду забезпечує коректну роботу алгоритму. Класифікація припиняється, якщо на даному етапі неправильно позначено об’єкт як негативний. У такому випадку виправити помилку більше неможливо. Однак кожна стадія може мати високий рівень хибнопозитивних результатів [35]. Навіть якщо детектор неправильно позначає об’єкт як позитивний, то можна виправити помилку, виконавши наступні дії. Додавання додаткових ступенів зменшує загальну частоту помилкових позитивних результатів, але також зменшує загальну частоту справжніх позитивних результатів.

Набір зразків позитивних та негативних зображень є обов’язковою умовою для навчання каскадного класифікатора. Позначати об’єкти можна використовувати обмежуючі полям – Image Labeler [36]. Для використання позитивних зразків виводить таблицю Image Labeler. Тип функції та її параметри, а також кількість етапів встановлюються для досягнення прийнятної точності детектора (рис. 2.8).

Рисунок 2.8 – Схема навчання каскадів

Функція має автоматично генерувати негативні зразки із наданих наборів негативних зображень. Алгоритм навчання каскадів Хаара для виявлення машини на зображенні з відеокамер зображено на рис. 2.9.

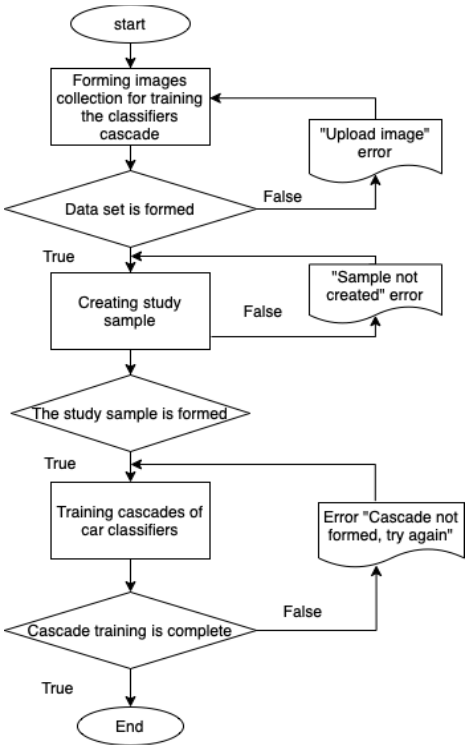

Рисунок 2.9 – Алгоритм навчання каскаду Хаара

Таким чином, використання алгоритмів для розпізнавання об’єктів з відеокамер, дозволяє покращити навігацію дорогами. Водій зможе планувати маршрут, розрахувати скільки часу потрібно переміщення з точки А в точку В відповідно до трафіку на дорогах. Використання данного алгоритму та програмного забезпечення допоможе вийти системам навігації на новий рівень.

2.3 Алгоритм роботи застосунку для навігації водіїв транспортних засобів

Автомобільна навігація математично базується на пошуці найкоротшого шляху згідно з теорією графів, за допомогою яких можна визначити між двома точками у великій

мережі найкращий шлях, що відповідає заданим критеріям (найкоротший, найшвидший тощо).

Для пошуку найкоротшого шляху по заданому маршруту на основі аналізу трафіку було обрано алгоритм Дейкстри. Алгоритм Дейкстри призначений для пошуку найкоротших шляхів між вузлами в графі. Він був розроблений голландським вченим Едсгером Вібе Дейкстрою в 1956 році, коли він обмірковував найкоротший шлях від Роттердама до Гронінгена [36]. На рис. 2.10 зображено принцип роботи алгоритму.

Рисунок 2.10 – Принцип роботи алгоритму Дейкстри

Алгоритм Дейкстри змінювався протягом багатьох років, і існують різні версії та варіації. Спочатку він використовувався для обчислення найкоротшого шляху між двома вузлами. Завдяки цьому його було адаптовано для обчислення найкоротшого шляху між початковим вузлом і кожним іншим вузлом на графі. Таким чином, його можна використовувати для створення дерева найкоротшого шляху, яке складається з найкоротшого шляху між двома вузлами, а також усіма іншими вузлами. Потім можна отримати найкоротший шлях між двома вузлами, обчисливши все дерево, що є недоліком алгоритму Дейкстри [37].

Нехай вузол, з якого ми починаємо, називатиметься початковим вузлом. Нехай відстань вузла Y – це відстань від початкового вузла до Y. Алгоритм Дейкстри спочатку почне з нескінченних відстаней і спробує покращити їх крок за кроком. Розглянемо детальніше алгоритм.

Блок-схему алгоритму показано нижче (рис. 2.11).

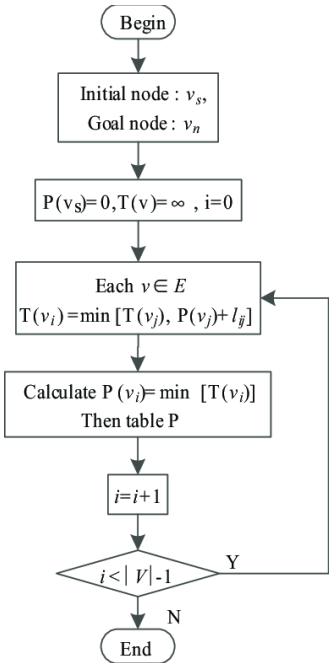

Рисунок 2.11 – Блок-схема алгоритму Дейкстри

Алгоритм Дейкстри працює на неорієнтованих зв'язних зважених графах. Спочатку створюємо набір відвіданих вершин, щоб відстежувати всі вершини, яким було призначено правильний найкоротший шлях. Також потрібно встановити «вагу» всіх вершин у графі (довжини поточного найкоротшого шляху, який веде до нього). Як вже зазначалось, на початку всі вузли будуть встановлені на «нескінченність», щоб переконатися, що всі інші вузли, з якими ми можемо порівняти їх, будуть меншими за початкові. Єдиним винятком є значення першої початкової вершини – ця вершина матиме значення 0, оскільки вона не має шляху до себе – позначена як node s. Потім ми повторюються два основні кроки, поки граф не буде пройдено: вибрати вершину з найкоротшим поточним значенням, відвідати її та додати до набору відвіданих вершин та оновити значення усіх суміжних вершин, які ще не відвідані [38].

Для кожного краю між n і m, де m є невідвіданим – якщо cheapestPath (s, n)+cheapestPath(n, m)<cheapestPath(s,m), оновити найменший шлях між s і m до рівня cheapestPath(s,n)+cheapestPath(n,m). Розглянемо покрокове виконання алгоритму на прикладі графа (рис. 2.12).

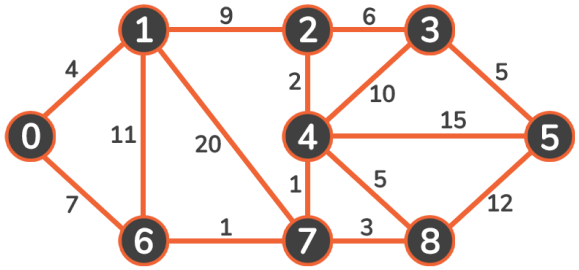

Рисунок 2.12 – Неорієнтований, зважений зв'язний граф

Припустимо, вершина 0 є відправною точкою, тоді ваговий коефіцієнт – це довжина найкоротшого шляху із вершини 0. Встановлюємо початкові значення вершин у цьому графі до нескінченності, за винятком початкової вершини (таблиця 2.1):

Таблиця 2.1 – Початкові значення вершин у графі

Вершина	0	1	2	3	4	5	6	7	8
Ваговий коефіцієнт	0	inf	inf	inf	inf	inf	inf	inf	inf

Обираємо вершину з мінімальною вагою, тобто вершину 0. Позначаємо її як відвідану і додаємо до набору відвіданих вершин. Початковий вузол завжди матиме найнижчу вагу, тому він завжди додаватиметься першим (рис. 2.13):

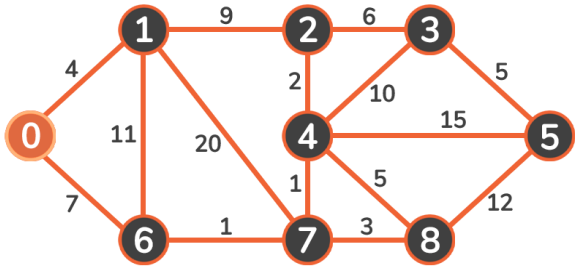

Рисунок 2.13 – Найкоротші відстані від 1-ї вершини 0 до інших

Потім необхідно оновити вагу суміжних вершин (1 і 6). Оскільки 0 + 4 < infinity і 0 + 7 < infinity, оновлюємо ваговий коефіцієнт цих вершин (таблиця 2.2):

Таблиця 2.2 – Початкові значення вершин у графі

Вершина	0	1	2	3	4	5	6	7	8
Ваговий коефіцієнт	0	4	inf	inf	inf	inf	7	inf	inf

Тепер відвідуємо наступну вершину з найменшим ваговим коефіцієнтом. Вага 4 менша за 7 , тому переходимо до вершини 1. Після обходу необхідно позначити вершини як відвідані, а потім оновити сусідні вершини: 2, 6 і 7: так як 4 + 9 < infinity, новий ваговий коефіцієнт вершини 2 становитиме 13; так як 4 + 11 > 7, ваговий коефіцієнт вершини 6 залишиться 7; так як 4 + 20 < infinity, ваговий коефіцієнт вершини 7 становитиме 24.

Повторюємо крок алгоритму, обираємо вершину 6. Позначаємо її як відвідану і оновлюємо ваговий коефіцієнт суміжних вершин (вузлів) Повторюємо крок алгоритму для вершини 7. Повертаємось до вершини 4. Повертаємось до вершини 2. Розглянемо вершину 3. Оскільки 11 + 6 < 19, ваговий коефіцієнт вершини 3 оновлюється. Потім переходимо до вершини 8. Зрештою було оновлено вершини петлеподібної структури, тож тепер необхідно пройти її спочатку до вершини 3 (рис. 2.14):

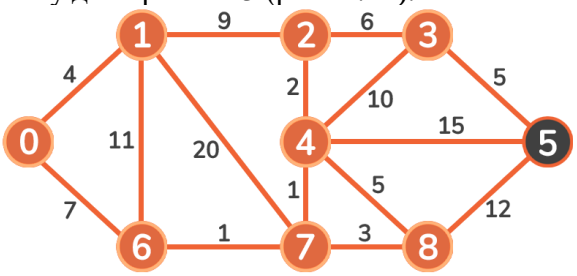

Рисунок 2.14 – Проходження всіх вершин з початку третьої вершини

І, нарешті, до вершини 5 (рис. 2.15):

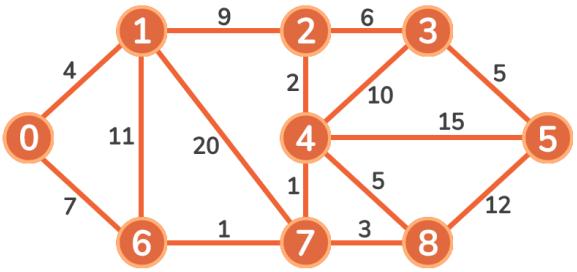

Рисунок 2.15 – Проходження до п’ятої вершини

Більше немає невідвіданих вершин, які можуть потребувати оновлення [39]. Було знайдено найкоротші шляхи від вузла 0 до будь-якого іншого вузла на графіку (таблиця 2.3):

Таблиця 2.3 – Початкові значення вершин у графі

Вершина	0	1	2	3	4	5	6	7	8
Ваговий коефіцієнт	0	4	11	17	9	24	7	8	11

Таким чином скорочується таблиця та зберігається відстань між вузлом X і вузлом Y.

Пов’язані проблеми та алгоритми. Функціональність алгоритму Дейкстри може бути розширена модифікаціями. Наприклад, можна додати рішення, які не є математично оптимальними. Щоб отримати ранжований список неоптимальних рішень спочатку розраховується оптимальне рішення. Одне ребро, що у оптимальному рішенні, видаляється з графіка, і обчислюється оптимальне рішення цього нового графа. Кожне ребро вихідного рішення по черзі проходиться і розраховується найкоротший шлях. Потім вторинні рішення ранжуються та подаються після першого оптимального рішення. Алгоритм Дейкстри є принципом роботи протоколів маршрутизації стану каналу, найбільш поширеними з яких є OSPF і IS-IS.

На відміну від алгоритму Дейкстри алгоритм Беллмана-Форда може використовуватися на графах з негативною вагою ребер, якщо граф містить негативний цикл, досяжний з вихідної вершини. Наявність таких циклів означає, що немає найкоротшого шляху, оскільки загальна вага стає нижчою кожного разу, коли цикл проходить. Можна адаптувати алгоритм Дейкстри для обробки ребер негативної ваги, об’єднавши його з алгоритмом Беллмана-Форда.

Перспективи динамічного програмування. З точки зору динамічного програмування алгоритм Дейкстри є послідовною апроксимаційною схемою, яка вирішує функціональне рівняння динамічного програмування для задачі про найкоротший шлях методом Reaching.

Таким чином, розроблення програмного забезпечення для навігації водіїв було здійснено на основі алгоритму Дейкстри мовою програмування Python. Область застосування розробленого програмного забезпечення включає такі сфери як створення систем навігації і стеження, картографічних сервісів. Даний алгоритм дозволяє оптимізувати багатокритеріальний пошук оптимального шляху на основі трафіку з урахуванням вимог заданого маршруту.

Висновки до розділу 2

У даному розділі було описано алгоритми накопичення даних розподіленої мережі вебкамер, обробки відео з вебкамер та роботи навігації водіїв транспортних засобів. Також було оптимізовано алгоритм процесу зіставлення запиту ключового кадру. Моніторинг завантаженості доріг стане легшим і простішим. Водій зможе планувати маршрут, розрахувати скільки часу потрібно переміщення з точки А в точку В. Розроблення програмного забезпечення для навігації водіїв було здійснено на основі алгоритму Дейкстри мовою програмування Python, оптимізувати багатокритеріальний пошук оптимального шляху на основі трафіку з урахуванням вимог заданого маршруту. Використання данного алгоритму та програмного забезпечення допоможе вийти системам навігації на новий рівень.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА ПРОЄКТУВАННЯ ВЕБСЕРВІСУ НАВІГАЦІЇ

3.1 Вибір технологій розробки

Правильно обрані технології – це найважливіша частина будь-якого проєкту. Обраний стек технологій для програмного комплексу було обрано з розрахунком на стабільність, гнучкість та продуктивність програмного середовища. У табл. 3.1 наведено опис засобів реалізації.

Таблиця 3.1 — Опис засобів реалізації

Позиція	Технологія
Мова програмування	Python
Технологія	Flask
База даних	MySQL
ORM	SQLAlchemy

Для реалізації вебсервісу були обрані такі технології, як *Python, Flask, JS (HTML, CSS), OpenCV [40], MySQL*. Розглянемо кожну з них більш детально.

Python є першою найпопулярнішою мовою програмування у світі, мова посідає перше місце в першій десятці найкращих індексів мов програмування TIOBE та PYPL PopularitY. У всьому світі Python є найпопулярнішою мовою, поширеність використання Python виросла найбільше за останні 5 років (7,8%), а мова Java втратила найбільше (-5,2%).

Для створення вебсервісу було обрано Python через простий синтаксис та наявність фреймворків, необхідних для виконання поставленої задачі. Python – об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Дана мова програмування підходить для швидкої розробки додатків, оскільки вона має високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та прив’язкою. Також її використовують як сценарій або мову для підключення існуючих компонентів. Простий, легкий у освоєнні синтаксис Python підкреслює зручність читання та полегшує обслуговування програм, написаних цією мовою. Модулі та пакети, які підтримує Python, дозволяють повторне використання коду [41-45]. Python є мовою загального призначення, тобто її можна використовувати в різноманітних програмах, включаючи науку про дані, програмне забезпечення та веброзробку, автоматизацію тощо. Вихідний код Python тепер доступний за Стандартною громадською ліцензією GNU (GPL) [46].

Переваги Python: має просту структуру та чітко визначений синтаксис, легко читається, простий в обслуговуванні, має велику стандартну бібліотеку, підтримує інтерактивний режим, підтримує графічні програми, працює на різних апаратних платформах на яких має однаковий інтерфейс та надає їх до всіх основних комерційних баз даних, до інтерпретатора Python може додавати модулі низького рівня, бібліотеки та системи Windows, Windows MFC, Macintosh і система X Window Unix, масштабований [47].

Недоліками є продуктивність, низька швидкодія, глобальне блокування інтерпретатора (GIL), вбудовані класи неможливо модифікувати, статична типізація відсутня.

Flask – це мікрофреймворк, що пропонує всі основні функції вебзастосунку. Легко почати та розширити за допомогою великої бібліотеки та розширень Python. Можливості самого фреймворку обмежені, але він дозволяє розробникам змінювати та адаптувати його до своїх конкретних потреб на власний розсуд. З Flask диверсифікація структури програмного проєкту за допомогою мікрофреймворків походить без особливих зусиль.

Гнучкість є основною особливістю цього фреймворку з відкритим кодом. Flask ідеально підходить для експериментів з архітектурою, бібліотеками та новими технологіями. Практично кожен проєкт Flask має свій власний унікальний «внутрішній технологічний стек» бібліотек, фреймворків тощо. Таким чином Flask значною мірою полегшує експерименти [48].

Переваги Flask: гнучкість, сумісність із новими технологіями, масштабованість для простих вебзастосунків, можливість експериментувати, прості налаштування та використання, висока продуктивність використанні для простих випадків, менший розмір кодової бази.

Недоліки Flask: технологічний стек є складним, можливі ризики безпеки, більш повільний розвиток MVP, технічне обслуговування складних систем має високі витрати, ускладнене обслуговування великих проєктів.

JavaScript – це динамічна мова програмування. Вона легка і найчастіше використовується як частина вебсторінок, реалізації яких дозволяють скрипту клієнта взаємодіяти з користувачем і створювати динамічні сторінки. Це інтерпретована мова програмування з об'єктно-орієнтованими можливостями. JavaScript спочатку був відомий як LiveScript, але Netscape змінив свою назву на JavaScript, можливо через подібність до створюваного Java. Ядро мови загального призначення було вбудоване в Netscape, Internet Explorer та інші браузери.

Переваги використання JavaScript: мінімальна взаємодія з сервером, зворотний зв'язок з відвідувачами, підвищена інтерактивність, різноманітність інтерфейсів.

JavaScript не може розглядатися як повноцінна мова програмування. Їй не вистачає таких важливих особливостей: клієнтський JavaScript не дозволяє читати чи записувати файли, не може бути використаний для мережевих програм, не має багатопотокових або багатопроцесорних можливостей. JavaScript є однією з основних технологій Всесвітньої павутини поряд з *HTML* та *CSS*.

HTML (мова гіпертекстової розмітки) – це текстовий підхід до опису структури контенту, що міститься в HTML-файлі. Ця розмітка повідомляє веббраузеру, як відображати текст, зображення та інші форми мультимедіа на вебсторінці. HTML є офіційною рекомендацією Консорціуму Всесвітньої павутини (W3C) і, як правило, дотримується всіх основних веб-браузерів, включаючи як настільні, так і мобільні веббраузери. HTML5 – це остання версія специфікації.

CSS (каскадні таблиці стилів) – це мова таблиць стилів, що застосовується для опису представлення документа, написаного мовою розмітки, такою як HTML або XML [51].

Для реалізації алгоритму розпізнавання об'єктів було використано бібліотеку комп'ютерного зору OpenCV. *OpenCV (англ. Open Computer Vision)* – це бібліотека комп'ютерного зору з відкритим вихідним кодом, що надає ряд числових алгоритмів і типів даних для обробки зображень за допомогою алгоритмів комп'ютерного зору. Програма має інтерфейс шаблону, який легко працює з контейнерами STL. Бібліотека містить багато оптимізованих алгоритмів, які включають підмножину як класичних, так і сучасних алгоритмів комп'ютерного зору та машинного навчання [27]. Алгоритми застосовуються для ідентифікації об'єктів, відстеження рухомих об'єктів, відстеження рухів камери, класифікації дій людей у відео, з'єднання зображень для отримання високої роздільної здатності зображень всієї сцени, пошук подібних зображень у базах даних зображень, виявлення та розпізнавання облич, отримання 3D-хмарних точок зі стереокамер, усунення ефекту червоних очей у зображеннях зі спалахом, відстеження руху очей, визначення пейзажу, вилучення 3D-моделей об'єктів тощо [28].

OpenCV було створено, щоб прискорити використання машинного сприйняття в комерційних продуктах та створити загальну інфраструктуру для програм комп'ютерного зору. Бібліотека має інтерфейси Java, C++ та Python, а також підтримує Android, Windows, Mac OS та Linux. У ряді випадків виділення об'єктів на зображеннях використовується як

основний етап обробки зображення для вирішення складних завдань, а саме для розпізнавання машин.

MySQL – це система управління реляційними базами даних (RDBMS), яка є вільною, з відкритим вихідним кодом та використовує різні пропрієтарні ліцензії, включаючи Стандартну громадську ліцензію GNU (GPL). Як СУБД MySQL використовує SQL для управління даними всередині бази даних. Він організує корельовані дані в одну або кілька таблиць даних, і ця кореляція допомагає структурувати дані. Це дозволяє програмістам використовувати SQL для створення, зміни та вилучення даних з реляційної бази даних. Нормалізуючи дані в рядках та стовпцях таблиць, MySQL перетворюється на масштабовану, але гнучку систему зберігання даних із зручним інтерфейсом, який може керувати великою кількістю даних [49].

SQLAlchemy – це набір інструментів SQL з відкритим вихідним кодом та об'єктно-реляційний картограф (ORM) для мови програмування Python, випущений за ліцензією MIT. Система представлення схем, яка може видавати як інструкції DDL, так і самоаналізувати існуючі схеми, і система типів, яка дозволяє будь-яке зіставлення типів Python з типами баз даних, завершує систему [50].

Таким чином, розробка програмних проєктів має свої етапи, зокрема початковими є обрання основного функціоналу та визначення функціональних вимог до розробленого вебсервісу.

3.2 Проєктування взаємодії користувачів вебзастосунку

Вебзастосунок є частиною системи, яка надає функціонал у виконані вебсайту. Вебзастосунок написано з використанням фреймворку Flask мовою програмування Python. Даний застосунок має декілька ролей: адміністратора та клієнта. Адміністратор здійснює моніторинг роботи системи, створює та редагує події. Дана роль виконує функції суперадміністратора, який має повний контроль над системою. Зв'язок між компонентами проєктованої системи відображено на рис. 3.1.

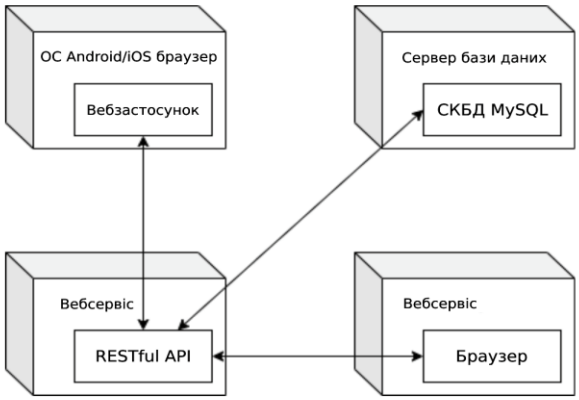

Рисунок 3.1 – Зв'язок між компонентами системи, що проєктується

Розглянемо функціонал та реалізовану архітектуру вебзастосунку для двох ролей користувачів більш детальніше. Система створена для прокладання автомобільного маршруту водіїв за допомогою аналізу відео з камер. Для цього необхідно окреслити внутрішню функціональність системи. З метою визначення функціональних вимог [52] було розроблено діаграми варіантів використання адміністратора системи та водія, які представлені на рис. 3.2 та 3.3. Основні функції адміністратора: авторизація, підключення

та налаштування камер, модерація користувачів, редагування та видалення камер. Тоді як для клієнта: авторизація, пошук місць за адресою, побудова маршруту.

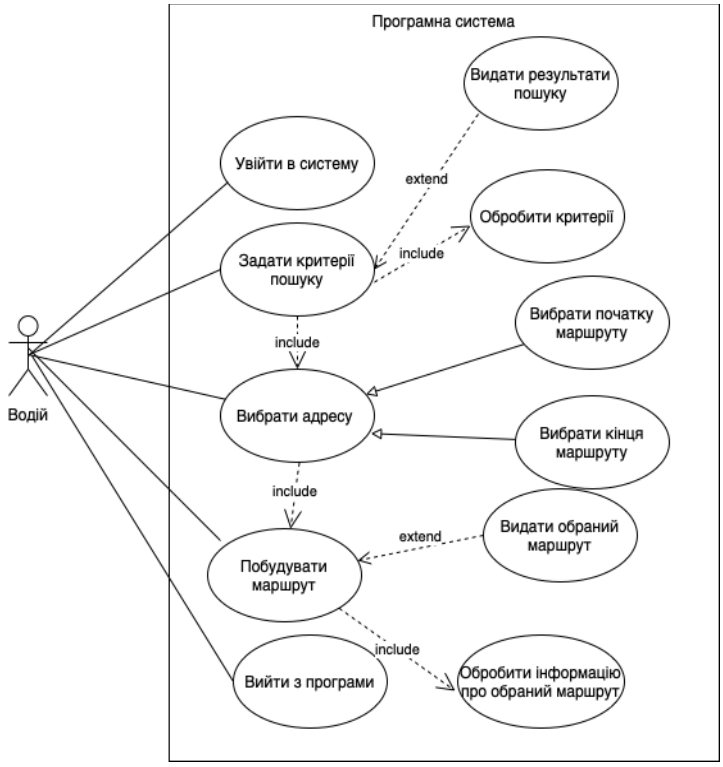

Рисунок 3.2 – Діаграма варіантів використання

Рисунок 3.3 – Діаграма варіантів використання

Оскільки вебсервіс, що розробляється є досить комплексним, розглянемо опис основних прецедентів. У табл. 3.2 наведено опис прецеденту «Адміністрування мережі камер відеоспостереження».

Таблиця 3.2 – Опис прецеденту «Адміністрування мережі камер відеоспостереження»

Use case section	***Description***
Use case Name	Адміністрування мережі камер відеоспостереження
Scope	Програмний комплекс системи навігації водіїв.
Level	Успішно виконати процес налаштувань, отримати доступ до вебкамер
Primary Actor	Адміністратор
Stakeholders and interests	1. Адміністратор – отримання відеопотоку з вебкамери
Preconditions	1. Адміністратор налаштував мережеві параметри вебкамер з можливістю отримання кадрів за стандартом кодування двійкових даних base64.
Success guarantee	1.Адміністратор використовує браузер з підтримкою HTML5, CSS3; 2.Адміністратор має використовувати стабільне підключення до мережі Інтернет.
Main Success Scenario	1.Адміністратор має можливість побачити результати визначення швидкості руху автівок вебкамерою.
Extensions	1.Адміністратор намагається налаштувати вебкамеру при змінному значенні: - адміністратор проходить процес авторизації та аутентифікації; - адміністратор потрапляє на сторінку з доступними вебкамерами для спостереження за процесом визначення швидкості руху на ділянці дороги, натискає кнопку редагування мережевих характеристик; - система відображає повідомлення про неможливість налаштувань вебкамери у випадку мережевих технічний несправностей.
Special Requirements	1. Швидкість доступу до мережі Інтернет має бути більшою ніж 20 Мб/сек.
Frequency of Occurrence	< 20%.
Miscellaneous	1. Чи активний обліковий запис адміністратора.

У табл. 3.3 наведено опис прецеденту «Фіксація швидкості автомобілів у автоматичному режимі».

Таблиця 3.3 — Опис прецеденту «Фіксація швидкості автомобілів у автоматичному режимі»

Use case section	***Description***
Use case Name	Фіксація швидкості автомобілів у автоматичному режимі.
Scope	Програмний комплекс системи навігації водіїв.
Level	Функціональний модуль розпізнавання об’єктів на кадрах з вебкамер.
Primary Actor	Адміністратор.
Stakeholders and interests	1. Адміністратор – конфігурація необхідних форматів кадрів та характеристик якості роботи алгоритмів розпізнавання.
Preconditions	1. Адміністратор пройшов процес авторизації і відкрив сторінку відображення з зображення проміжкових етапів розпізнавання.
Success guarantee	1. Адміністратор використовує броузер з підтримкою *HTML5, CSS3;* 2. Адміністратор має використовувати стабільне підключення до мережі Інтернет.
Main Success Scenario	1. Адміністратор має можливість отримати значення швидкості руху автівок ділянкою дороги.
Extensions	1. Адміністратор бачить відсутність значення швидкості, яка мала б бути визначена вебкамерою: - адміністратор переходить на сторінку конфігурації етапів розпізнавання; - адміністратор обирає необхідний xml файл з описом характеристик каскадів Хаара для розпізнавання; -адміністратор обирає необхідне налаштування фонового зображення ділянки; - система розпізнавання зберігає значення і відображає повідомлення про валідацію полей конфігурації; 2. Водію відображається повідомлення про відсутність даних про дану ділянку дороги: - водій обирає опцію очікування реконфігурації маршруту в разі відновлення визначення швидкості вебкамерою на ділянці дорогу;

	- обрає інший маршрут в ручному режимі; - водій підтверджує налаштування про можливість неповної інформації про швидкість спричиненої технічними проблема вебкамер на ділянці дороги.
Special Requirements	1. Швидкість доступу до мережі Інтернет має бути більшою ніж 20 Мб/сек.
Frequency of Occurrence	> 80%
Miscellaneous	1.Чи активні облікові запис водія чи адміністратора. 2.Чи є доступ до бази даних.

У табл. 3.4 наведено опис прецеденту «Побудова маршруту руху водія в автоматичному режимі».

Таблиця 3.4 -– Опис прецеденту «Побудова маршруту руху водія в автоматичному режимі»

Use case section	***Description***
Use case Name	Побудова маршруту руху водія в автоматичному режим.
Scope	Програмний комплекс системи навігації водіїв.
Level	У клієнтському застосунку успішно отримувати оновлену інформацію швидкість руху на ділянках дороги.
Primary Actor	Водій.
Stakeholders and interests	1. Водій – отримання маршруту з максимально можливою швидкістю руха ділянками маршруту, щоб швидше дістатись від початкової до кінцевою точки.
Preconditions	1.Вебзастосунок водія використовує достатню кількість оперативної пам’яті мобільного пристрою для швидкої обробки даних.
Success guarantee	1.Водій використовує броузер з підтримкою *HTML5, CSS3;* 2.Водій має використовувати стабільне підключення до мережі Інтернет.

Main Success Scenario	1. Водій має доступ до оновлення системи.

Розглядаючи основні прецеденти системи, можна зрозуміти, які функціональні можливості є основними і першочерговими. У випадку універсального програмного комплексу системи навігації водіїв це – підтримка актуального стану програмного забезпечення, наявність гнучкого налаштування та можливість контролю основного процесу – розпізнавання об’єктів машин на ділянках доріг.

Діаграма діяльності UML дозволяє більш детально візуалізувати конкретний випадок використання. Її також можна використовувати для відображення потоку подій у бізнес-процесі. Вони можуть бути використані для вивчення бізнес-процесів з метою визначення їх потоку та вимог [54]. Діаграми діяльності також можна використовувати для опису ролей і сфер в бізнесі – іншими словами, хто за що відповідає в бізнесі. Ролі та зони відповідальності задокументовані у вигляді стовпців (доріжок UML) на діаграмі діяльності. Swimlanes (плаваючі доріжки) показують, які представники бізнесу беруть участь у реалізації робочого процесу [6].

Розглянемо діаграми діяльності безпосередньо для проєкту, що розробляється: діаграми діяльності «Адміністрування мережі камер відеоспостереження» (рис. 3.4), «Фіксація швидкості автомобілів у автоматичному режимі» (рис. 3.5), «Побудова маршруту руху водія в автоматичному режимі» (рис. 3.6).

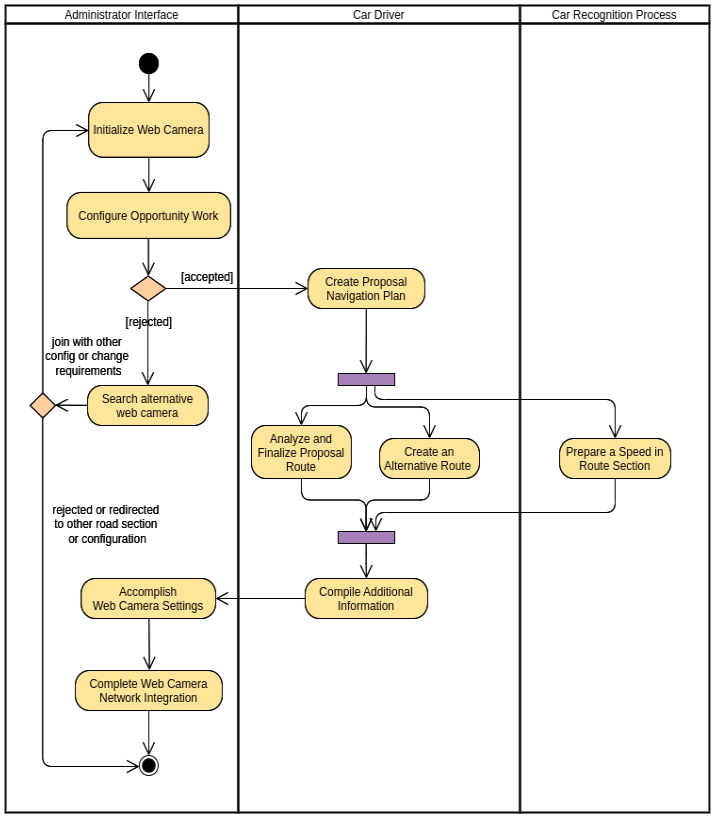

Рисунок 3.4 – Діаграма діяльності «Адміністрування мережі камер відеоспостереження»

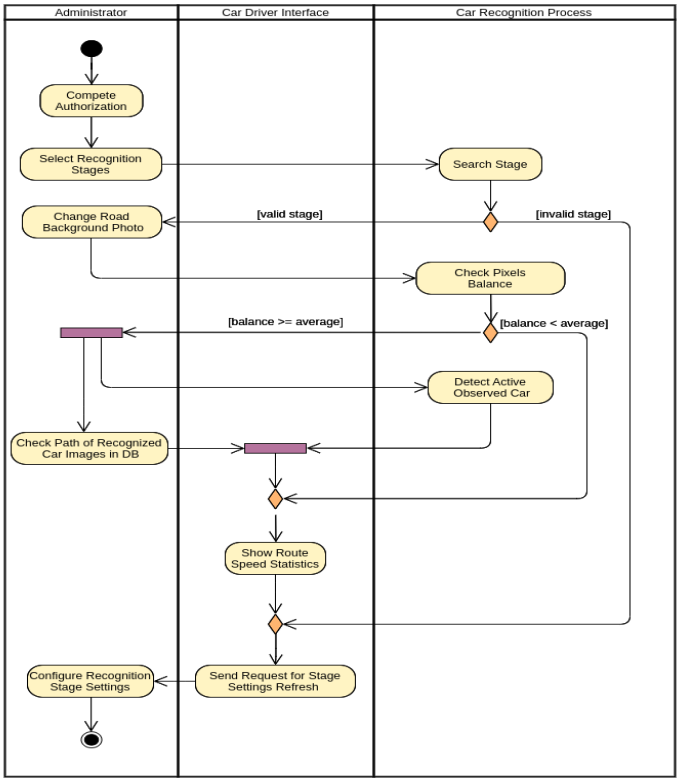

Рисунок 3.5 – Діаграма діяльності «Фіксація швидкості автомобілів у автоматичному режимі»

Розглянемо детальніше діаграму діяльності «Побудова маршруту руху водія в автоматичному режимі» (рис. 3.6). У лівій колонці, яка слугує областю відображення процесів пов’язаних із водієм автомобіля («Car Driver»), першою активністю є формування ключових точок маршрутів «Way points». Тобто це завдання спрямовано на формування даних точок. Потім з процесу, з яким взаємодіє водій, активність за допомогою повідомлення з даними передається в «Asigment». Далі ці завдання переходять до області процесів, пов’язаних із розпізнаванням машин («Car Recognition Process») вебкамерою. Якщо активність аналізу точок маршруту для планування навігації («Analyze Waypoints for Navigation Planning») більша за 20 сек., тоді водію відображається повідомлення про очікування виконання завдання, якщо менша за 20 сек., то починається активність завершення навігаційного планування («Complete Navigation Planning»). Якщо процес цієї активності відхилено, то процес завершується, а якщо ні, тоді виконується активність підтвердження планування («Submit Planning»). Планування переходить до області процесів з якими взаємодіє водій, тобто відбувається очікування завершення процесу обробки («Request Timeout»). Якщо процес очікування обробки запиту відбувся менше ніж за 20 сек., тоді водій отримує план руху маршрутом до заданої точки. Водій погоджує маршрут, тобто чи задовольняє він його чи ні. Якщо маршрут підходить, тоді маршрути зберігаються в БД з прив’язкою до карти (<<datastore>> Routes On Map). Далі виконується активність повернення маршрутів до процесу, який має виконувати активність ініціації (або початку) навігаційної підтримки на сервері.

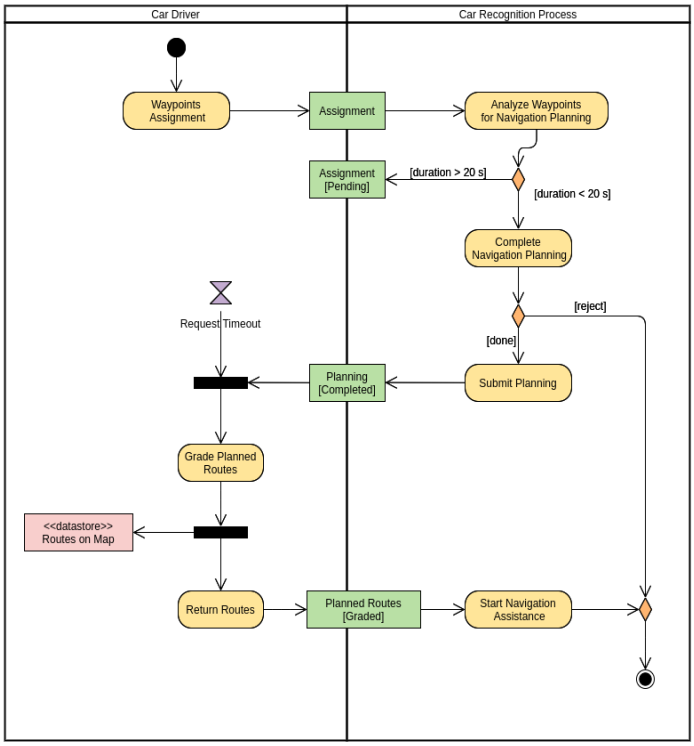

Рисунок 3.6 – Діаграма діяльності «Побудова маршруту руху водія в автоматичному режимі»

Достатньо велику кількість паралельних процесів включають діаграми, які застосовують для опису поведінки компонентів системи. На діаграмі діяльності кожен стан відповідає за виконання певних простих операцій. Лише за умови завершення однієї операції відбувається перехід до наступного стану. Тобто можна вважати діаграму діяльності приватним випадком діаграми станів. Візуалізовуючи особливості реалізації операцій класів, за умови необхідності надання алгоритмів їх виконання, можна відобразити основні напрямки використання діаграм діяльності.

Також, представимо діаграму розгортання (рис. 3.7). Вона відображає графічне представлення ПЗ. Такий вид діаграми допомагає організувати компоненти раціонально, від чого залежить продуктивність системи та її безпека [8].

Рисунок 3.7 – UML діаграма розгортання роботи застосунку

Спільні методи та дії мають об’єкти системи, пов’язані з алгоритмами, що створюють, редагують, видаляють камери та користувачів. Наприклад, для виконання авторизації у систему треба ввести дані у форму та зібрати їх у один об’єкт для виклику методу, який надсилає запит на відповідну адресу та перенаправляє зібрані дані на сервер. Отримавши їх сервер виконує авторизацію та повертає токен з об’єктом користувача на

клієнтську частину, де зберігається інформація. На діаграмі послідовностей зображено цей алгоритм (рис. 3.8), який описує даний процес.

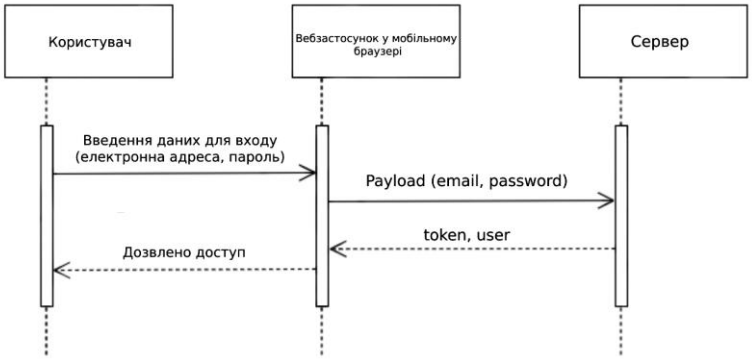

Рисунок 3.8 – Діаграма послідовності для функції «Авторизація»

До функцій працюючих із даними застосовується подібний алгоритм, але зокрема, необхідно у обов'язковому порядку під час наступних запитів надсилати роль користувача та токен, що повертаються при авторизації від сервера.

3.3 Проєктування архітектури вебзастосунку

Ефективне проєктування архітектури вебзастосунку є складовою частиною для створення програмного продукту. До вебастосунку входять: API, що надає методи керування об'єктами даних та програні модулі, що написані з використанням вебфреймворку Flask. Для вебзастосунків широкий набір функцій представляє вебфреймворк Flask. Використання RESTful архітектури дозволить відокремити функціональність клієнтської програми від програмних інтерфейсів для роботи з базою [7].

Прозорість зв'язків між компонентами системи для сервісних служб, простота уніфікованого інтерфейсу, переміщення програмного коду разом з даними за рахунок перенесення компонентів системи є основними перевагами даної архітектури. Частини системи працюють незалежно, оскільки для них була побудована клієнт-серверна модель. MySQL було обрано у якості системи управління базами даних.

Мову структурованих запитів SQL, яку підтримує реляційна СКБД, може використовуватися як сервер SQL. Її перевагами є безпека, швидкість, надійність і мобільність.

Діаграми класів в об'єктно-орієнтованому моделюванні є основними будівельними блоками, що застосовуються для детального моделювання, перетворення моделей в програмний код та для агального концептуального моделювання структури програми. Іноді дані діаграми використовують для моделювання даних. Класи у цих діаграмах представлені основними елементами взаємодії у застосунку як такі, що підлягають програмуванню.

Під час проєктування системи визначається кількість класів, які групуються в діаграму класів, що дозволяє визначити статичні зв'язки між ними. Класи концептуального дизайну у детальному моделюванні часто діляться на підкласи [53]. Розглянемо діаграму класів безпосередньо для проєкту, що розробляється і проаналізуємо ключові моменти (рис. 3.9).

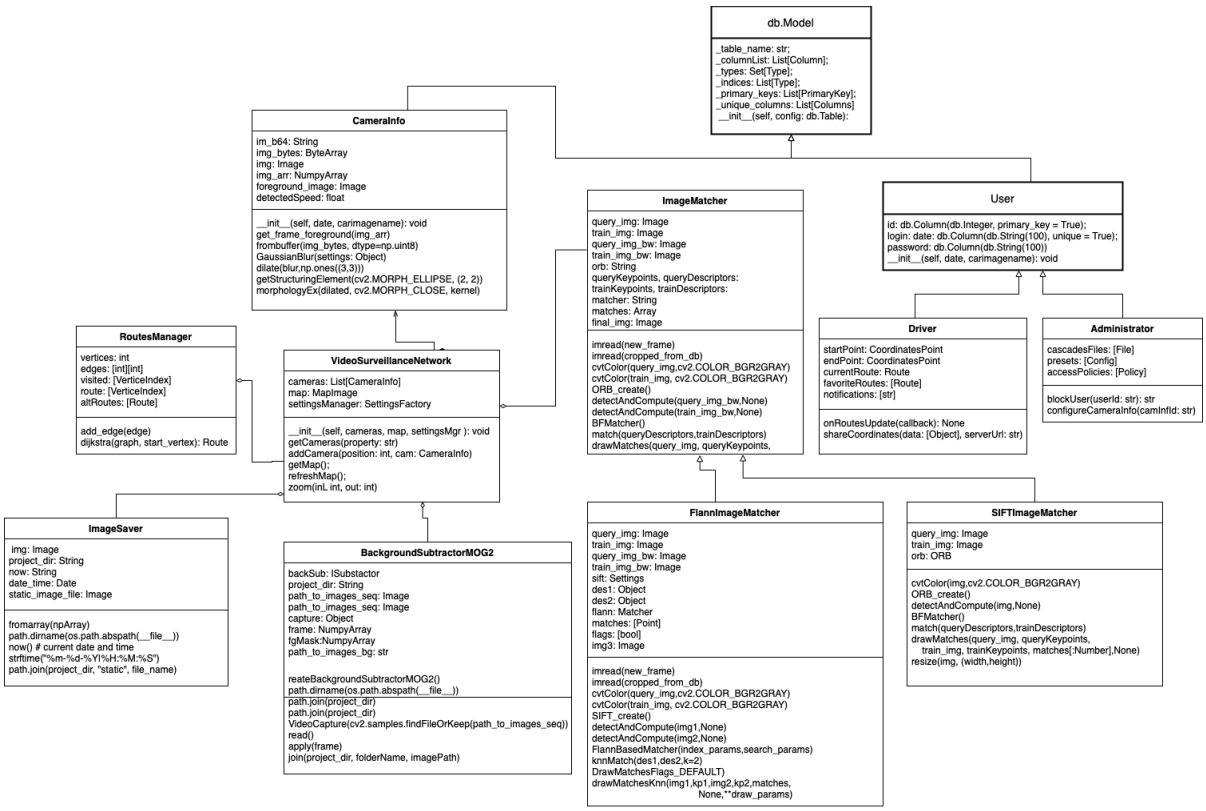

Рисунок 3.9 – Діаграма класів вебзастосунку

Отже, дана система побудована на архітектурі, в якій за функціональність відповідають сервіси, на які концептуально розділена система. Для реалізації кожен сервіс має власний інтерфейс. Тобто через єдиний інтерфейс можна додати кілька сервісів.

У проекті реалізовані наступні класи: db.Model (клас бази даних), User (користувачі), Driver (водій), Administrator (адміністратор), CameraInfo (інформація про камери), VideoSurveillanceNetwork, RoutesManager, ImageSaver, BackgroundSubstractorMOG2, ImageMatcher, FlannImageMatcher, SIFTImageMatcher.

Кожен з класів має свій певний набір полів та функцій.

Driver:

onRoutesUpdate() – оновлення маршруту;

shareCoordinates() – відкриття даних про координати іншим клієнтським класу.

Administrator:

blockUser() – блокування користувача;

configureCameraInfo() – конфігурація інформації про камеру.

CameraInfo:

__init__() – конструктор;

get_frame_foreground() – вилучення фону зображення;

frombuffer() – конвертація байтів зображення;

GaussianBlur() – Розмивання Гауса (фільтрація зображень за допомогою функції Гауса);

dilate(blur,np.ones((3,3))) – фільтрування шуму за допомогою розмиття;

getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2)) – функція NumPy, для визначення структурного елементу для аналізу зображення;

morphologyEx(dilated, cv2.MORPH_CLOSE, kernel) – функція вказує на операцію MORPH_CLOSE на вхідному зображенні;

ImageMatcher:

imread(new_frame) – функція читання зображення нового кадру;

imread(cropped_from_db) – функція читання зображення для активної машини;

cvtColor() – функція для перетворення зображення з одного колірного простору в інший;

ORB_create() – функція для створення об’єкта;

detectAndCompute() – функція для виявлення ключових точок та обчислення їхніх дескрипторів;

BFMatcher() – функція використовується для зіставлення функцій одного зображення з іншим;

drawMatches() – функція підбору відповідників до кінцевого зображення, що містить декілька зображень.

VideoSurveillanceNetwork:

__init__(self, cameras, map, settingsMgr) – конструктор;

getCameras() – функція отримання властивостей камери;

addCamera() – функція додавання камери;

getMap() – функція отримання головної карти з урахуванням кадру;

refreshMap() – функція оновлення карти з урахуванням кадру;

zoom() – функція масштабування кадрів відеопотоку.

RoutesManager:

add_edge() – функція додавання ребра між двома вершинами;

dijkstra(): Route – функція розрахунку найкращого шляху з критерієм максимальної швидкості руху ділянкою дороги;

BackgroundSubtractorMOG2:

createBackgroundSubtractorMOG2() – функція створення екземпляру класу сабстрактору;

path.dirname(os.path.abspath(__file__)) – функція, що повертає базове ім’я шляху, заданого як параметр у функції;

path.join(project_dir) – метод об’єднання шляхів до файлу;

VideoCapture() – функція, що дозволяє працювати з відео шляхом його запису за допомогою вебкамери у реальному часі або за допомогою відеофайлу;

read() – функція, що повертає кортеж кадрів відео;

apply(frame) – функція застосування кадру;

join() – функція об’єднання рядка з ітерованим об’єктом.

ImageSaver:

fromarray(npArray) – функція повернення зображення з перетворенням масиву numpy;

now() – отримання поточної дати і часу;

strftime() – функція, яка використовується для перетворення формату дати й часу.

FlannImageMatcher:

SIFT_create() – функція створення об’єкту Sift;

detectAndCompute() – функція виявлення ключових точок та обчислення їх дескрипторів;

FlannBasedMatcher(index_params,search_params) – функція, яка використовується для відповідності дескрипторів на обох зображеннях за алгоритмами Фланну;

knnMatch() – функція пошуку двох найближчих сусідів;

DrawMatchesFlags() – функція, що малює збіги ключових точок із двох зображень у вихідному зображенні;

drawMatchesKnn() – функція, яка використовується для встановлення збігів між ключовими точками двох зображень.

SIFTImageMatcher:

resize(img, (width,height)) – кортеж для визначення параметрів нового зображення.

У програмній інженерії шаблон проєктування – це загальне повторюване рішення, яке часто зустрічається при розробці програмного забезпечення. Це шаблон вирішення проблеми, який можна використовувати в багатьох різних ситуаціях. Шаблони проєктування можуть прискорити процес розробки, надаючи перевірені парадигми розробки. Ефективна розробка програмного забезпечення вимагає розгляду проблем, які можуть стати видимими лише на більш пізньому етапі реалізації. Повторне використання шаблонів проєктування допомагає уникнути серйозних проблем і покращує читаємість коду для програмістів і архітекторів, знайомих з шаблонами.

Шаблони проєктування надають загальні рішення, задокументовані у форматі, що не вимагає деталей, пов’язаних з конкретною проблемою. Крім того, шаблони використовують добре відомі та понятні назви для взаємодії з програмним забезпеченням. Загальні шаблони проєктування можуть бути покращені з часом, що робить їх більш надійними, ніж спеціальні проєкти.

Усі шаблони проєктування пов’язані зі створенням екземплярів класу. Цей шаблон можна далі розділити на шаблони створення класів і шаблони створення об’єктів. У той час як шаблони створення класів ефективно використовують успадкування в процесі створення екземплярів, шаблони створення об’єктів ефективно використовують делегування для виконання роботи.

Було використано адаптер завантаження зображень ImageBase64, що перетворює рядки з кодуванням Base64 у вихідні дані масивів або зображень . Дані зображень через мережу передаються у base64 форматі, ми використовуємо адаптери для наступного використання зображень у форматі NumPy масиву або конкретного формату зображень: png, jpg. Алгоритми комп'ютерного зору на різних етапах можуть використовувати різні формати зображень. Додаткове або повторне завантаження вимагає серверних ресурсів, тому адаптери зображень конвертують формати зображень, що можуть зберігатися разом з метаданими і використовуватись в конвеєрних процесах розпізнавання об'єктів без додаткових запитів.

NPArrayMatcherAdapter – це паттерн проєктування Adapter для фундаментальни бібліотеки для контейнерів масивів у стеку Python Scientific Computing. Багато бібліотек Python, включаючи SciPy, Pandas та OpenCV, використовують NumPy ndarray як загальний формат для обміну даними. Ці бібліотеки можуть створювати, працювати та працювати з масивами NumPy (рис. 3.10).

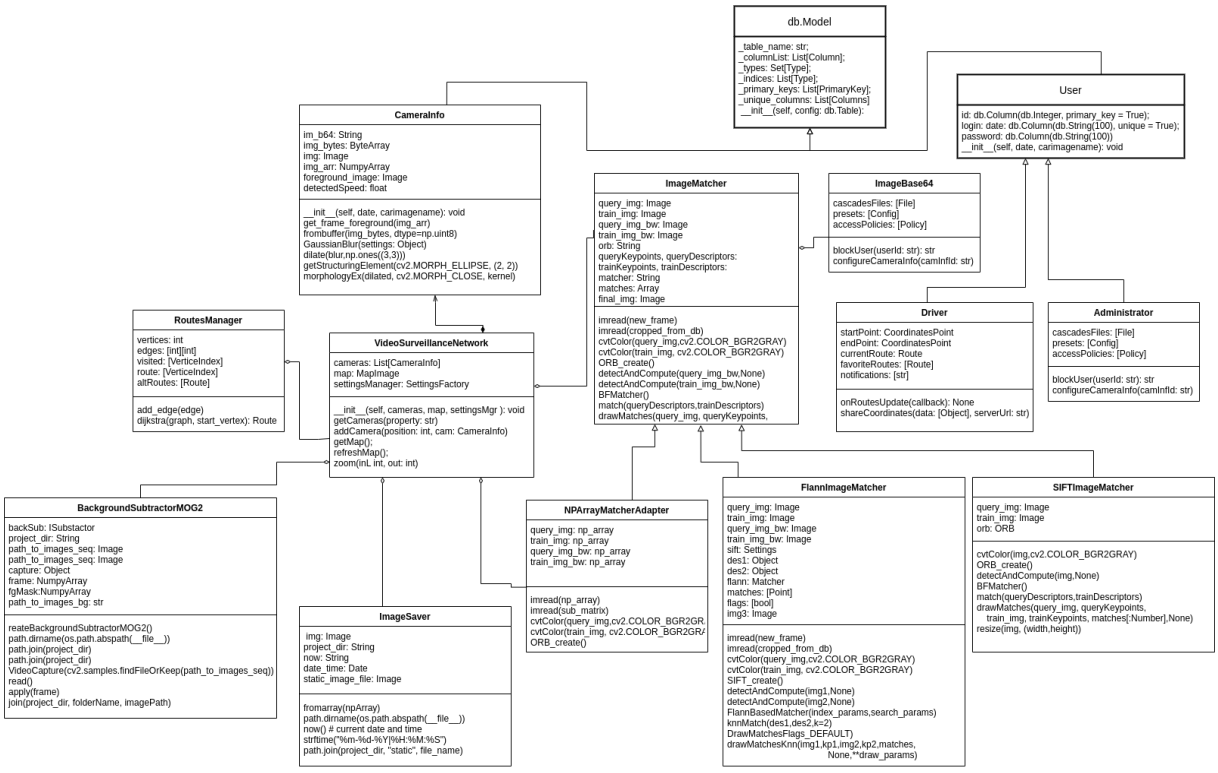

Рисунок 3.10 – Інтеграція патерну адаптера в архітектуру вебзастосунку

Таким чином, адаптер дозволяє двом несумісним інтерфейсам працювати разом. Інтерфейси можуть бути несумісними, але внутрішня функціональність повинна відповідати потребам. Шаблон проєктування адаптера дозволяє несумісним класам працювати разом, перетворюючи інтерфейс одного класу на інтерфейс, очікуваний клієнтами. Цей шаблон адаптера використовує кілька поліморфних інтерфейсів, що реалізують або успадковують як очікуваний інтерфейс, так і існуючий інтерфейс. Зазвичай

очікуваний інтерфейс створюється як чистий клас інтерфейсу, які підтримують множинне успадкування класів.

Висновки до розділу 3

Розробка програмних проєктів має етапи, зокрема початковими є обрання стеку технологій, визначення функціональних вимог до розробленого вебсервісу. Стек технологій для програмного комплексу було обрано з розрахунком на стабільність, гнучкість та продуктивність програмного середовища. Для реалізації вебсервісу були обрані такі технології, як Python, Flask, JS (HTML, CSS), OpenCV, MySQL. Для створення вебсервісу було обрано мову програмування Python через простий синтаксис та наявність фреймворків, необхідних для виконання поставленої задачі. Python працює на різних апаратних платформах на яких має однаковий інтерфейс. До інтерпретатора Python можна додавати модулі низького рівня, бібліотеки. Flask ідеально підходить для експериментів з архітектурою, бібліотеками та новими технологіями. Flask також дуже добре працює з вебзастосунками, які можуть бути розділені на безліч невеликих сервісів, які не вимагають багато коду для досягнення необхідного результату. JavaScript займає унікальне положення як найбільш поширена мова браузера, повністю інтегрована з HTML/CSS. JavaScript є єдиною технологією браузера, що поєднує подібну інтеграцію. OpenCV у розгорнутому вигляді застосовується для багатьох задач, зокрема, прискорює використання машинного сприйняття в комерційних продуктах та створює загальну інфраструктуру для програм комп’ютерного зору. У ряді випадків виділення об’єктів на зображеннях використовується як основний етап обробки зображення для вирішення складних завдань, а саме для розпізнавання машин. Як СУБД MySQL використовує SQL для управління даними всередині бази даних. Це дозволяє використовувати SQL для створення, зміни та вилучення даних з реляційної бази даних. SQLAlchemy обрано як набір інструментів SQL з відкритим вихідним кодом та об’єктно-реляційний картограф (ORM) для мови програмування Python.

Даний застосунок має декілька ролей: адміністратора та клієнта (водія). Основні функції адміністратора: авторизація, підключення та налаштування камер, модерація користувачів, редагування та видалення камер. Тоді як для клієнта: авторизація, пошук місць за адресою, побудова маршруту. Оскільки вебсервіс, що розробляється є досить комплексним, було розглянуто та описано основні прецеденти. Розглядаючи основні прецеденти системи, можна зрозуміти, які функціональні можливості є основними і першочерговими. У випадку універсального програмного комплексу системи навігації водіїв це – підтримка актуального стану програмного забезпечення, наявність гнучкого налаштування та можливість контролю основного процесу – розпізнавання об’єктів машин на ділянках доріг. Також було представлено діаграму розгортання вебсервісу та діаграму послідовності для функції «Авторизація». Дана система побудована на архітектурі, в якій за функціональність відповідають сервіси, на які концептуально розділена система. Для реалізації кожен сервіс має власний інтерфейс. Тобто через єдиний інтерфейс можна додати кілька сервісів. Також було описано реалізовані класи та їх функції.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка прототипу серверної частини вебсервісу аналізу дорожнього трафіку

Для системи виявлення транспортних засобів буде достатньо роботи як із зображеннями, так і з відео, буде використовуватись OpenCV для виконання всіх операцій обробки зображень. Для для виявлення та класифікації автомобілів було обрано каскадний класифікатор Хаара [55]. Для серверної частини, яка виконує отримання медіа даних про дорожній трафік необхідно виконати імпорт необхідних бібліотек. Після того, як було підключено необхідні бібліотеки, виконуємо конфігурацію головного застосунку Flask.

У налаштуваннях екземпляру застосунку Flask треба вказати URI СКБД MySQL для ORM SQLAlchemy. До нього входять: username, пароль, доменне ім'я хостингу та назва бази даних cardetectdb. Екземпляр застосунку Flask передається в якості аргументу до конструктору класу SQLAlchemy для створення db – екземпляру класу для роботи зі SQLAlchemy. Для роботи механізму спільного використання ресурсів між джерелами екземпляр app потрібно передати в конструктор CORS. Даний механізм, що використовує додаткові HTTP-заголовки, щоб дати можливість агенту користувача отримувати дозволи на доступ до вибраних ресурсів із сервера джерела (домену), відміно від того, що сайт використовує в даний момент. Агент клієнтського програмного забезпечення робить перехресний запит HTTP з іншого джерела, якщо джерело поточного документа відрізняється від запитуваного ресурсу доменом, протоколом або портом. Після цього визначаються шляхи до робочих директорій STATIC_FOLDER та STATIC_DATA_FOLDER. Наступним етапом є декларація моделей у SQLAlchemy.

При першому зверненні серверної частини сервісу моніторингу спрацьовує декоратор @app.route('/'), який рендерить шаблон render_template("/taxi-master/index.html") де відображається головна сторінка вебсайту асистентського інтерфейсу для водія. Після введення вхідних даних про початкові точки руху спрацьовує декоратор @app.route('/service', methods=['GET', 'POST']), в якому виконується розрахунок маршруту за алгоритмом Дейкстри. Для початку роботи алгоритму необхідно ініціалізувати граф вулиць міста даними про ділянку дороги на якому встановлена камера та про швидкість руху автомобілів на ній.

Реалізація алгоритму Дейкстри мовою програмування Python. Для сортування та відстеження вершин, які ще не було відвідано, використовуємо PriorityQueue: *from queue import PriorityQueue.* Тепер реалізуємо конструктор для класу під назвою *Graph.* Визначаємо кількість вершин у графі як аргумент та ініціалізуємо три поля. V: представляє кількість вершин у графі, edges: представляє список ребер у формі матриці. Для вузлів u та v, self.edges[u][v] = weight – ребра. Visited: набір, який міститиме відвідані вершини.

Далі визначаємо функцію, що додає ребро до графіка: *def add_edge(self, u, v, weight): self.edges[u][v] = weight, self.edges[v][u] = weight.* Використовуємо алгоритм Дейкстри, що реалізовано в функції *def dijkstra(aGraph, start_vertex, target)*, код якої продемонстровано в додатку А.

Після компіляції програми отримуємо результат розрахунку відстаней між вершинами, що є точками встановлення камер для відслідковування швидкості руху на ділянці дороги. Відстань між початковим вузлом і кожним іншим вузлом у дереві обчислюється та виводиться. Після закінчення розгляду всіх сусідів поточного вузла, позначаємо поточний вузол як відвіданий і видаляємо його з невідвіданого набору *while len(unvisited_queue).* Розглядаємо вершину з найменшою відстанню: *uv = heapq.heappop(unvisited_queue).* Перебудовуємо вершини та розбиваємо кожний ваговий коефіцієнт. Переміщаємо у чергу всі невідвідані вершини. Для кожного нового

відвідування вузла ми перебудовуємо вершини: видаляємо всі елементи, повторно заповнюємо unvisited_queue, а потім об’єднуємо їх в купу.

Відвіданий вузол більше не перевірятиметься. Якщо немає невідвіданого вузла, алгоритм завершено. Збираються попередні вузли, починаючи з цільового вузла ('e'). У коді це робиться за допомогою функції *shortest()*. Сформований маршрут передається до функції render_template ("/taxi-master/service.html") з параметрами placefrom, placeto, route. У результаті користувач має побачити відображення карти з камерами та маршрутом, що виділений яскравим кольором.

Для відображення технічної інформації адміністратору необхідно клікнути на зображення камери на ділянці дороги. Декоратор @app.route('/camera-about') виконає рендеринг render_template("/taxi-master/camera-about.html") з параметрами name, address, camPosition. Адміністратору відображається інтерфейс конфігурації камери де він може виконати автоматичні експериментальні тести швидкості на ділянці дороги, яка визначається в результаті порівняння зсуву розпізнаної машини на декількох кадрах. Так як серверна частина, що відображає відеопотік автомобільного трафіку на ділянці дороги знаходиться на хостингу датацентру, а вебсервіс трансляції зображень з камери безпосередньо на ділянці дороги, то адміністратору потрібно робити декілька конфігураційних запитів на отримання кадрів до географічно віддаленого апаратного хостингу вебсервісу трансляції зображень. На рис. 4.1 зображено лістинг коду, який відправляє кадри зображень у base64 форматі до серверної частини аналізу швидкості дорожнього трафіку.

main.py

```
app.config['CORS_HEADERS'] = 'Content-Type'
cors = CORS(app)
@app.route('/')
def index():
    return 'cameras monitoring web service'
@app.route('/getframe', methods=['GET', 'POST'])
def get_frame():
    frame_data = request.get_json()
    api = 'https://mgybarev.pythonanywhere.com/api/frame'
    frame_num = frame_data['num'];
    image_file = f'cars-{frame_num}.png'
    with open(image_file, "rb") as f:
        im_bytes = f.read()
        im_b64 = base64.b64encode(im_bytes).decode("utf8")
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        payload = json.dumps({"image": im_b64, "date": datetime.now().strftime("%m-%d-%Y|%H:%M:%S")})
        response = requests.post(api, data=payload, headers=headers)
        print("Start frame sending...")
```

Рисунок 4.1 – Відправлення зображень з вебкамери на ділянці дороги

Для цього адміністратор має натиснути на кнопки виконання конфігурації «Config Frame 1», «Config Frame 2», «Config Frame 3». При натисканні «Config Frame 1» та надсиланні запиту спрацьовує метод який зв’язаний з end-point і обробляється @app.route('/api/frame', methods=['POST']), який викликає метод def get_frame(). При цьому з вебсервісу трансляції зображень з камери отримується перший кадр відносно якого буде починатись виконання розрахунку швидкості на ділянці руху. Після конвертації кадру з base64 до масиву байтів та конвертації до формату np.array виконується процес видалення фону асфальту та навколишнього середовища. На отриманому зображені 'cars-fg-mask.png' залишаються лише області автомобілів. На наступному кроці виконується попередня обробка методами: *cv2.imdecode(image,cv2.IMREAD_COLOR), cv2.cvtColor(img_arr,cv2.COLOR_BGR2GRAY), cv2.GaussianBlur(grey,(5,5),0).*

До зображення кадру мають бути застосовані морфологічні перетворення. Це операції, що засновані на формі зображення та виконуються на бінарних зображеннях, яким потрібні два входи: один – це вихідний образ, другий – структурний елемент або ядро, яке визначає характер роботи: *kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2)), closing = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE, kernel).*

Для початку розпізнавання потрібно завантажити до car_cascade_src файл 'train_Cas1.xml' з функціями, що є окремими значеннями, отриманими шляхом віднімання суми пікселів під білим прямокутником із суми пікселів під чорним прямокутником при навчанні. Встановити каскад класифікаторів можна за допомогою методу: *cv2.CascadeClassifier()*. Після цього виконується розпізнавання за допомогою методу *cv2.detectMultiScale()*, який повертає прямокутники меж для виявлених автомобілів на ділянці дороги.

Але під час розпізнавання можуть виникати ситуації некоректного розпізнавання, які будують прямокутники меж в області зображення, що не містять автомобілів. Було модифіковано процес розпізнавання за допомогою додавання етапу співставлення зображення 'cars-fg-mask.png' з поточним кадром (рис. 4.2).

```
max_gray_percentage = 0,  max_gray_countour_tuple = (), cnt = 0
    for (x,y,w,h) in cars:
        gray_percentage = 0, gray_counter = 0
        for i in range(y, y+h):
            for j in range(x, x+w):
            (b, g, r) = foreground_image[i, j]
                if (b == 127 and g == 127 and r == 127): #if pixel is gray
                    gray_counter +=1, gray_percentage = gray_counter / (h) * (w)
        if ((w > 320 and w < 520 ) and (y+h < cvImg.shape[0] * 0.8) and (gray_percentage > 10)):
            if (gray_percentage > max_gray_percentage):
                max_gray_percentage = gray_percentage, max_gray_countour_tuple = (x,y,w,h)
            cv2.rectangle(img_arr,(x,y),(x+w,y+h),(255,0,0), 5), cnt += 1
```

Рисунок 4.2 – Код процесу розпізнавання авто

Прямокутники, які не міститимуть кількості пікселів кольору (b==127 and g==127 and r==127) вищої за допустимі межі (gray_percentage > 10) не будуть рахуватись межами області зображення, що містять автівки. Лише відібрані прямокутники, які задовольняють вищевказаним вимогам будуть слугувати областями відносно яких можна виконувати аналіз швидкості руху автомобілів в наступних кадрах. Виділяємо необхідний прямокутник max_gray_countour_tuple, виокремлюємо його у cropped_contour та зберігаємо шлях його файлу в БД. Також в моделі CameraInfo зберігаються дата створення зображення активної машини в кадрі та координати центру прямокутника з активною машиною в кадрі. Потім клієнтській частині відправляється системна інформація щодо статусу розпізнавання у першому кадрі та список зображень згенерованих в процесі розпізнавання. Для того, щоб виконати аналіз швидкості потрібно отримати другий кадр від вебсервісу. Для відправлення зображень з вебкамери на ділянці дороги адміністратор в режимі конфігурації має натиснути кнопку «Config Frame 2» або «Config Frame 3». Різниця між ними полягає у значенні інтервалу отримання кадру відносно першого кадру.

Якщо в БД знаходиться зображення з активною машиною, яку було знайдено в попередньому кадрі: *cam = CameraInfo.query.order_by(CameraInfo.id.desc()).first()*. Тоді можна розпочати процес пошуку центру прямокутника *curFrameCenter* з активною машиною за допомогою методу *searchCurrentCarInFrameFlann()* в якому використовуються: *original_image* – поточний кадр, *cam.cur_car_img* – зображення активної машини яку потрібно детектувати, *dict(coords = cam.cur_car_coords, separator= separator))* – координати центру активної машини в попередньому кадрі. Запускаємо детектор SIFT: *sift=cv2.SIFT_create()*. Знаходимо ключові точки та дескриптори за допомогою SIFT: *kp1, des1=sift.detectAndCompute(img1,None), kp2, des2=sift.detectAndCompute(img2,None)*. Параметри FLANN (рис. 4.3):

```
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
```

Рисунок 4.3 – Параметри FLANN

Потрібно відобразити позитивні збіги, тому створюємо маску: *matchesMask = [[0,0] for i in range(len(matches))]*. Тестуємо співвідношення згідно з документом Лоу. Малюємо пари фіолетовим кольором, щоб переконатися, що результат правильний (рис. 4.4):

```
matchesCount +=1
            cv2.circle(img1, (int(pt1[0]),int(pt1[1])), 5, (255,0,255), 3)
            X.append(int(pt1[0])), Y.append(int(pt1[1]))
        draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0), matchesMask = matchesMask,
                   flags = cv2.DrawMatchesFlags_DEFAULT)
img3=cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
```

Рисунок 4.4 – Код перевірки результату

Результатом буде розрахунок дистанції зсуву центрів активної машини у двох кадрах [56]. Розділивши дистанцію зсуву на інтервал часу між двома кадрами ми отримаємо значення швидкості, яке буде доступне у змінній speed. Системні дані відправляється до клієнтської частини системна інформація щодо статусу розпізнавання у другому кадрі та список зображень згенерованих в процесі розпізнавання, проте з вже визначеною швидкістю, яка відобразиться адміністраторові.

4.2 Розробка адаптивного інтерфейсу вебзастосунку для навігації водіїв

На початку використання клієнтського вебзастосунку керувальник транспортним засобом повинен обрати початкову та кінцеву точку руху. Камера буде виконувати моніторинг дороги в напрямку попередньої вершини зі встановленою камерою. Основними інструментами для розробки клієнтського застосунку було обрано JS, HTML, CSS. Проаналізувавши існуючи альтернативи було обрано Twitter Bootstrap та для забезпечення динамічності користувацького інтерфейсу наступні JS бібліотеки: easing.min.js, hoverIntent.js, jquery-ui.js, jquery.ajaxchimp.min.js, jquery.magnific-popup.min.js, jquery.nice-select.min.js, mail-script.js, superfish.min.js. Функціональність, яка пов’язана з бізнес-логікою побудови маршруту з урахуваннями заторів на вулицях міста, аналізом швидкості руху автівок та адміністрування камер відеоспостереження реалізовано в файлі main.js (рис. 4.5).

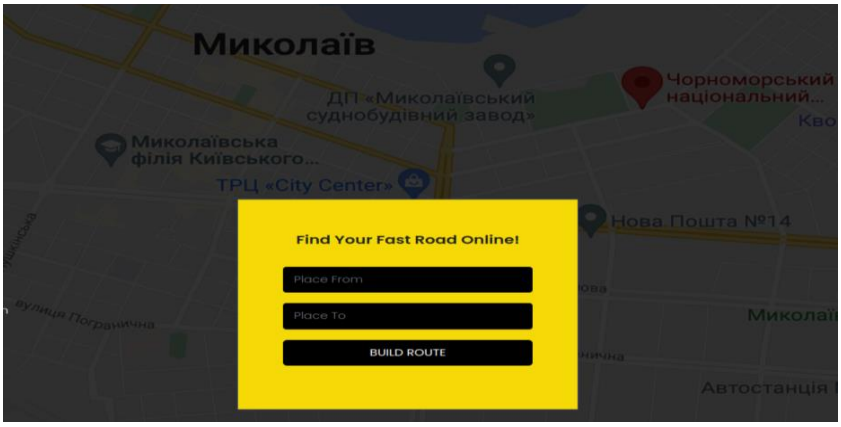

Рисунок 4.5 – Форма для вводу інформації про маршрут руху

Для того, щоб серверна частина отримала початкові дані і почала будувати маршрут руху, на основі аналізу швидкості міського трафіку, водієві потрібно заповнити форму. Форма відправляє POST запит до серверної кінцевої точки "/service" вказаної в параметрі action тегу form. У відповідь з серверу клієнтська частина отримує рендеринг render_template("/taxi-master/service.html", placefrom, placeto, route). Отримані дані placefrom, placeto та маршрут route зберігаються у об’єкті. Клас CamerasGraph

використовується для побудови мапи з відрізками доріг та розташуванням камер, що реалізоване у функції обробнику події mapImg.onload() для зображення карти (рис. 4.6).

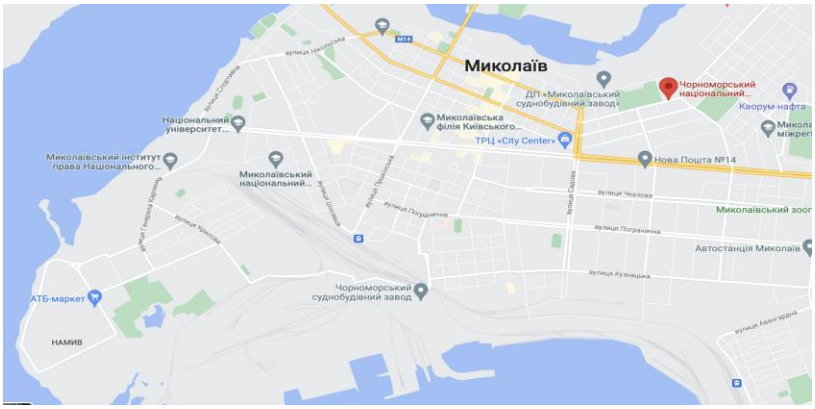

Рисунок 4.6 – Фонове зображення карти

На основі графу CamerasGraph за допомогою функції drawRoute(routeData) відображається оптимальний маршрут з початкової до кінцевої точки маршруту руху водія. Лінії графу будуються функціями ctx.moveTo(drawFrom.pos[0], drawFrom.pos[1]) та ctx.lineTo(toX, toY). У залежності від того, чи це граф можливих маршрутів, чи це розрахований маршрут кольори ліній встановлюються функцією ctx.strokeStyle = '#HEXCOLOR' (рис. 4.7).

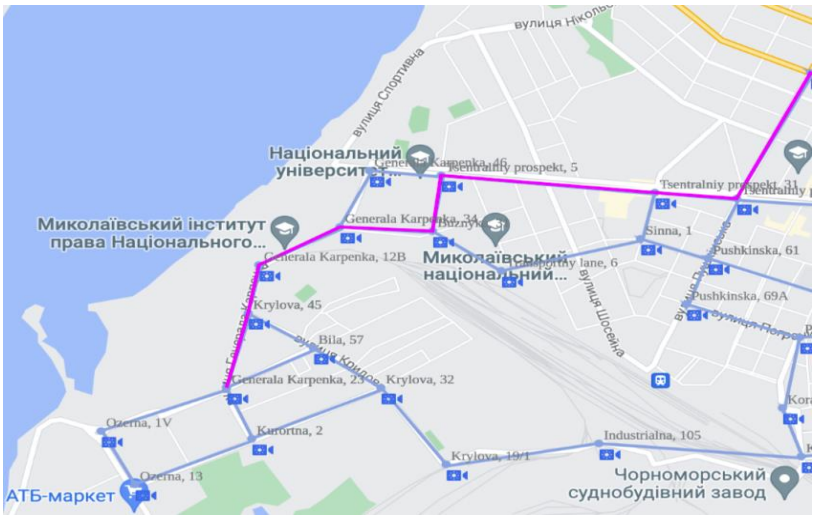

Рисунок 4.7 – Фонове зображення карти

Зображення камер на мапі малюються у вершинах графу за допомогою функції ctx.drawImage(camera_image, fromX, fromY + 4). Так, як зображення камер є відмальованими на зображенні карти, то постає розрахункове завдання визначити подію попадання координат кліку миші в деякий окіл навколо зображень камер. Єдиними даними які є доступними для використання в визначенні кліку по камері є координати вершини. За допомогою функції getCursorPosition(canvas, event) виконується перевірка значення, яке повертає checkAPoint(a, b, x, y, r) - функція приналежності точки околу вершини графу камер. Визначивши приналежність кліку зображенню камери можно отримати всі дані про камеру та надіслати їх до серверної кінцевої точки @app.route('/camera-about').

Для того, щоб серверною частиною було надано доступ до камери необхідна авторизація адміністратора (рис. 4.8).

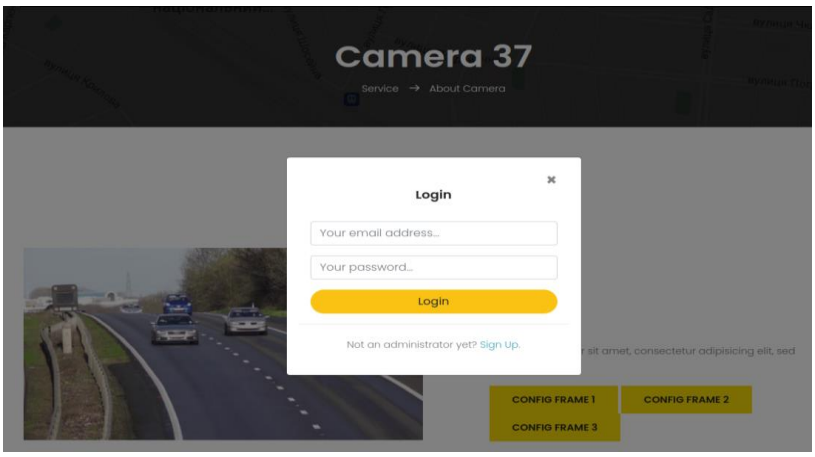

Рисунок 4.8 – Авторизація адміністратора при отриманні доступу до камери

Після того як адміністратор авторизувався, він може побачити швидкість руху, яка визначається алгоритмами OpenCV на серверній частині та виконати відправку файлів кадрів, які продемонструють адміністраторові поточний стан роботи алгоритмів та сконфігурують роботу розпізнавання кадрів.

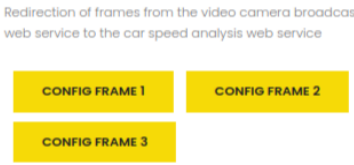

Рис. 4.9 – Інтерфейс відлагодження роботи алгоритмів розпізнавання

При натисканні на кнопку «Config Frame 1», «Config Frame 2», «Config Frame 3» (рис. 4.9) буде виконуватись відправка запитів на перенаправлення кадрів з вебсервісу трансляції відео з камери до вебсервісу аналізу швидкості руху автомобілей.

4.3 Тестування програмного забезпечення

Тестування програм відноситься до тестування будь-якого ПЗ з використанням сценаріїв, інструментів або фреймворків автоматизації тестування для виявлення помилок. Це допомагає створювати та випускати безпомилкові та надійні програмні застосунки, а також виявляти помилки на ранніх стадіях розробки та заощаджувати час розробки. Тестування проводиться у два етапи – тестування Frontend або User Interface та Backend тестування, яке перевіряє поведінку бази даних. Існує два поширені способи протестувати ПЗ: ручне тестування та автоматизоване тестування. [57].

Flask надає спосіб перевірити програму, відкривши тестовий клієнт Werkzeug і обробивши локальні контексти. Використовуємо пакет pytest як базову основу для тестів. Можна встановити його за допомогою pip, наприклад: $ pip install pytest. Почнемо з додавання каталогу тестів до кореневого каталогу програми. Потім створюємо файл Python для зберігання тестів (test_flaskr.py). Відформатуємо ім’я файлу, як test_*.py, воно буде

автоматично знайдено за допомогою pytest. Потім створюємо пристрій pytest під назвою client(), який налаштовує програму для тестування та ініціалізує нову базу даних (рис. 4.10):

```
import os
import tempfile

import pytest

from flaskr import flaskr


@pytest.fixture
def client():
    db_fd, flaskr.app.config['DATABASE'] = tempfile.mkstemp()
    flaskr.app.config['TESTING'] = True

    with flaskr.app.test_client() as client:
        with flaskr.app.app_context():
            flaskr.init_db()
        yield client

    os.close(db_fd)
    os.unlink(flaskr.app.config['DATABASE'])
```

Рисунок 4.10 – Тестування та ініціалізація нової БД

Він викликається кожним окремим тестом, що надає простий інтерфейс, де можна ініціювати тестові запити до програми. Клієнт також відстежуватиме файли cookie. Під час налаштування активується прапорець конфігурації TESTING. Він вимикає перехоплення помилок під час обробки запитів, щоб отримувати найкращі звіти про помилки під час виконання тестових запитів до програми.

Тому у SQLite3 є файломсистеми, засновані на використанні tempfile module для створення temporary database and initialize it. Функція mkstemp() виконується для відображення низького рівня файлу handle і рядку файлу назви, що буде використовуватися в якості database name. Для того, щоб зберегти db_fd, можна використати os.close() функцію для закриття файлу. Щоб видалити базу даних після тесту, пристрій закриває файл і видаляє його з файлової системи. Якщо запустити набір тестів, то побачимо такі результати (рис. 4.11):

```
$ pytest

================ test session starts ================
rootdir: ./flask/examples/flaskr, inifile: setup.cfg
collected 0 items

=========== no tests ran in 0.07 seconds ============
```

Рисунок 4.11 – Результати тестування

Незважаючи на те, що flask не запускав жодних фактичних тестів, дана програма синтаксично дійсна, інакше імпортування завершилось.

Розпочинаємо тестування функціональності програми. Програма показує «Поки тут немає записів», якщо ми отримуємо доступ до кореня програми (/). Для цього додаємо нову тестову функцію до test_flaskr.py. Тестові функції починаються зі словесного тесту, що дозволяє pytest автоматично ідентифікувати функцію як тест для запуску. Використовуючи client.get, можемо надіслати HTTP-запит GET програмі з заданим шляхом. Поверненим значенням буде об’єкт response_class. Тепер можемо використовувати атрибут data для перевірки значення, що повертається (як рядок) із програми. У цьому випадку виведення

тексту «Поки що тут немає записів» є частиною результату. Запускаємо його ще раз і побачимо ще один прохідний тест (рис. 4.12):

```
$ pytest -v

================ test session starts ================
rootdir: ./flask/examples/flaskr, inifile: setup.cfg
collected 1 items

tests/test_flaskr.py::test_empty_db PASSED

============= 1 passed in 0.10 seconds ==============
```

Рисунок 4.12 – Повторний запуск тесту

Більшість функціональності даної програми доступна тільки для адміністративного користувача, тому нам потрібен спосіб входу та виходу нашого тестового клієнта з програми. Для цього запускаємо деякі запити на сторінки входу та виходу з необхідними даними форми (ім’я користувача та пароль). І оскільки сторінки входу та виходу перенаправляються, то для клієнта використовується follow_redirects. Додаємо дві функції у файл test_flaskr.py. Тепер можемо перевірити, чи вхід та вихід працює і чи він не працює з недійсними обліковими даними. Додаємо ці нові функції тестування (рис. 4.13):

```
def test_login_logout(client):
    """Make sure login and logout works."""

    rv = login(client, flaskr.app.config['USERNAME'], flaskr.app.config['PASSWORD'])
    assert b'You were logged in' in rv.data

    rv = logout(client)
    assert b'You were logged out' in rv.data

    rv = login(client, flaskr.app.config['USERNAME'] + 'x', flaskr.app.config['PASSWORD'])
    assert b'Invalid username' in rv.data

    rv = login(client, flaskr.app.config['USERNAME'], flaskr.app.config['PASSWORD'] + 'x')
    assert b'Invalid password' in rv.data
```

Рисунок 4.13 – Додавання нової функції тестування

Далі перевіряємо чи додавання повідомлень працює. Додаємо нову тестову функцію, подібну до попередньої (рис. 4.14):

```
def test_messages(client):
    """Test that messages work."""

    login(client, flaskr.app.config['USERNAME'], flaskr.app.config['PASSWORD'])
    rv = client.post('/add', data=dict(
        title='<Hello>',
        text='<strong>HTML</strong> allowed here'
    ), follow_redirects=True)
    assert b'No entries here so far' not in rv.data
    assert b'&lt;Hello&gt;' in rv.data
    assert b'<strong>HTML</strong> allowed here' in rv.data
```

Рисунок 4.14 – Додавання нової функції для тестування повідомлень

Далі перевіряємо, що HTML дозволено в тексті, але не в заголовку, що є передбачуваною поведінкою. Баг, який тепер має показати три прохідні тести (рис. 4.15):

```
$ pytest -v

================ test session starts ================
rootdir: ./flask/examples/flaskr, inifile: setup.cfg
collected 3 items

tests/test_flaskr.py::test_empty_db PASSED
tests/test_flaskr.py::test_login_logout PASSED
tests/test_flaskr.py::test_messages PASSED

============= 3 passed in 0.23 seconds ==============
```

Рисунок 4.15 – Тестування багів

Клієнтська частина була написана з використанням jquery.js та jquery-ui.js Для отримання доступу до функцій Jest під час початкового завантаження для тестування jquery.js необхідно виконати налаштування: *$.fn.modal = jest.fn()*.

Часто під час написання тестів є певна робота з налаштування, яка має бути виконана перед запуском тестів, і у є певна завершальна робота, яка має бути виконана після виконання тестів. Для цього Jest надає допоміжні функції. Якщо є певна робота, яку потрібно виконати неодноразово для багатьох тестів, можна використовувати beforeEachта afterEach хуки. Наприклад, припустимо, що кілька тестів взаємодіють з базою даних міст. У вас є метод initializeCityCamerasDatabase(), який потрібно викликати перед кожним із цих тестів, і метод clearCityCamerasDatabase(), який потрібно викликати після кожного з цих тестів.

Функції beforeEach і afterEach можуть обробляти асинхронний код так само, як тести можуть обробляти асинхронний код – вони можуть приймати done-параметр або повертати до попередньої умови. Наприклад, якщо initializeCityCamerasDatabase() повернуто до попередньої умови, яка була вирішена під час ініціалізації бази даних, то вона виглядатиме наступним чином: beforeEach(() => { return initializeCityCamerasDatabase(); });

У деяких випадках необхідно виконати налаштування лише один раз, на початку файлу. Коли налаштування є асинхронним, не можна зробити це вбудовано. Jest надає beforeAll хуки afterAll для вирішення даної ситуації. Верхній рівень before* і after* хуки застосовуються до кожного тесту у файлі. Хуки, оголошені всередині describeблоку, застосовуються лише до тестів у цьому describe блоці. Наприклад, для бази даних міста з камерами та бази даних кадрів із зображеннями автівок можна зробити різні налаштування для різних тестів. Верхній рівень beforeEach виконується перед beforeEach внутрішнім describe блоком. Jest виконує всі обробники опису в тестовому файлі перед виконанням будь-яких фактичних тестів. Це ще одна причина для налаштування та демонтажу всередині before* обробників after*, а не всередині describe блоків. Після того, як describe блоки завершені, за замовчуванням Jest запускає всі тести послідовно в тому порядку, в якому вони зустрічалися на етапі збору, чекаючи, поки кожен завершиться і буде приведений у порядок, перш ніж рухатися далі.

Як describe і test блоки and, Jest викликає хуки before* та after*у порядку оголошення. Зверніть увагу, що after* хуки охоплюючої області викликаються першими. Якщо використовувати jasmine2 засіб виконання тестів, він викликає after* хуки в порядку, зворотному до оголошення. Якщо тест зазнає невдачі, однією з перших речей, яку слід перевірити, є те, чи не проходить тест, коли це єдиний тест, який виконується. Щоб запустити лише один тест із Jest, тимчасово змініть цю test команду на test.only. Якщо наявний тест, який часто зазнає невдачі під час його виконання, як частини більшого набору, але не зазнає невдачі, за умов його окремого запуску, то цілком можливо, що щось із іншого тесту заважає цьому. Часто це можна виправити, очистивши деякий спільний стан за допомогою beforeEach. Якщо не має впевненості, чи змінюється якийсь спільний стан, також можна спробувати, beforeEach який реєструє дані.

4.4 Інструкції користувача

Після того, як вебсервіс було протестовано необхідно розробити інструкції для користувачів, які будуть розгортати програмний комплекс на хостингу та для. Адміністраторові потрібно авторизуватись в хмарному IDE та натиснути кнопку «Run» для запуску main.py файлу, щоб почалося очікування отримання запитів для передачі кадрів з камери відбувається через клієнтський вебсервіс (рис. 4.16).

```python
import base64
import json

import requests

api = 'https://mgybarev.pythonanywhere.com/api/frame'
image_file = 'image-forgnd.png' #first frame
# image_file = 'cars.png' #first frame
# image_file = 'image-2.jpg' #first frame
# image_file = 'car-violet-frame-end.png' #second frame
# image_file = 'Screenshot from 2022-12-19 00-09-23.png'

with open(image_file, "rb") as f:
    im_bytes = f.read()
im_b64 = base64.b64encode(im_bytes).decode("utf8")

headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}

payload = json.dumps({"image": im_b64, "other_key": "value"})

response = requests.post(api, data=payload, headers=headers)
try:
    data = response.json()
    print(data)
except requests.exceptions.RequestException:
    print(response.text)
```

Рисунок 4.16 – Вигляд програмного коду вебсервіс для передачі кадрів з вебкамери в хмарному IDE

Виконується аналіз із двох кадрів. Перший кадр – це кадр який буде містити зображення машини, за допомогою якого буде відслідкуватись швидкість руху на ділянці дороги. Саме на другому кадрі ми будемо використовувати зображення машини з першого кадру і шукаючи співпадіння зображення машини за допомогою алгоритму з використанням функцій фреймворку OpenCV (рис. 4.17, рис. 4.18).

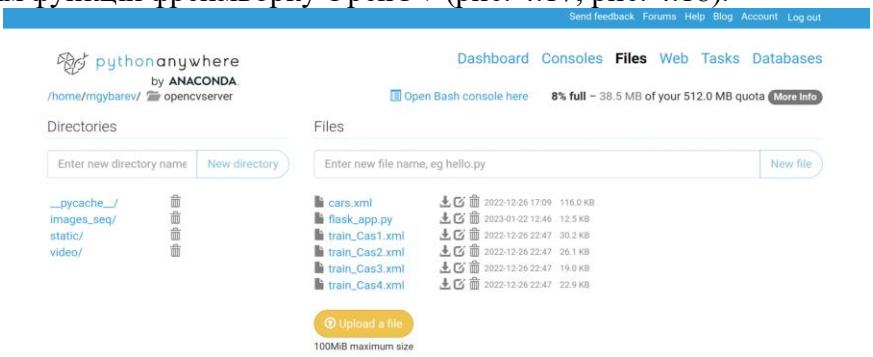

Рисунок 4.17 – Структура директорій серверної частини програмного комплексу

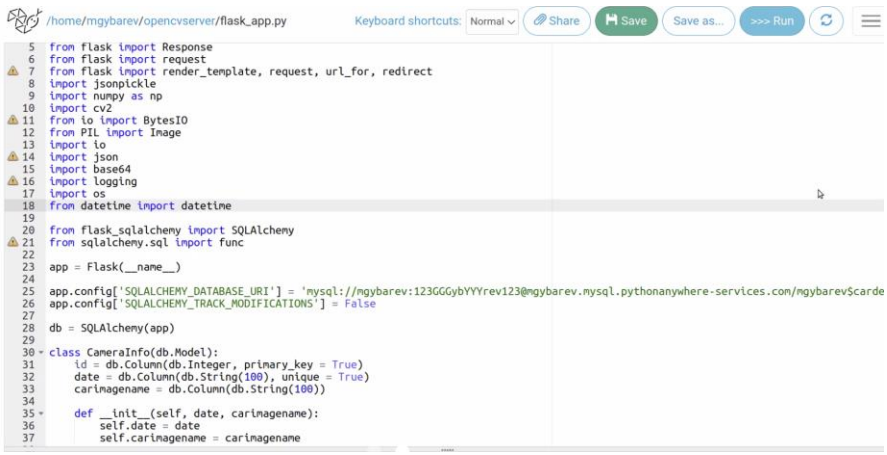

Рисунок 4.18 – Підключення обов'язкових бібліотек для роботи алгоритмів розпізнавання у головному файлі проєкту

Для початку роботи серверної частини потрібно щоб за допомогою функції db.create_all() створилася таблиця «camera_info» в БД з назвою «cardetectdb». Для цього необхідно відкрити MySQL console на хмарному хостингу (рис. 4.19).

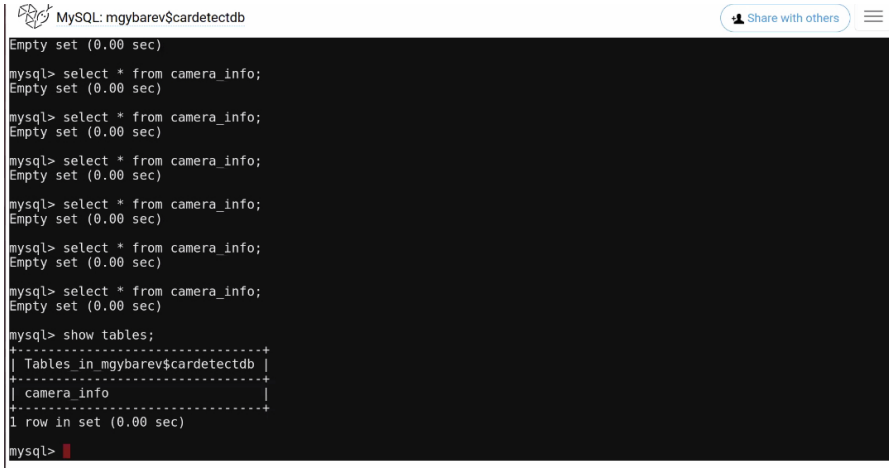

Рисунок 4.19 – Стан таблиці в БД про розпізнавання на початку

Перед початком роботи камер потрібно перевірити щоб в директорії для збереження проміжкових даних процесу розпізнавання не було попередньо збережених системних файлів, дані в яких є неактуальними для майбутнього розпізнавання автівок камерою і можуть призвести до зниження ефективності аналізу швидкості на ділянках доріг (рис. 4.20).

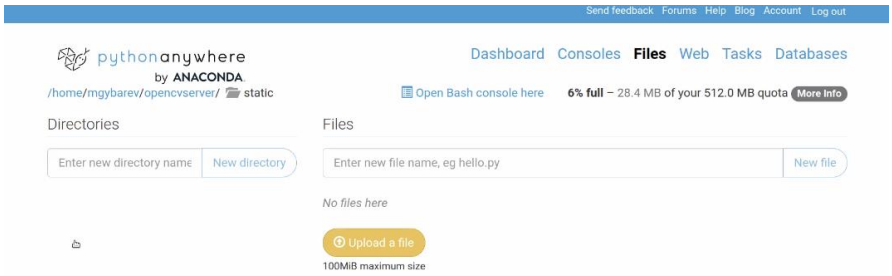

Рисунок 4.20 – Стан директорії для збереження результатів розпізнавання перед початком заміру швидкості на ділянці дороги

Початок та успішне виконання процесу передачі зображень кадрів адміністратор може відслідковувати в повідомленнях в console та в логах хостингу. Перше місце, куди дивитися, якщо щось піде не так – це файли журналів: журнал доступу: mgybarev.pythonanywhere.com.access.log, журнал помилок: mgybarev.pythonanywhere.com.error.log, журнал сервера: mgybarev.pythonanywhere.com.server.log, журнали періодично змінюються, знайти старі журнали можна у директорії: /var/log (рис. 4.21, рис. 4.22).

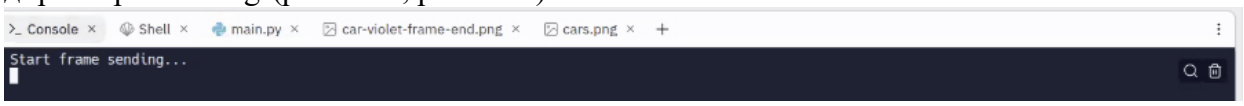

Рисунок 4.21 – Відправлення першого кадру для початку заміру швидкості

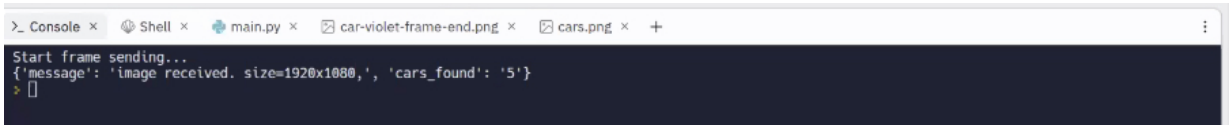

Рисунок 4.22 – Розпізнавання на першому кадрі приблизної кількості об’єктів машин

Після того відповіді про успішне розпізнавання були отримані з головної серверної частини, адміністратор може переглянути стан таблиці «camera_info» в БД. Там буде відображено дату збереження області активної машини, шлях до зображення та статус, чи присутнє зображення активної машини з попереднього кадру в поточному кадрі (рис. 4.23).

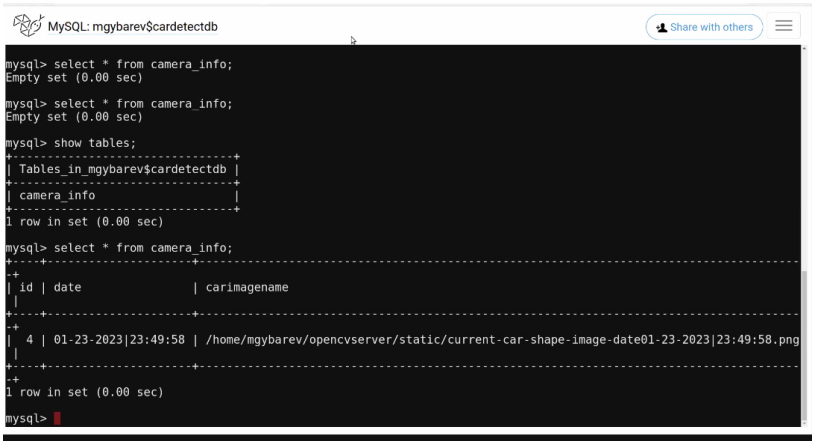

Рисунок 4.23 – Інформація в таблиці в БД про розпізнавання об’єкту машини для початку вимірювання швидкості на ділянці дороги

Під час циклічної відправки наступних кадрів адміністратор може бачити системні повідомлення про результати розпізнавання саме в термінальному інтерфейсі хмарного хостингу. Це є зручним коли адміністратор знаходиться в приміщеннях датацентрів чи в офісах компанії, що забезпечує надання послуг відеоспостереження та аналізу швидкості на ділянках доріг (рис. 4.24, рис. 4.25).

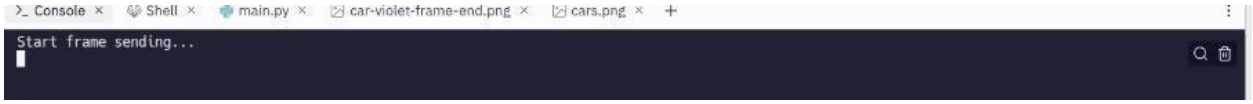

Рисунок 4.24 – Відправлення другого кадру зі зміщенням

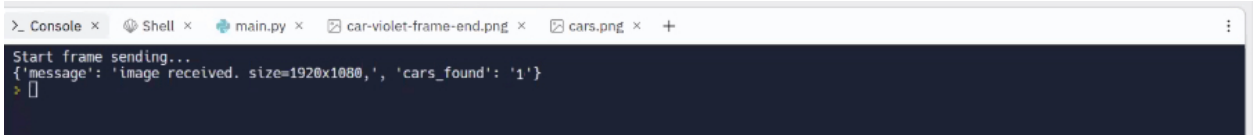

Рисунок 4.25 – Знайдено шукану активну машину з попередього кадру

Коли адміністратор знаходиться в мережі з високою швидкістю інтернету, то зображення результатів розпізнавання можуть переглядатись в системних директоріях хмарного хостингу (рис. 4.26).

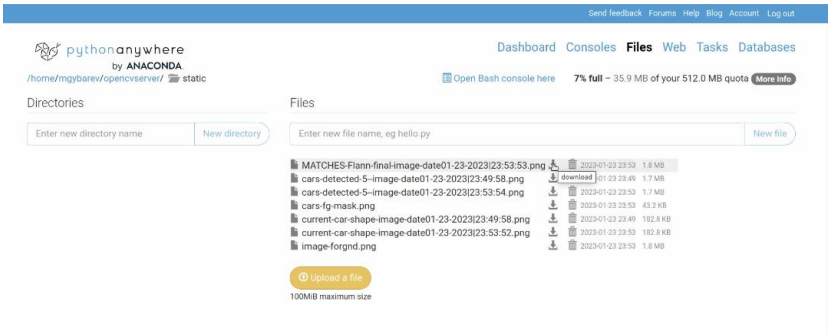

Рисунок 4.26 – Зображення які були згенеровані в процесі розпізнавання машин

Адміністратор може побачити стартовий кадр відносно якого починається пошук активної автівки для наступного аналізу кадрів. Це робиться щоб візуально можна проконтролювати якість об’єктів у кадрі в результаті аналізу етапів процесу розпізнавання в проблемних сутаціях (рис. 4.27).

Рисунок 4.27 – Перший кадр для розпізнавання машини відносно якого буде вимірятись швидкість потоку трафіку на ділянці дороги

Як можна побачити виникла проблемна ситуація у розпізнаванні (рис. 4.28). Одна з областей не містить зображення автівки та розташована на узбіччі дороги. Як зазначено в пункті наукової новизни, що для подолання данної проблеми в була модифікована послідовність етапів шляхом додавання етапу видалення фону (рис. 4.29) навколо машин.

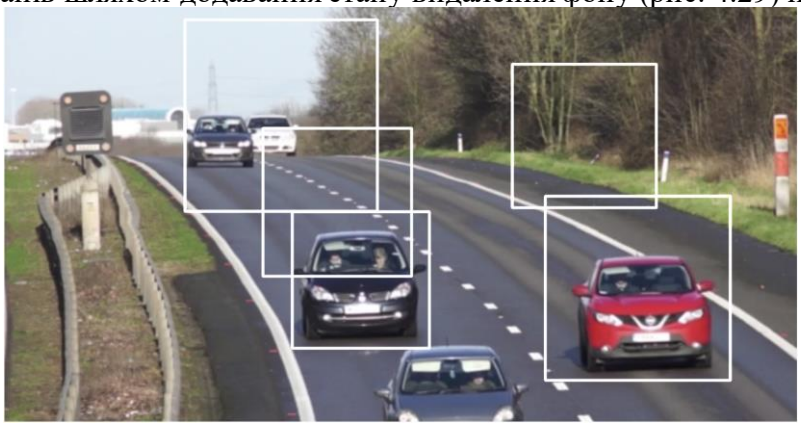

Рисунок 4.28 – Результат виявлення об’єктів машин за допомогою каскадних класифікаторів на основі функцій Хаара

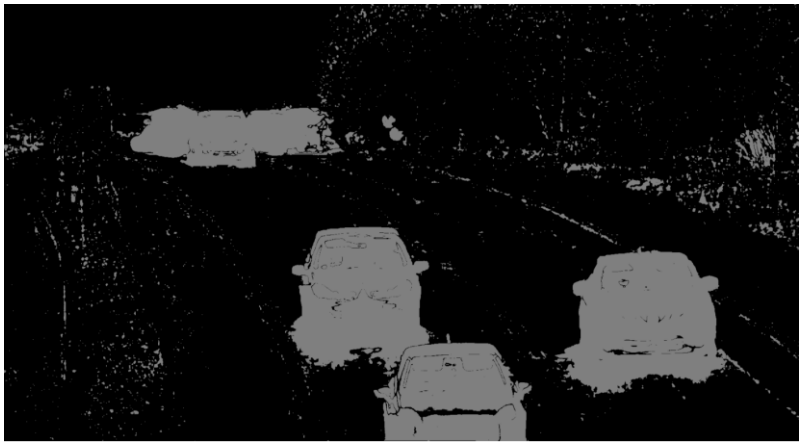

Рисунок 4.29 – Результат фільтрації фону об’єктів машин для оптимізації кількості областей с низьким рівнем розпізнавання

Адміністратор може передивлятись зображення машин, які використовувались у різних кадрах для визначення швидкості руху автівок ділянкою дороги (рис. 4.30).

Рисунок 4.30 – Результат вибору на основі кольорових ознак біколорного фону області об’єкту машини для початку вимірювання швидкості на ділянці дороги

Спочатку адміністратор може пересвідчитись в наявності машина на першому кадрі (рис. 4.31), а потім у наступних кадрах (рис. 4.32).

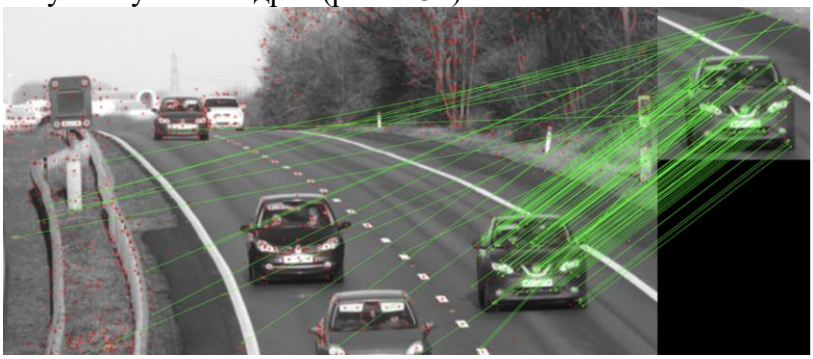

Рисунок 4.31 – Результат визначення зміни положення об’єкта машини відносно якої проводяться виміри швидкості на ділянці дороги в точці А

Адміністратор може візуально пересвідчитись у факті зсуву області з шуканою автівкою, але також потрібно визначати відстань зсуву (рис. 4.32).

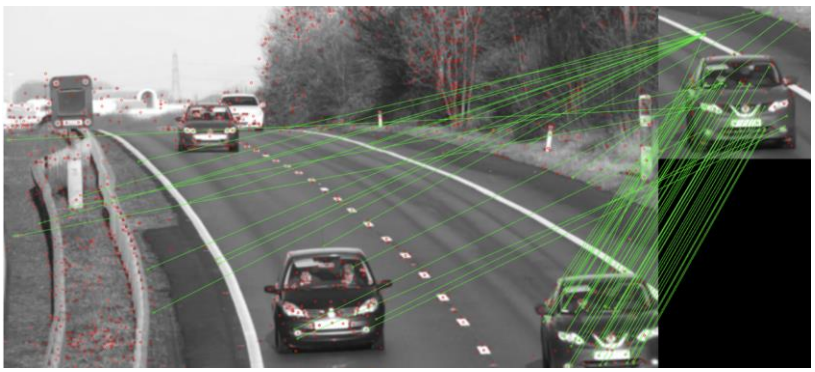

Рисунок 4.32 – Результат визначення зміни положення об’єкта машини відносно якої проводяться виміри швидкості на ділянці дороги в точці Б

Рисунок 4.33 – Файл з хмарного хостингу з результатами побудови траєкторії переміщення автівки

Адміністратор має права доступу до камер, які відображені на мапі (рис. 4.33). Дана функціональність необхідна коли адміністратор буде знаходитись в в місті в умовах ремонту камери чи огляду її технічного стану.

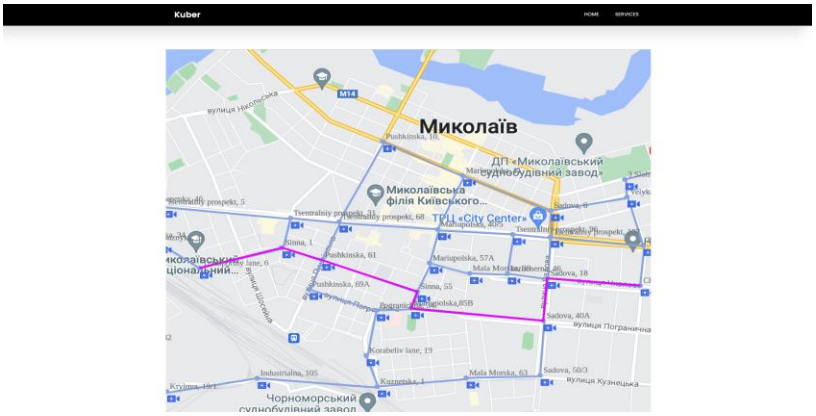

Рисунок 4.34 – Карта перегляду адміністратором технічного стану мережі камер

При відкритті інтерфейсу взаємодії з камерою бачить вікно авторизації і має підтвердити свою особистість, як уповноважена особа. Адміністратору потрібно ввести логін і пароль (рис. 4.35).

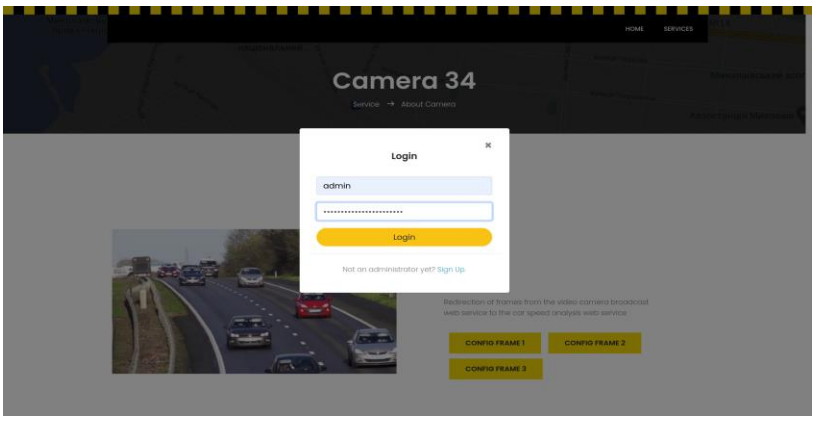

Рисунок 4.35 – Карта перегляду адміністратором технічного стану мережі камер

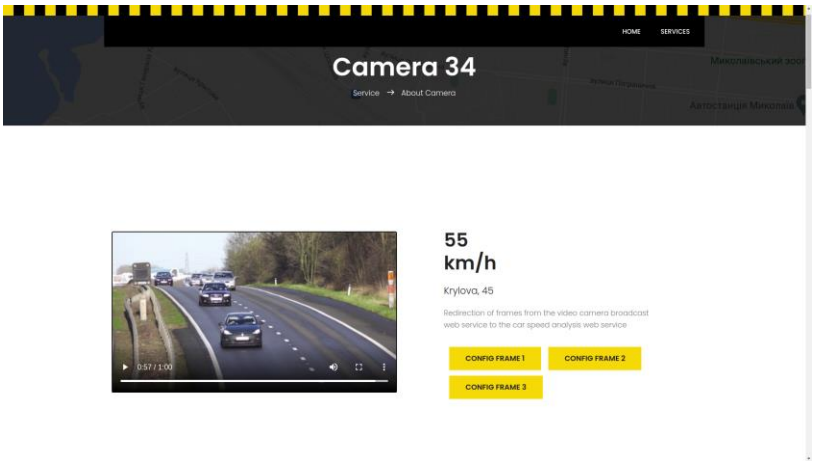

Рисунок 4.36 – Інтерфейс налаштування камери для авторизованого адміністратора

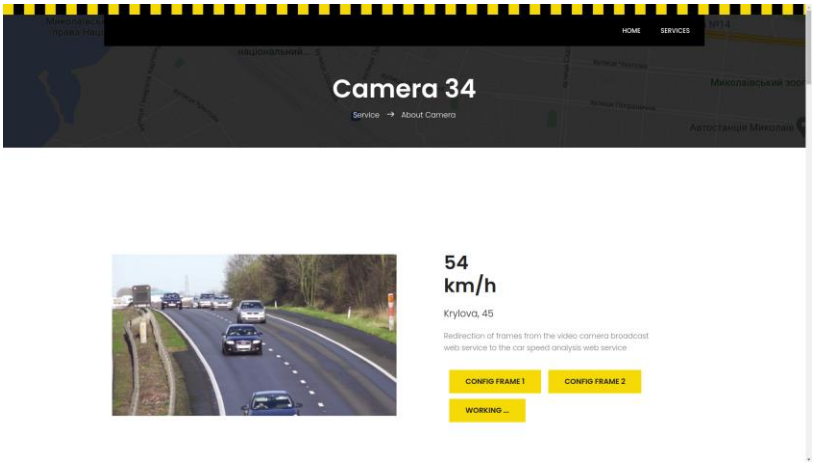

Рисунок 4.37 – Статус «Working» під час конфігурації камери адміністратором

Авторизований адміністратор може виконувати операцію конфігурування камер. На відповідному, обраному адміністратором, етапі конфігурації можуть демонструватися статуси виконання конфігурації. При цьому вміст кнопки з назвою етапу конфігурації буде набувати значення «Working», а при завершенні – повертати початкову назву етапу конфігурування. Системні зображення, які згенерувалися підчас етапів розпізнавання об’єктів авторизованому адміністраторові відразу відображаються у вебінтерфейсі у форматі галереї. На зменшені копії зображень можна натискати, щоб подивитися збільшене зображення. Під час перегляду збільшених зображень можна їх змінювати за допомогою елементів навігації (рис. 4.38, рис. 4.39).

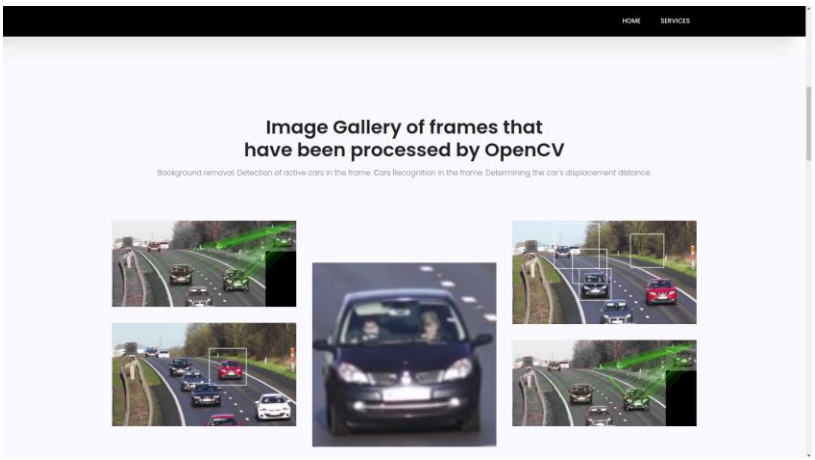

Рисунок 4.38 – Галерея системних зображень в режимі конфігурації камери

Рисунок 4.39 – Навігація галереєю системних зображень в режимі конфігурації камери

Також в розробленому програмному забезпеченні присутня роль користувача, а саме водія автівки. Водії автомобілів – це основна цільова аудиторія, яка буде використовувати мережу дорожніх камер для оптимізації руху маршрутами міста (рис. 4.40).

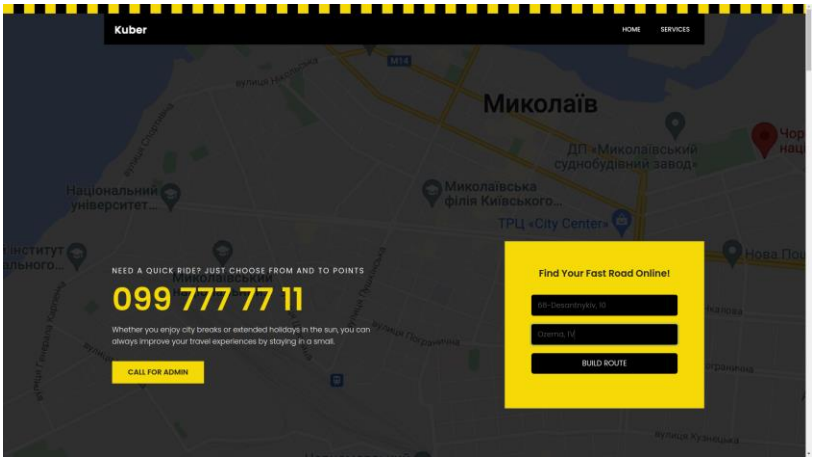

Рисунок 4.40 – Вибір початкової та кінцевої точки маршруту

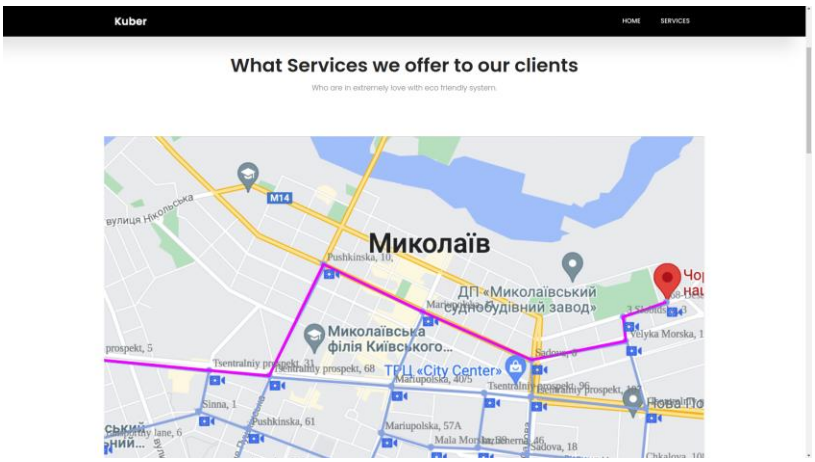

Рисунок 4.41 – Перегляд карти початкової частини маршруту водія

На рис. 4.40-4.42 преставлені частини маршруту. Маршрут було прокладено на основі даних про швидкості руху ділянками доріг міста. Швидкість аналізувалася мережею дорожніх камер, які надсилають кадри зображень, після чого виконуються етапи аналізу швидкості руху автомобілів. Якщо водієві необхідно обрати інший маршрут, то він має повернутись на «Home» вебсторінку та обрати нові початковий і кінцевий пункти маршруту.

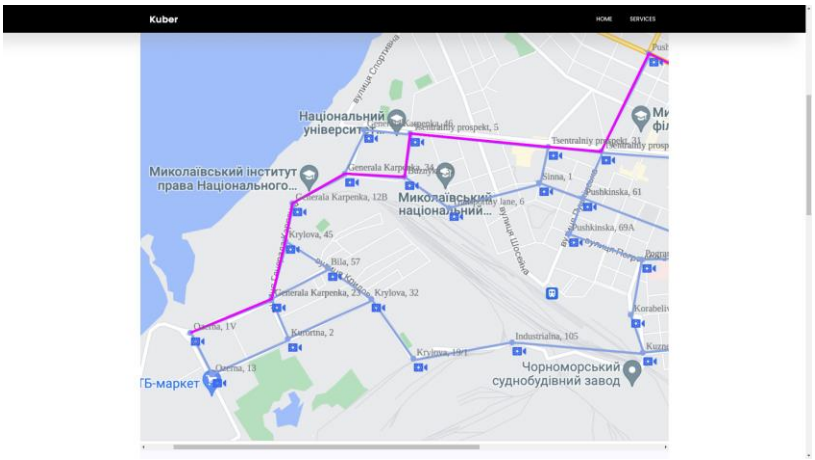

Рисунок 4.42 – Перегляд карти кінцева частини маршруту водія

Водіям в автівках рекомендовано використовувати програмне забезпечення на мобільних платформах. Адаптивний інтерфейс клієнтського вебзастосунку надає зручний формат перегляду карт маршруту у мобільному браузері.

Висновки до розділу 4

Було вирішено, що серверна частина вебсервісу аналізу дорожнього трафіку має бути реалізована за допомогою OpenCV. Для системи виявлення транспортних засобів буде достатньо роботи як із зображеннями, так і з відео, до того ж буде використовуватись OpenCV для виконання всіх операцій обробки зображень. Для для виявлення та класифікації автомобілів було обрано каскадний класифікатор Хаара. У результаті було розраховано дистанцію зсуву центрів активної машини у двох кадрах. Розділивши дистанцію зсуву на інтервал часу між двома кадрами ми отримаємо значення швидкості, яке буде доступне у змінній speed. Системні дані відправляється до клієнтської частини системна інформація щодо статусу розпізнавання у другому кадрі та список зображень

згенерованих в процесі розпізнавання, проте з вже визначеною швидкістю, яка відобразиться адміністраторові.

На початку використання клієнтського вебзастосунку керувальник транспортним засобом повинен обрати початкову та кінцеву точку руху. Камера буде виконувати моніторинг дороги в напрямку попередньої вершини зі встановленою камерою.

Основними інструментами для розробки клієнтського вебзастосунку було обрано JS, HTML, CSS. Проаналізувавши існуючи альтернативи було обрано Twitter Bootstrap та для забезпечення динамічності користувацького інтерфейсу наступні JS бібліотеки: easing.min.js, hoverIntent.js, jquery-ui.js, jquery.ajaxchimp.min.js, jquery.magnific-popup.min.js, jquery.nice-select.min.js, mail-script.js, superfish.min.js. Функціональність, яка пов’язана з бізнес-логікою побудови маршруту з урахуванням заторів на вулицях міста, аналізом швидкості руху автівок та адміністрування камер відеоспостереження реалізовано в файлі main.js.

Після того як адміністратор авторизувався, він може побачити швидкість руху, яка визначається алгоритмами OpenCV на серверній частині та виконати відправку файлів кадрів, які продемонструють адміністраторові поточний стан роботи алгоритмів та сконфігурують роботу розпізнавання кадрів.

ВИСНОВКИ

У ході виконання роботи було проведено аналіз існуючих аналогів навігаційних систем, визначення їх основних переваг та недоліків, розроблено інформаційну модель інформаційно-аналітичної системи, нормалізовано відносини сутностей у системі керування базами даних, розроблено програмні засоби для динамічної мережі відеомоніторингуз метою асистування водієві під час руху, а також проаналізовано отримані результати та вироблено рекомендацій для їх практичного застосування.

Також, крім недостатньої точності існуючі системи навігації змушують водіїв відволікатись від управління транспортним засобом. Більш точні та функціональні системи навігації розробляються за допомогою сучасних технологій отримання та обробки інформації. Тому актуальним є створення нових більш функціональних та точних методів навігації.

Розроблено інформаційну модель інформаційно-аналітичної системи для накопичення даних розподіленої мережі вебкамер, обробки відео з вебкамер та роботи навігації водіїв транспортних засобів. Також було оптимізовано алгоритм процесу зіставлення запиту ключового кадру. Моніторинг завантаженості доріг стане легшим і простішим. Водій зможе планувати маршрут, прорахувати скільки часу потрібно переміщення з точки А в точку В. Розроблення програмного забезпечення для навігації водіїв було здійснено на основі алгоритму Дейкстри мовою програмування Python. Використання данного алгоритму та програмного забезпечення допоможе вийти системам навігації на новий рівень.

Було нормалізовано відносини сутностей у системі керування базами даних. У ході роботи було сформульовано основні вимоги щодо створення програмного забезпечення. Вибір було зроблено у сторону ПЗ на основі аналізу міського трафіку, що матиме трирівневу архітектуру та складатиметься з клієнтської, серверної частин та СКБД (обрано MySQL). Обмін даними у даній систем буде здійснюватись з використанням REST API, а ведення інформаційної бази відбуватиметься через SQLAlchemy. Серверна частина програмної системи буде побудована за допомогою Flask, фронтенд ПЗ буде написано з використанням Javascript-фреймворку JQuery.

Розроблено програмні засоби для динамічної мережі відеомоніторингу. Основні функції адміністратора: авторизація, підключення та налаштування камер, модерація користувачів, редагування та видалення камер. Основні функції клієнта: авторизація, пошук місць за адресою, побудова маршруту. Оскільки вебсервіс, що розробляється є досить комплексним, було розглянуто основні прецеденти та першочергові функціональні можливості. Для програмного комплексу системи навігації водіїв це – наявність гнучкого налаштування побудови маршрутів та можливість контролю процесу розпізнавання об’єктів машин на ділянках доріг. Було представлено діаграми розгортання вебсервісу та діаграми послідовності. Дана система побудована на архітектурі, де кожен сервіс має RESTful API.

У результаті проведеного аналізу особливостей різних алгоритмів, було розроблено алгоритм для передачі даних та сповіщення про завантаженість. За допомогою реалізованої технології розпізнавання завантаженості автомобільного трафіку дозволяє навігаційній системі збирати та аналізувати дані, що, у свою чергу, поліпшує повсякденне керування дорожнім рухом та адаптує до довгострокових потреб у транспорті. Було вирішено, що серверна частина вебсервісу аналізу дорожнього трафіку має бути реалізована за допомогою OpenCV. Для системи виявлення транспортних засобів буде достатньо роботи як із зображеннями, так і з відео, до того ж буде використовуватись OpenCV для виконання всіх операцій обробки зображень. Для для виявлення та класифікації автомобілів було обрано каскадний класифікатор Хаара

Результатом є розрахунок дистанції зсуву центрів активної машини у двох кадрах. Розділивши дистанцію зсуву на інтервал часу між двома кадрами ми отримаємо значення швидкості, яке буде доступне у змінній speed. Системні дані відправляється до клієнтської частини системна інформація щодо статусу розпізнавання у другому кадрі та список зображень згенерованих в процесі розпізнавання, проте з вже визначеною швидкістю, яка відобразиться адміністраторові.

Також було оптимізовано алгоритм розпізнавання транспортних засобів на ділянках мережі доріг міста за рахунок модифікована послідовність етапів шляхом додавання етапу видалення фону з використанням класу NPArrayMatcherAdapter.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Давиденко Є. О., Швед А. В., Кандиба І. О. Методичні рекомендації до виконання кваліфікаційних робіт студентами спеціальності 121 «Інженерія програмного забезпечення» ступеня вищої освіти «Магістр»: схвалено Вченою радою факультету комп'ютерних наук ЧНУ ім. Петра Могили від 29.08.2022 р. (дата звернення 03.01.2023).

2. Definitions.net : dictionary. URL: https://www.definitions.net/definition/navigation+system (Last accessed: 22.10.2022).

3. Alan Rankin. What is an Automotive Navigation System? WikiMotors : website. URL: https://www.wikimotors.org/what-is-an-automotive-navigation-system.htm (Last accessed: 22.10.2022).

4. Navigation Systems. CVEL : website. URL: https://cecas.clemson.edu/cvel/auto/systems/navigation.html (Last accessed: 23.10.2022).

5. What is Galileo? European Space Agency : website. URL: https://www.esa.int/Applications/Navigation/Galileo/What_is_Galileo (Last accessed: 11.11.2022).

6. China's Beidou GPS-substitute opens to public in Asia. BBC: website. URL: https://www.bbc.com/news/technology-20852150 (Last accessed: 11.11.2022).

7. Bart Hendrickx. The secret payloads of Glonass navigation satellites. The space review : website. URL: https://www.thespacereview.com/article/4502/1 (Last accessed: 19.12.2022).

8. Caleb Henry. Japan mulls seven-satellite QZSS system as a GPS backup. SpaceNews : website. URL: https://spacenews.com/japan-mulls-seven-satellite-qzss-system-as-a-gps-backup/ (Last accessed: 13.11.2022).

9. Navic: How is India's very own navigation service different from us-owned GPS? Firstpost : website. URL: https://www.firstpost.com/tech/news-analysis/navic-how-is-indias-very-own-navigation-service-different-from-us-owned-gps-11342771.html (Last accessed: 15.11.2022).

10. В Україні розпочала свою роботу мережа референцних станцій «System.NET». System Solutions : вебсайт. URL: https://web.archive.org/web/20170316113025/http://systemnet.com.ua/v-ukraïni-rozpochala-svoyu-robotu-merezha-referencnix-stancij-system-net/ (дата звернення: 12.11.2022).

11. На ракеті Маска. Україна – знову в космічній грі. Шість важливих фактів про український супутник Січ-2-30. Ліга: вебсайт. URL: https://biz.liga.net/ua/all/it/article/ukraina-snova-v-kosmose-shest-vajnyh-faktov-ob-ukrainskom-sputnike-sich-2-30 (дата звернення: 13.11.2022).

12. Методи корекції геоданих. Leica Geosystems: вебсайт. URL: https://ngc.com.ua/info/correction.htm (дата звернення: 14.11.2022).

13. Mir M. Bohlooli. Standalone vs. Integrated Car Navigation Systems: What's the Best Option? MUO : website. URL: https://www.makeuseof.com/standalone-vs-integrated-car-navigation-systems/ (Last accessed: 19.11.2022)

14. Best Practices for Implementing Advanced Automotive Navigation Systems. Infopulse : website. URL: https://www.infopulse.com/blog/best-practices-for-implementing-advanced-navigation-solutions (Last accessed: 18.11.2022).

15. Browse encyclopedia : dictionary. URL: https://www.pcmag.com/encyclopedia/term/navigation-system (Last accessed: 12.11.2022).

16. Modern Car Navigation Systems and Their Features. Infopulse : website. URL: https://www.infopulse.com/blog/modern-car-navigation-systems-and-their-features (Last accessed: 18.11.2022).

17. 10 GPS Apps For Navigation [Android and iOS]. GISGeography : website. URL: https://gisgeography.com/gps-apps-navigation/ (Last accessed: 19.11.2022).

18. Jackie Dove. Waze vs. Google Maps: Which one is right for you? Digital Trends : website. URL: https://www.digitaltrends.com/mobile/waze-vs-google-maps/ (Last accessed: 23.11.2022).

19. Giacomo Rotella. 17 Best GPS Navigation Apps for iOS and Android for 2023. Badger Maps : website. URL: https://www.badgermapping.com/blog/best-navigation-apps/#gmaps (Last accessed: 25.11.2022).

20. Rakesh Patel.15 Best Navigation Apps for Android & iOS Devices. Upper Route Planner : website. URL: https://www.upperinc.com/blog/best-navigation-apps-android-ios/ (Last accessed: 28.11.2022).

21. High-performance distributed storage system for large-scale high-definition video data. Ruan Jian Xue Bao SD Cao, Y Hua, D Feng, YY Sun, PF Zuo - Journal of Software, 2017

22. J. Calic and E. Izquierdo, "Efficient key-frame extraction and video analysis," in Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2002, pp. 28–33, USA, April 2002. View at: Publisher Site | Google Scholar

23. J. He, Study on key technique in video surveillance storage system based on IP-SAN, Shanghai, Shanghai Jiaotong University, 2011, in Chinese.

24. H. E. Xiao-Feng, Analysis and Application of Network Video Recorder (NVR) Storage Technology, China Science & Technology Information, 2011.

25. G. Liu and J. Zhao, "Key frame extraction from MPEG video stream," in Proceedings of the 3rd International Symposium on Information Processing, ISIP 2010, pp. 423–427, China, November 2010. View at: Publisher Site | Google Scholar

26. Y. Yang, F. Dadgostar, C. Sanderson, and B. C. Lovell, "Summarisation of surveillance videos by key-frame selection," in Proceedings of the 2011 5th ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC 2011, Belgium, August 2011. View at: Publisher Site | Google Scholar

27. Кутковецький В. Я. Розпізнавання образів : навчальний посібник / В. Я. Кутковецький. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2017. – 420 с. URL: https://dspace.chmnu.edu.ua/jspui/bitstream/123456789/60/1/Кутковецький%20В.%20Я.%20Розпізнавання%20образів.pdf (дата звернення 10.10.2022).

28. Путятін Є. П., Гороховатській В.О., Матат О.О. Методи та алгоритми комп'ютерного зору: Навч. посібник. Х: СМІТ, 2006. 236 с. (дата звернення 10.10.2022).

29. Вовк С.М., Гнатушенко В.В., Бондаренко М.В. Методи обробки зображень та комп'ютерний зір : навч. посіб. / С.М. Вовк, В.В. Гнатушенко, М.В. Бондаренко. – Д. : ЛІРА, 2016. – 148 с. (дата звернення 10.10.2022).

30. Simon J.D. Prince. Computer Vision: Models, Learning, and Inference. – Cambridge: Cambridge University Press. – 567 p. – 2017. URL: http://www.cambridge.org/ (Last accessed: 03.11.2022).

31. Прохоренок Н. OpenCV и Java. Обработка изображений и компьютерное зрение. – СПб.: БХВ, 2018. – 320 с. URL: https://www.pdfdrive.com/opencv-и-java-Обработка-изображений-и-компьютерное-зрение-e158439226.html (дата звернення 04.11.2022).

32. Богуто Д. Г., Волинець В. І., Ніколюк П. К., Ніколюк П. П. Автоматизована система керування рухом транспортних засобів в межах міста// Вісник Харківського університету, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – No5. – 2017. – С. 3-9. URL: http://dspace.wunu.edu.ua/bitstream/316497/25072/2/VKNU_2017_35.pdf (дата звернення 05.11.2022).

33. Пуйда В.Я. Мультипроцесорна система технічного зору // Кіберфізичні системи: проблеми створення та напрями розвитку // Вісник Національного університету "Львівська

політехніка" "Комп'ютерні системи та мережі". – 2017. – No 710. – С. 85–91 (дата звернення 08.11.2022).

34. Li C., Xu C., Cui Z., Wang D., Jie Z., Zhang T., and Yang J. Learning object-wise semantic representation for detection in remote sensing imagery. In Proc. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 20-27. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9888013 (Last accessed: 11.11.2022).

35. Kanhere, Birchfield, Sarasua, Whitney, "Real-Time Detection and Tracking of Vehicle Base Fronts for Measuring Traffic Counts and Speeds on Highways" M. : Opera RT, 2012. – 340 с. URL: http://cecas.clemson.edu/~stb/publications/vehicle_tracking_trb2007.pdf (Last accessed: 08.11.2022).

36. Archer, J. 2004, "Methods for the Assessment and Prediction of Traffic Safety at Urban Intersections and their Application in Micro-simulation Modelling", PhD Thesis, Royal Institute of Technology. URL: https://www.researchgate.net/publication/281642251_Application_of_traffic_microsimulation_for_evaluating_safety_performance_of_urban_signalized_intersections (Last accessed: 07.11.2022).

37. Жовтов С. Ю. та ін. Обробка та аналіз зображень у завданнях машинного зору. - М.: Фізматкнига, 2010. – С. 75-89 (дата звернення 09.11.2022).

38. Писарчук О. О. Технологія автоматизованого управління транспортними потоками в мультимодальній транспортній мережі // О.О. Писарчук, Т.І. Конрад. – Наукоємні технології № 4(48), 2020, С.443-450. URL: https://comsys.kpi.ua/pisarchuk-oleksiy-oleksandrovich (дата звернення 28.10.2022).

39. He K., Zhang X., Ren S., and Sun J.. Deep residual learning for image recognition. In Proc. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. pp. 770–778. DOI: 10.1109/CVPR.2016.90 (Last accessed: 10.11.2022).

40. Real-time video analysis for surveillance and monitoring. *Neurosys*: website.URL:https://neurosys.com/case-study/real-time-video-analysis-for-surveillance-and-monitoring/ (Last accessed: 11.11.2022).

41. Redmon J., Divvala S., Girshick R., and Farhadi A., "You only look once: Unified, real-time object detection," in Proc. *IEEE Conf. Comput. Vis. Pattern Recognit.* 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (Last accessed: 12.11.2022).

42. Rajalingappaa Shanmugam. Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras. - Paperback – January 23, 2018. – 305 с. URL: https://journals.com.ua/books/programming/12394-deep-learning-for-computer-vision-expert-techniques-to-train-advanced-neural-networks-using.html (Last accessed: 12.11.2022).

43. Sergiyenko O. Yu., Tyrsa V.V. 3D optical machine vision sensors with intelligent data management for robotic swarm navigation improvement, *IEEE Sensors Journal.* 2021. Vol. 21 (10), pp. 11262-11274. DOI: 10.1109/JSEN.2020.3007856 (Last accessed: 13.11.2022).

44. Python – the language of today and tomorrow. Open Education and Development Group: website. URL: https://pythoninstitute.org/about-python (Last accessed: 15.01.2023).

45. Python – Overview. Tutorials Point: website. URL: https://www.tutorialspoint.com/python/python_overview.htm (Last accessed: 16.01.2023).

46. Abdelhadi Dyouri. How To Make a Web Application Using Flask in Python 3. DigitalOcean : website. URL: https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3 (Last accessed: 17.01.2023).

47. Mark Drake. What is MySQL? DigitalOcean : website. URL: https://www.digitalocean.com/community/tutorials/what-is-mysql (Last accessed: 17.01.2023).

48. Abdelhadi Dyouri. How to Use Flask-SQLAlchemy to Interact with Databases in a Flask Application. DigitalOcean : website. URL: https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application (Last accessed: 18.01.2023).

49. HTML-CSS-JS. The Client-Side Of The Web: website. URL: https://html-css-js.com/#about (Last accessed: 18.01.2023).

50. Unified Modeling Language (UML). GeeksforGeeks: website. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/ (Last accessed: 19.01.2023).

51. Unified Modeling Language (UML) | Class Diagrams. GeeksforGeeks: website. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ (Last accessed: 20.01.2023).

52. Unified Modeling Language (UML) | Activity Diagrams. GeeksforGeeks: website. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/ (Last accessed: 20.01.2023).

53. Aditya Mittal. Haar Cascades. Medium: website. URL: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d (Last accessed: 22.01.2023).

54. Dijkstra's shortest path algorithm. BogoToBogo: website. URL: https://www.bogotobogo.com/python/python_Dijkstras_Shortest_Path_Algorithm.php (Last accessed: 19.01.2023).

55. Jash Unadkat. How to learn Software Application Testing. BrowserStack: website. URL: https://www.browserstack.com/guide/learn-software-application-testing/ (Last accessed: 20.01.2023).

56. Testing Flask Applications. Pallets: website. URL: https://flask.palletsprojects.com/en/1.1.x/testing/ (Last accessed: 22.01.2023).

57. Waweru Mwaura. Testing Flask framework with Pytest. CircleCI: website. URL: https://circleci.com/blog/testing-flask-framework-with-pytest/ (Last accessed: 25.01.2023).

ДОДАТОК А

Лістинг програмного коду алгоритму Дейкстри

```
def dijkstra(aGraph, start, target):
        print ("'Dijkstra's shortest path"')
        # Set the distance for the start node to zero
        start.set_distance(0)
        # Put tuple pair into the priority queue
        unvisited_queue = [(v.get_distance(),v) for v in aGraph]
        heapq.heapify(unvisited_queue)
        while len(unvisited_queue):
            # Pops a vertex with the smallest distance
            uv = heapq.heappop(unvisited_queue)
            current = uv[1]
            current.set_visited()
            #for next in v.adjacent:
            for next in current.adjacent:
                # if visited, skip
                if next.visited:
                    continue
                new_dist = current.get_distance() + current.get_weight(next)
                if new_dist < next.get_distance():
                    next.set_distance(new_dist)
                    next.set_previous(current)
                    print ('updated : current = %s next = %s new_dist = %s' \
                            %(current.get_id(), next.get_id(), next.get_distance()))
                else:
                    print ('not updated : current = %s next = %s new_dist = %s' \
                            %(current.get_id(), next.get_id(), next.get_distance()))

            # Rebuild heap
            # 1. Pop every item
            while len(unvisited_queue):
                heapq.heappop(unvisited_queue)
            # 2. Put all vertices not visited into the queue
            unvisited_queue = [(v.get_distance(),v) for v in aGraph if not v.visited]
            heapq.heapify(unvisited_queue)
```

ДОДАТОК Б

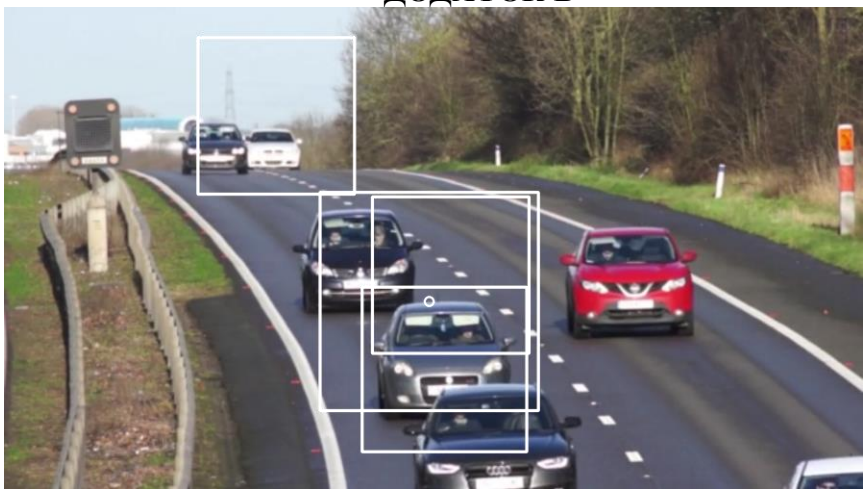

Рисунок 1. –Зображення області автомобіля з центром прямокутника

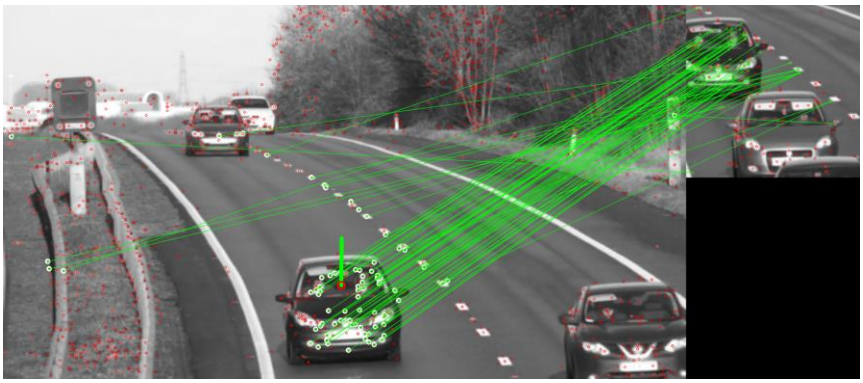

Рисунок 2. – Зсув області автомобіля з центром прямокутника з рис. 1

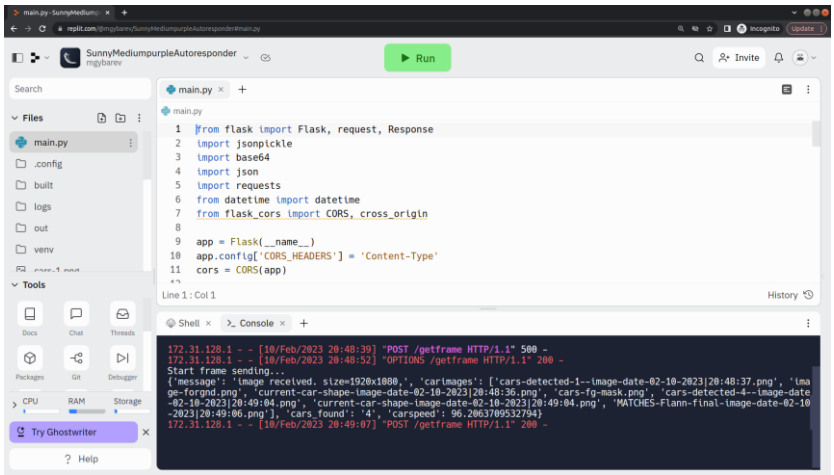

Рисунок 3. – Евристичний розрахунок швидкості автомобілю на основі дистанції зсуву з рис. 2

ДОДАТОК В
Лістинг коду серверної частини аналізу швидкості руху автомобілів дорожньою ділянкою

```
# A very simple Flask Hello World app for you to get started with...

from flask import Flask
from flask import Response
from flask import request
from flask import render_template, request, url_for, redirect
import random
import jsonpickle
import numpy as np
import cv2
from io import BytesIO
from PIL import Image
import io
import json
import base64
import logging
import os
from datetime import datetime
import math
from matplotlib import pyplot as plt
from queue import PriorityQueue
import sys
import heapq
from flask_cors import CORS, cross_origin
import base64
import json
import requests
import urllib.request
from os import walk
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.sql import func
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://mgybarev:123GGGybYYYrev123@mgybarev.mysql.pythonanywhere-
services.com/mgybarev$cardetectdb'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['CORS_HEADERS'] = 'Content-Type'
cors = CORS(app)
db = SQLAlchemy(app)
STATIC_FOLDER = "static/rec-images/"
STATIC_DATA_FOLDER = "static/images_seq/"

class CameraInfo(db.Model):
  id = db.Column(db.Integer, primary_key = True)
```

```
        date = db.Column(db.String(100), unique = True)
        cur_car_img = db.Column(db.String(500))
        cur_car_coords = db.Column(db.String(100))
        found_in_next_frame =  db.Column(db.Integer)
        def __init__(self, date, carimagename, coordinates, is_found):
            self.date = date
            self.cur_car_img = carimagename
            self.cur_car_coords = coordinates
            self.found_in_next_frame = is_found

    db.create_all()
    print("Database created.")
    @app.route('/')
    def index_template():
        return render_template("/taxi-master/index.html")
    @cross_origin()
    @app.route('/camera-redirect', methods=['GET', 'POST'])
    def camera_redirect():
        camera_data = request.get_json()
        name = camera_data['name']
        address = camera_data['addr']
        camPosition = camera_data['pos']

        return render_template("/taxi-master/camera-about.html", name=name,
address=address, camPosition=camPosition)

    @cross_origin()
    @app.route('/camera-about')
    def camera_about():
        name = request.args.get('name')
        address = request.args.get('addr')
        camPosition = request.args.get('pos')
        return render_template("/taxi-master/camera-about.html", name=name,
address=address, camPosition=camPosition)

    @cross_origin()
    @app.route('/update-img-list')
    def update_img_list():
        img_list = get_img_list()
        response = {
            'carimages': img_list
        }
        response_pickled = jsonpickle.encode(response)
        return Response(response=response_pickled, status=200,
mimetype="application/json", headers={'Access-Control-Allow-Origin':'*'})

    @app.route('/service', methods=['GET', 'POST'])
```

```
def map_service():
    if request.method == 'POST':
        placefrom = request.form.get('placefrom')
        placeto = request.form.get('palceto')
        print("PLACES")
        print(f'FROM {placefrom} TO {placeto}')
        g = Graph()
        for vertex in range(0, 43):
            print(f'Added vertex -> {str(vertex)}')
            g.add_vertex(str(vertex))
        g.add_edge('0', '1', get_speed_by_camera())
        g.add_edge('1', '2', get_speed_by_camera())
        g.add_edge('2', '7', get_speed_by_camera())
        g.add_edge('2', '3', get_speed_by_camera())
        g.add_edge('3', '6', get_speed_by_camera())
        g.add_edge('3', '4', get_speed_by_camera())
        g.add_edge('4', '5', get_speed_by_camera())
        g.add_edge('5', '6', get_speed_by_camera())
        g.add_edge('5', '13', get_speed_by_camera())
        g.add_edge('5', '18', get_speed_by_camera())
        g.add_edge('6', '7', get_speed_by_camera())
        g.add_edge('6', '12', get_speed_by_camera())
        g.add_edge('7', '8', get_speed_by_camera())
        g.add_edge('8', '9', get_speed_by_camera())
        g.add_edge('8', '11', get_speed_by_camera())
        g.add_edge('9', '10', get_speed_by_camera())
        g.add_edge('10', '11', get_speed_by_camera())
        g.add_edge('10', '27', get_speed_by_camera())
        g.add_edge('10', '25', get_speed_by_camera())
        g.add_edge('11', '12', get_speed_by_camera())
        g.add_edge('11', '15', get_speed_by_camera())
        g.add_edge('12', '13', get_speed_by_camera())
        g.add_edge('13', '14', get_speed_by_camera())
        g.add_edge('14', '15', get_speed_by_camera())
        g.add_edge('15', '16', get_speed_by_camera())
        g.add_edge('16', '17', get_speed_by_camera())
        g.add_edge('16', '25', get_speed_by_camera())
        g.add_edge('17', '18', get_speed_by_camera())
        g.add_edge('17', '23', get_speed_by_camera())
        g.add_edge('18', '19', get_speed_by_camera())
        g.add_edge('19', '20', get_speed_by_camera())
        g.add_edge('20', '21', get_speed_by_camera())
        g.add_edge('21', '22', get_speed_by_camera())
        g.add_edge('21', '38', get_speed_by_camera())
        g.add_edge('22', '23', get_speed_by_camera())
        g.add_edge('23', '24', get_speed_by_camera())
        g.add_edge('24', '25', get_speed_by_camera())
```

```
            g.add_edge('25', '26', get_speed_by_camera())
            g.add_edge('26', '27', get_speed_by_camera())
            g.add_edge('26', '28', get_speed_by_camera())
            g.add_edge('27', '29', get_speed_by_camera())
            g.add_edge('28', '31', get_speed_by_camera())
            g.add_edge('29', '31', get_speed_by_camera())
            g.add_edge('29', '30', get_speed_by_camera())
            g.add_edge('30', '32', get_speed_by_camera())
            g.add_edge('31', '32', get_speed_by_camera())
            g.add_edge('32', '33', get_speed_by_camera())
            g.add_edge('33', '34', get_speed_by_camera())
            g.add_edge('34', '35', get_speed_by_camera())
            g.add_edge('34', '40', get_speed_by_camera())
            g.add_edge('35', '40', get_speed_by_camera())
            g.add_edge('35', '36', get_speed_by_camera())
            g.add_edge('36', '39', get_speed_by_camera())
            g.add_edge('36', '37', get_speed_by_camera())
            g.add_edge('37', '38', get_speed_by_camera())
            g.add_edge('39', '40', get_speed_by_camera())
            g.add_edge('39', '42', get_speed_by_camera())
            g.add_edge('40', '41', get_speed_by_camera())
            g.add_edge('41', '42', get_speed_by_camera())
            print ('Graph data:')
            for v in g:
                for w in v.get_connections():
                    vid = v.get_id()
                    wid = w.get_id()
                    print ('( %s , %s, %3d)'  % ( vid, wid, v.get_weight(w)))
            dijkstra(g, g.get_vertex(placefrom), g.get_vertex(placeto))
            target = g.get_vertex(placeto)
            path = [target.get_id()]
            shortest(target, path)
            print ('The shortest path : %s' % (path[::-1]))
            route = '%s' % (path[::-1])
        return render_template("/taxi-master/service.html", placefrom=placefrom,
placeto=placeto, route=str(route))

    @app.route('/add-camera')
    def add_camera_stream():
        return "Add ip, url to stream webcam or upload video"

    @app.route('/foreground')
    def get_foreground():
        backSub = cv2.createBackgroundSubtractorMOG2()
        project_dir = os.path.dirname(os.path.abspath(__file__))
        mode = "img"
        if (mode == "vid"):
```

```
            path_to_images_seq = os.path.join(project_dir, "video", "Cars.mp4")
            capture =
cv2.VideoCapture(cv2.samples.findFileOrKeep(path_to_images_seq))
            if not capture.isOpened():
                print('Unable to open images sequence')
                exit(0)
            steps = 0
            while steps < 10:
                ret, frame = capture.read()
                if frame is None:
                    break
                fgMask = backSub.apply(frame)
                saveArrayToImage(fgMask, "cars-fg-mask")
                steps = steps + 1
        if (mode == "img"):
            #https://www.authentise.com/post/segment-background-using-computer-
vision
            path_to_images_bg = os.path.join(project_dir, "images_seq",
"Background.png")
            path_to_images_seq_1 = os.path.join(project_dir, "images_seq","image-
forgnd.png")
            fgMask = backSub.apply(cv2.imread(path_to_images_bg))
            fgMask = backSub.apply(cv2.imread(path_to_images_seq_1))
            saveArrayToImage(fgMask, "cars-fg-mask", True)
        return "Foreground extracted"

    def get_frame_foreground(image_arr):
        backSub = cv2.createBackgroundSubtractorMOG2()
        project_dir = os.path.dirname(os.path.abspath(__file__))
        mode = "img"
        if (mode == "vid"):
            path_to_images_seq = os.path.join(project_dir, "video", "Cars.mp4")
            capture =
cv2.VideoCapture(cv2.samples.findFileOrKeep(path_to_images_seq))
            if not capture.isOpened():
                print('Unable to open images sequence')
                exit(0)
            steps = 0
            while steps < 10:
                ret, frame = capture.read()
                if frame is None:
                    break
                fgMask = backSub.apply(frame)
                saveArrayToImage(fgMask, "cars-fg-mask")
                steps = steps + 1
        if (mode == "img"):
```

```
            path_to_images_bg = os.path.join(project_dir, "images_seq", "Background.png")
            saveArrayToImage(image_arr,'image-forgnd', True)
            path_to_images_seq_1 = os.path.join(project_dir, STATIC_FOLDER, "image-forgnd.png")
            fgMask = backSub.apply(cv2.imread(path_to_images_bg))
            fgMask = backSub.apply(cv2.imread(path_to_images_seq_1))
            saveArrayToImage(fgMask, "cars-fg-mask", True)
        return "Foreground extracted"

    def saveArrayToImage(npArray, label, nodate = False):
        img = Image.fromarray(npArray)
        project_dir = os.path.dirname(os.path.abspath(__file__))
        now = datetime.now() # current date and time
        date_time = now.strftime("%m-%d-%Y|%H:%M:%S")
        if nodate == False:
            file_name = label + "-image-date-"+ date_time +".png"
        else:
            file_name = label + ".png"

        static_image_file = os.path.join(project_dir, STATIC_FOLDER,  file_name)
        img.save(static_image_file)
        return static_image_file

    @app.route('/api/frame', methods=['POST'])
    def get_frame():
        if not request.json or 'image' not in request.json:
            abort(400)
        im_b64 = request.json['image']
        img_bytes = base64.b64decode(im_b64.encode('utf-8'))
        # convert bytes data to PIL Image object
        img = Image.open(io.BytesIO(img_bytes))
        # PIL image object to numpy array
        img_arr = np.asarray(img)
        get_frame_foreground(img_arr)
        project_dir = os.path.dirname(os.path.abspath(__file__))
        file_name= 'cars-fg-mask.png'
        static_image_file = os.path.join(project_dir, STATIC_FOLDER,  file_name)
        foreground_image = cv2.imread(static_image_file) # static file_name for foreground image
        original_image = np.asarray(img) #for cropping
        #image from bytes after opencv enconding at the another application
        image = np.frombuffer(img_bytes, dtype=np.uint8)
        # image = (img * 255).round().astype(np.uint8)
        # process your img_arr here
        cvImg = cv2.imdecode(image, cv2.IMREAD_COLOR)
        #add car's difference detection from foreground
```

```
        grey = cv2.cvtColor(img_arr,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(grey,(5,5),0)
        dilated = cv2.dilate(blur,np.ones((3,3)))
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,  (2, 2))
        closing = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE,  kernel)
        project_dir = os.path.dirname(os.path.abspath(__file__))
        car_cascade_src = os.path.join(project_dir, 'train_Cas1.xml') #train_Cas1
        car_cascade = cv2.CascadeClassifier(car_cascade_src)
        cars = car_cascade.detectMultiScale(closing, 1.1, 1)
        max_gray_percentage = 0
        max_gray_countour_tuple = ()
        cnt = 0
        for (x,y,w,h) in cars:
            gray_percentage = 0
            gray_counter = 0
            for i in range(y, y+h):
                for j in range(x, x+w):
                    (b, g, r) = foreground_image[i, j] #(b, g, r) format
                    if (b == 127 and g == 127 and r == 127): #if pixel is gray
                        gray_counter +=1
            gray_percentage = gray_counter / (h) * (w)
            if ((w > 320 and w < 520 ) and (y+h < cvImg.shape[0] * 0.8) and
(gray_percentage > 10)): #@todo
                if (gray_percentage > max_gray_percentage):
                    max_gray_percentage = gray_percentage
                    max_gray_countour_tuple = (x,y,w,h)
                cv2.rectangle(img_arr,(x,y),(x+w,y+h),(255,0,0), 5)
                cnt += 1
        if (len(max_gray_countour_tuple) == 0):
            print("WARNING MAX max_gray_countour_tuple is EMPTY" )
        x = max_gray_countour_tuple[0]
        y = max_gray_countour_tuple[1]
        w = max_gray_countour_tuple[2]
        h = max_gray_countour_tuple[3]
        cropped_contour= original_image[y:y+h, x:x+w] # with (x,y,w,h) format use
range y:y+h, x:x+w
        cur_car_img_file = saveArrayToImage(cropped_contour, "current-car-shape")
        cur_car_coordsCenterX = int(x + w/2)
        cur_car_coordsCenterY = int(y + h/2) # width height
        cv2.circle(img_arr, (cur_car_coordsCenterX, cur_car_coordsCenterY), 10, (0, 0,
255), 3)
        saveArrayToImage(img_arr, "cars-detected-" + str(cnt) + "-")
        camera_data = CameraInfo.query.all()
        save_new_cur_car = True
        separator = "|"
        if(len(camera_data) != 0):
            save_new_cur_car = False
```

```
        if (save_new_cur_car):
            cur_car_date=cur_car_img_file.split('date')[1].split('.')[0]
            c = CameraInfo(cur_car_date, cur_car_img_file, str(cur_car_coordsCenterX)
+ separator + str(cur_car_coordsCenterY), 0)
            db.session.add(c)
            db.session.commit()
        else:
            cam = CameraInfo.query.order_by(CameraInfo.id.desc()).first()
            print(f'-> {cam.cur_car_img}')
            curFrameCenter, speed = searchCurrentCarInFrameFlann(original_image,
cam.cur_car_img, dict(coords = cam.cur_car_coords, separator = separator)) #
cropped_contour_from_db => cam.carimagename
        img_list = get_img_list()
        response = {
            'message': 'image received. size={}x{},'.format(cvImg.shape[1],
cvImg.shape[0]),
            'carimages': img_list,
            'cars_found': '{}'.format(str(cnt)),
            'carspeed': speed
        }
        response_pickled = jsonpickle.encode(response)
        return Response(response=response_pickled, status=200,
mimetype="application/json", headers={'Access-Control-Allow-Origin':'*'})

    def get_img_list():
        project_dir = os.path.dirname(os.path.abspath(__file__))
        recognitionDir = os.path.join(project_dir, STATIC_FOLDER)
        img_list = []
        for (dirpath, dirnames, filenames) in walk(recognitionDir):
            img_list.extend(filenames)
            break
        return img_list

    def searchCurrentCarInFrame(new_frame, cropped_from_db): # query =>
new_frame [image as np.array], cropped_from_db => train_img
        query_img = new_frame
        train_img = cv2.imread(cropped_from_db)
        # Convert it to grayscale
        query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
        train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
        # Initialize the ORB detector algorithm
        orb = cv2.ORB_create()
        # Now detect the keypoints and compute
        # the descriptors for the query image
        # and train image
        queryKeypoints, queryDescriptors =
orb.detectAndCompute(query_img_bw,None)
```

```
        trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)
        # Initialize the Matcher for matching
        # the keypoints and then match the
        # keypoints
        matcher = cv2.BFMatcher()
        matches = matcher.match(queryDescriptors,trainDescriptors)
        # draw the matches to the final image
        # containing both the images the drawMatches()
        # function takes both images and keypoints
        # and outputs the matched query image with
        # its train image
        final_img = cv2.drawMatches(query_img, queryKeypoints,
        train_img, trainKeypoints, matches[:40],None)
        final_img = cv2.resize(final_img, (1000,650))
        saveArrayToImage(final_img, "MATCHES-SIFT-final")

    def searchCurrentCarInFrameFlann(new_frame, cropped_from_db, prevFrameConfig): # query => new_frame [image as np.array], cropped_from_db => train_img
        query_img = new_frame
        train_img = cv2.imread(cropped_from_db)
        # Convert it to grayscale
        query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
        train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
        img1 = query_img_bw # queryImage
        img2 = train_img_bw # trainImage
        # Initiate SIFT detector
        sift = cv2.SIFT_create()
        # find the keypoints and descriptors with SIFT
        kp1, des1 = sift.detectAndCompute(img1,None)
        kp2, des2 = sift.detectAndCompute(img2,None)
        # FLANN parameters
        FLANN_INDEX_KDTREE = 1
        index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
        search_params = dict(checks=50) # or pass empty dictionary
        flann = cv2.FlannBasedMatcher(index_params,search_params)
        matches = flann.knnMatch(des1,des2,k=2)
        # Need to draw only good matches, so create a mask
        matchesMask = [[0,0] for i in range(len(matches))]
        # ratio test as per Lowe's paper
        X = []
        Y = []
        matchesCount = 0
        for i,(m,n) in enumerate(matches):
            if m.distance < 0.7*n.distance:
                matchesMask[i]=[1,0]
                ## Notice: How to get the index
```

```
                pt1 = kp1[m.queryIdx].pt
                pt2 = kp2[m.trainIdx].pt
                matchesCount +=1
                cv2.circle(img1, (int(pt1[0]),int(pt1[1])), 5, (255,0,255), 3)
                X.append(int(pt1[0]))
                Y.append(int(pt1[1]))
            draw_params = dict(matchColor = (0,255,0),
                    singlePointColor = (255,0,0),
                    matchesMask = matchesMask,
                    flags = cv2.DrawMatchesFlags_DEFAULT)
            img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
            print("MATCHES  COUNT")
            print(matchesCount)
            coordinatesCenter = findCenters(X, Y)
            prevCenterCoords = list(map(int, prevFrameConfig['coords'].split(prevFrameConfig['separator'])))
            print ('Distance')
            rawDistance = math.dist(coordinatesCenter, prevCenterCoords)
            print (rawDistance)
            hCoef = 0.554
            rawSpeed = (rawDistance / get_time_between_frame()) * hCoef
            color = (0, 255, 0)
            # Line thickness of 9 px
            thickness = 9
            print("Current Frame Center Coordinates")
            print(tuple(coordinatesCenter))
            print("Previous Frame Center Coordinates")
            print(tuple(prevCenterCoords))
            cv2.line(img3, tuple(coordinatesCenter), tuple(prevCenterCoords), color, thickness)
            cv2.circle(img3, (int(coordinatesCenter[0]), int(coordinatesCenter[1])), 10, (255,0,0), 3)
            saveArrayToImage(img3, "MATCHES-Flann-final")
            return (coordinatesCenter, rawSpeed)

        def findCenters(xList, yList):
            coordinates = []
            xCenter = np.sum(xList)/len(xList)
            yCenter = np.sum(yList)/len(yList)
            coordinates.append(int(xCenter))
            coordinates.append(int(yCenter))
            return coordinates

        class Vertex:
            def __init__(self, node):
                self.id = node
```

```
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Predecessor
        self.previous = None

    def __lt__(self, other):
        return self.distance<other.distance

    def add_neighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def get_connections(self):
        return self.adjacent.keys()

    def get_id(self):
        return self.id

    def get_weight(self, neighbor):
        return self.adjacent[neighbor]

    def set_distance(self, dist):
        self.distance = dist

    def get_distance(self):
        return self.distance

    def set_previous(self, prev):
        self.previous = prev

    def set_visited(self):
        self.visited = True

    def __str__(self):
        return str(self.id) + ' adjacent: ' + str([x.id for x in self.adjacent])

class Graph:
    def __init__(self):
        self.vert_dict = {}
        self.vert_dict_addr = {
            "68-Desantnykiv, 10": "0",
            "3 Slobidska, 3" : "1",
            "Velyka Morska, 142": "2",
            "Tsentralniy prospekt, 154": "3",
            "Chkalova, 108D" : "4",
```

```
            "Sadova, 18" : "5",
            "Tsentralniy prospekt, 107": "6",
            "Sadova, 6": "7",
            "Mariupolska, 11": "8",
            "Pushkinska, 10,": "9",
            "Tsentralniy prospekt, 68": "10",
            "Mariupolska, 40/5": "11",
            "Tsentralniy prospekt, 96": "12",
            "Inzhenerna, 46": "13",
            "Mala Morska, 39": "14",
            "Mariupolska, 57A": "15",
            "Sinna, 55": "16",
            "Mariupolska,85B": "17",
            "Sadova, 40A": "18",
            "Sadova, 50/3": "19",
            "Mala Morska, 63": "20",
            "Kuznetska, 1": "21",
            "Korabeliv lane, 19": "22",
            "Pogranichna, 34": "23",
            "Pushkinska, 69A": "24",
            "Pushkinska, 61": "25",
            "Sinna, 1": "26",
            "Tsentralniy prospekt, 31": "27",
            "Transportny lane, 6": "28",
            "Tsentralniy prospekt, 5": "29",
            "Generala Karpenka, 46": "30",
            "Buznyka, 31": "31",
            "Generala Karpenka, 34": "32",
            "Generala Karpenka, 12B": "33",
            "Krylova, 45": "34",
            "Bila, 57": "35",
            "Krylova, 32": "36",
            "Krylova, 19/1": "37",
            "Industrialna, 105": "38",
            "Kurortna, 2": "39",
            "Generala Karpenka, 23": "40",
            "Ozerna, 1V": "41",
            "Ozerna, 13": "42"
        }
        self.num_vertices = 0

    def __iter__(self):
        return iter(self.vert_dict.values())

    def add_vertex(self, node):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(node)
```

```
            self.vert_dict[node] = new_vertex
            return new_vertex

        def get_vertex_num(self, addr):
          return self.vert_dict_addr[addr] # @todo add addr in check like commented
above

        def get_vertex(self, addr):
            n = self.get_vertex_num(addr)
            if n in self.vert_dict:
                return self.vert_dict[n]
            else:
                return None

        def add_edge(self, frm, to, cost = 0):
            if frm not in self.vert_dict:
                self.add_vertex(frm)
            if to not in self.vert_dict:
                self.add_vertex(to)

            self.vert_dict[frm].add_neighbor(self.vert_dict[to], cost)
            self.vert_dict[to].add_neighbor(self.vert_dict[frm], cost)

        def get_vertices(self):
            return self.vert_dict.keys()

        def set_previous(self, current):
            self.previous = current

        def get_previous(self, current):
            return self.previous

    def shortest(v, path):
        ''' make shortest path from v.previous'''
        if v.previous:
            path.append(v.previous.get_id())
            shortest(v.previous, path)
        return

    def dijkstra(aGraph, start, target):
        print ('''Dijkstra's shortest path''')
        # Set the distance for the start node to zero
        start.set_distance(0)
        # Put tuple pair into the priority queue
        unvisited_queue = [(v.get_distance(),v) for v in aGraph]
        heapq.heapify(unvisited_queue)
        while len(unvisited_queue):
```

```
            # Pops a vertex with the smallest distance
            uv = heapq.heappop(unvisited_queue)
            current = uv[1]
            current.set_visited()
            #for next in v.adjacent:
            for next in current.adjacent:
                # if visited, skip
                if next.visited:
                    continue
                new_dist = current.get_distance() + current.get_weight(next)
                if new_dist < next.get_distance():
                    next.set_distance(new_dist)
                    next.set_previous(current)
                    print ('updated : current = %s next = %s new_dist = %s' \
                            %(current.get_id(), next.get_id(), next.get_distance()))
                else:
                    print ('not updated : current = %s next = %s new_dist = %s' \
                            %(current.get_id(), next.get_id(), next.get_distance()))
            # Rebuild heap
            # 1. Pop every item
            while len(unvisited_queue):
                heapq.heappop(unvisited_queue)
            # 2. Put all vertices not visited into the queue
            unvisited_queue = [(v.get_distance(),v) for v in aGraph if not v.visited]
            heapq.heapify(unvisited_queue)

        @cross_origin()
        @app.route("/getspeed")
        def get_camera_speed():
            response = {
                'camspeed': get_speed_by_camera()
            }
            # encode response using jsonpickle
            response_pickled = jsonpickle.encode(response)
            return Response(response=response_pickled, status=200,
mimetype="application/json", headers={'Access-Control-Allow-Origin':'*'})
```

ДОДАТОК Г
Лістинг програмного коду клієнтської частини аналізу швидкості руху автомобілів дорожньою ділянкою

```
$(document).ready(function() {
    "use strict";
    var window_width = $(window).width(),
        window_height = window.innerHeight,
        header_height = $(".default-header").height(),
        header_height_static = $(".site-header.static").outerHeight(),
        fitscreen = window_height - header_height;
    $(".fullscreen").css("height", window_height)
    $(".fitscreen").css("height", fitscreen);
    //------- Niceselect  js --------//
    if (document.getElementById("default-select")) {
        $('select').niceSelect();
    };
    if (document.getElementById("default-select2")) {
        $('select').niceSelect();
    };
    //------- Lightbox  js --------//
    $('.img-gal').magnificPopup({
        type: 'image',
        gallery: {
            enabled: true
        }
    });
    $('.play-btn').magnificPopup({
        type: 'iframe',
        mainClass: 'mfp-fade',
        removalDelay: 160,
        preloader: false,
        fixedContentPos: false
    });
    //------- Datepicker  js --------//
     $( function() {
        $( "#datepicker" ).datepicker();
        $( "#datepicker2" ).datepicker();
     } );
    //------- Superfist nav menu  js --------//
    $('.nav-menu').superfish({
        animation: {
            opacity: 'show'
        },
        speed: 400
    });
    if ( $(".site-content").hasClass("instagram") ) {
        footerIntagram();
```

```
        }
        function footerIntagram(){
            var feed = new Instafeed({
                target: "footer-ig-stream",
                get: "user",
                limit : 6,
                resolution: 'standard_resolution',
                userId: 2159114835,
                accessToken:
"2159114835.9e667eb.7a37f9b944ea4023b94541c661cbf43d",
                template: '<a href="{{image}}" class="mfp-ig-popup" data-effect="mfp-
zoom-in" title="{{title}}"><img src="{{image}}" alt="{{caption}}"></a>',
                after: function() {
                    $(".mfp-ig-popup").magnificPopup({
                        type: "image",
                        removalDelay: 500, //delay removal by X to allow out-animation
                        image: {
                            cursor: null
                        },
                        gallery: {
                            enabled: true // set to false to disable gallery
                        },
                        callbacks: {
                            beforeOpen: function() {
                                // just a hack that adds mfp-anim class to markup
                                this.st.image.markup = this.st.image.markup.replace("mfp-
figure", "mfp-figure mfp-with-anim");
                                this.st.mainClass = this.st.el.attr("data-effect");
                            }
                        },
                        midClick: true
                    });
                }
            });
            feed.run();
        }
        //------- Mobile Nav  js --------//
        if ($('#nav-menu-container').length) {
            var $mobile_nav = $('#nav-menu-container').clone().prop({
                id: 'mobile-nav'
            });
            $mobile_nav.find('> ul').attr({
                'class': '',
                'id': ''
            });
            $('body').append($mobile_nav);
```

```
            $('body').prepend('<button type="button" id="mobile-nav-toggle"><i
class="lnr lnr-menu"></i></button>');
            $('body').append('<div id="mobile-body-overly"></div>');
            $('#mobile-nav').find('.menu-has-children').prepend('<i class="lnr lnr-
chevron-down"></i>');
            $(document).on('click', '.menu-has-children i', function(e) {
               $(this).next().toggleClass('menu-item-active');
               $(this).nextAll('ul').eq(0).slideToggle();
               $(this).toggleClass("lnr-chevron-up lnr-chevron-down");
            });
            $(document).on('click', '#mobile-nav-toggle', function(e) {
               $('body').toggleClass('mobile-nav-active');
               $('#mobile-nav-toggle i').toggleClass('lnr-cross lnr-menu');
               $('#mobile-body-overly').toggle();
            });
               $(document).on('click', function(e) {
               var container = $("#mobile-nav, #mobile-nav-toggle");
               if (!container.is(e.target) && container.has(e.target).length === 0) {
                  if ($('body').hasClass('mobile-nav-active')) {
                     $('body').removeClass('mobile-nav-active');
                     $('#mobile-nav-toggle i').toggleClass('lnr-cross lnr-menu');
                     $('#mobile-body-overly').fadeOut();
                  }
               }
            });
         } else if ($("#mobile-nav, #mobile-nav-toggle").length) {
            $("#mobile-nav, #mobile-nav-toggle").hide();
         }
         //------- Smooth Scroll  js --------//
         $('.nav-menu a, #mobile-nav a, .scrollto').on('click', function() {
            if (location.pathname.replace(/^\//, '') == this.pathname.replace(/^\//, '') &&
location.hostname == this.hostname) {
               var target = $(this.hash);
               if (target.length) {
                  var top_space = 0;
                  if ($('#header').length) {
                     top_space = $('#header').outerHeight();
                     if (!$('#header').hasClass('header-fixed')) {
                        top_space = top_space;
                     }
                  }
                  $('html, body').animate({
                     scrollTop: target.offset().top - top_space
                  }, 1500, 'easeInOutExpo');

                  if ($(this).parents('.nav-menu').length) {
                     $('.nav-menu .menu-active').removeClass('menu-active');
```

```
                $(this).closest('li').addClass('menu-active');
            }
            if ($('body').hasClass('mobile-nav-active')) {
                $('body').removeClass('mobile-nav-active');
                $('#mobile-nav-toggle i').toggleClass('lnr-times lnr-bars');
                $('#mobile-body-overly').fadeOut();
            }
            return false;
        }
    }
});

$(document).ready(function() {
    $('html, body').hide();
    if (window.location.hash) {
        setTimeout(function() {
            $('html, body').scrollTop(0).show();
            $('html, body').animate({
                scrollTop: $(window.location.hash).offset().top - 108
            }, 1000)
        }, 0);
    } else {
        $('html, body').show();
    }
    if ( window.location.pathname === "/service") {
        var canvas = this.getElementById("myCanvas"); // document
        var cameraPositions = [
          {'name': '0', 'pos': [1849, 315], 'addr': "68-Desantnykiv, 10"},
          {'name': '1', 'pos': [1767, 343], 'addr': "3 Slobidska, 3"},
          {'name': '2', 'pos': [1772, 389],'addr': "Velyka Morska, 142"}, {'name': '3',
'pos': [1810, 517],'addr': "Tsentralniy prospekt, 154"},
          {'name': '4', 'pos': [1803, 623],'addr': "Chkalova, 108D"}, {'name': '5',
'pos': [1584, 604],'addr': "Sadova, 18"},
          {'name': '6', 'pos': [1592, 495],'addr': "Tsentralniy prospekt, 107"},
{'name': '7', 'pos': [1593, 425],'addr': "Sadova, 6"},
          {'name': '8', 'pos': [1386, 333],'addr': "Mariupolska, 11"}, {'name': '9',
'pos': [1198, 243],'addr': "Pushkinska, 10,"},
          {'name': '10', 'pos': [1095, 455],'addr': "Tsentralniy prospekt, 68"},
{'name': '11', 'pos': [1327, 475],'addr': "Mariupolska, 40/5"},
          {'name': '12', 'pos': [1503, 488],'addr': "Tsentralniy prospekt, 96"},
{'name': '13', 'pos': [1491,592],'addr': "Inzhenerna, 46"},
          {'name': '14', 'pos': [1402, 591],'addr': "Mala Morska, 39"}, {'name': '15',
'pos': [1308,564],'addr': "Mariupolska, 57A"},
          {'name': '16', 'pos': [1279,639],'addr': "Sinna, 55"}, {'name': '17', 'pos':
[1264,683],'addr': "Mariupolska,85B"},
          {'name': '18', 'pos': [1575,716],'addr': "Sadova, 40A"}, {'name': '19', 'pos':
[1568,860],'addr': "Sadova, 50/3"},
```

```
                {'name': '20', 'pos': [1394,867],'addr': "Mala Morska, 63"}, {'name':
'21','pos': [1185,890],'addr': "Kuznetska, 1"},
                {'name': '22', 'pos': [1159,808],'addr': "Korabeliv lane, 19"}, {'name': '23',
'pos': [1183,689],'addr': "Pogranichna, 34"},
                {'name': '24', 'pos': [1025,633],'addr': "Pushkinska, 69A"}, {'name': '25',
'pos': [1055,556],'addr': "Pushkinska, 61"},
                {'name': '26', 'pos': [961,523],'addr': "Sinna, 1"}, {'name': '27', 'pos':
[981,445],'addr': "Tsentralniy prospekt, 31"},
                {'name': '28', 'pos': [767,577],'addr': "Transportny lane, 6"}, {'name': '29',
'pos': [683,416],'addr': "Tsentralniy prospekt, 5"},
                {'name': '30', 'pos': [583,408],'addr': "Generala Karpenka, 46"}, {'name':
'31', 'pos': [671,510],'addr': "Buznyka, 31"},
                {'name': '32', 'pos': [542,503],'addr': "Generala Karpenka, 34"}, {'name':
'33', 'pos': [429,567],'addr': "Generala Karpenka, 12B"},
                {'name': '34', 'pos': [414, 648],'addr': "Krylova, 45"}, {'name': '35', 'pos':
[505,707],'addr': "Bila, 57"},
                {'name': '36', 'pos': [599,774],'addr': "Krylova, 32"}, {'name': '37', 'pos':
[690,903],'addr': "Krylova, 19/1"},
                {'name': '38', 'pos': [903,868],'addr': "Industrialna, 105"}, {'name': '39',
'pos': [419,860],'addr': "Kurortna, 2"},
                {'name': '40', 'pos': [384,774],'addr': "Generala Karpenka, 23"}, {'name':
'41', 'pos': [209,847],'addr': "Ozerna, 1V"},
                {'name': '42', 'pos': [256,937],'addr': "Ozerna, 13"},
            ]
            function checkAPoint(a, b, x, y, r) {
                let dist_points = (a - x) * (a - x) + (b - y) * (b - y);
                r *= r;
                if (dist_points < r) {
                    return true;
                }
                return false;
            }
            function getCursorPosition(canvas, event) {
                const rect = canvas.getBoundingClientRect()
                const a = event.clientX - rect.left
                const b = event.clientY - rect.top
                let camera = cameraPositions.find((camPos) => {
                    let [x, y] = camPos.pos
                    let isInside = checkAPoint(a, b, x, y, 35);
                    return isInside ? camPos : undefined;
                })
                if(camera) {
                    console.log("x: " + a + " y: " + b)
                    console.log(camera)
                    let url = '/camera-about';
                    window.location.href = url + '?' + new URLSearchParams(camera);
                } else {
```

```
        console.log("NOTHING FOUND at x: " + a + " y: " + b)
    }
}
canvas.addEventListener('mousedown', function(e) {
    getCursorPosition(canvas, e)
})
var ctx = canvas.getContext("2d");
            class CamerasGraph {
  edges = []
  constructor(size) {
    this.size = size;
  }
  add_edge(from, to, speed) {
    this.edges.push({
      'from': from,
      'to': to,
      'speed': speed
    })
  }
}
const g = new CamerasGraph();
g.add_edge(0, 1, 0)
g.add_edge(1, 2, 0)
g.add_edge(2, 7, 0)
g.add_edge(2, 3, 0)
g.add_edge(3, 6, 0)
g.add_edge(3, 4, 0)
g.add_edge(4, 5, 0)
g.add_edge(5, 6, 0)
g.add_edge(5, 13, 0)
g.add_edge(5, 18, 0)
g.add_edge(6, 7, 0)
g.add_edge(6, 12, 0)
g.add_edge(7, 8, 0)
g.add_edge(8, 9, 0)
g.add_edge(8, 11, 0)
g.add_edge(9, 10, 0)
g.add_edge(10, 11, 0)
g.add_edge(10, 27, 0)
g.add_edge(10, 25, 0)
g.add_edge(11, 12, 0)
g.add_edge(11, 15, 0)
g.add_edge(12, 13, 0)
g.add_edge(13, 14, 0)
g.add_edge(14, 15, 0)
g.add_edge(15, 16, 0)
g.add_edge(16, 17, 0)
```

```
g.add_edge(16, 25, 0)
g.add_edge(17, 18, 0)
g.add_edge(17, 23, 0)
g.add_edge(18, 19, 0)
g.add_edge(19, 20, 0)
g.add_edge(20, 21, 0)
g.add_edge(21, 22, 0)
g.add_edge(21, 38, 0)
g.add_edge(22, 23, 0)
g.add_edge(23, 24, 0)
g.add_edge(24, 25, 0)
g.add_edge(25, 26, 0)
g.add_edge(26, 27, 0)
g.add_edge(26, 28, 0)
g.add_edge(27, 29, 0)
g.add_edge(28, 31, 0)
g.add_edge(29, 31, 0)
g.add_edge(29, 30, 0)
g.add_edge(30, 32, 0)
g.add_edge(31, 32, 0)
g.add_edge(32, 33, 0)
g.add_edge(33, 34, 0)
g.add_edge(34, 35, 0)
g.add_edge(34, 40, 0)
g.add_edge(35, 40, 0)
g.add_edge(35, 36, 0)
g.add_edge(36, 39, 0)
g.add_edge(36, 37, 0)
g.add_edge(37, 38, 0)
g.add_edge(39, 40, 0)
g.add_edge(39, 42, 0)
g.add_edge(40, 41, 0)
g.add_edge(41, 42, 0)
const mapImg = new Image();
mapImg.onload = () => {
   ctx.drawImage(mapImg,  0, 0);
   ctx.beginPath();
   ctx.moveTo(cameraPositions[0].pos[0], cameraPositions[0].pos[1]);
   ctx.strokeStyle = '#85A5E0';
   ctx.lineWidth = 5;
   ctx.font = "20px serif";
   g.edges.map((edge) => {
     let from = cameraPositions.find((camPos) => camPos.name ==
+edge.from)
     let { pos: [fromX, fromY] } = from;
     ctx.moveTo(fromX, fromY);
     ctx.ellipse(fromX,fromY, 4, 4, Math.PI / 4, 0, 2 * Math.PI);
```

```
                let to = cameraPositions.find((camPos) => camPos.name == +edge.to)
                let { pos: [toX, toY] } = to;
                ctx.lineTo(toX, toY);
                ctx.ellipse(toX,toY, 4, 4, Math.PI / 4, 0, 2 * Math.PI);
              });

              ctx.stroke();
              drawRoute(route);
              ctx.beginPath();
              ctx.moveTo(cameraPositions[0].pos[0], cameraPositions[0].pos[1]);
                        ctx.font = "20px serif";
              ctx.fillStyle = "#74797B";
              g.edges.map((edge) => {
              let from = cameraPositions.find((camPos) => camPos.name ==
+edge.from)
              ctx.fillText(from.addr, fromX + 7 ,fromY - 7);
              let to = cameraPositions.find((camPos) => camPos.name == +edge.to)
              let { pos: [toX, toY] } = to;
              ctx.fillText(to.addr, toX + 7 , toY - 7);
              let camera_image = new Image();
              camera_image.src = '/static/taxi-master-files/taxi-master/img/camera-
img-30.png';
              camera_image.onload = () => {
                 ctx.drawImage(camera_image, fromX, fromY + 4);
                 ctx.drawImage(camera_image, toX, toY + 4);
              }
              });
              ctx.stroke();
            };
            mapImg.src = "/static/taxi-master-files/taxi-master/img/header-bg.png";
            function drawRoute(routeData) {
              let route = JSON.parse(routeData.replaceAll('\"', '\'"));
              ctx.beginPath();
              ctx.lineWidth = 5;
              ctx.strokeStyle = '#F700FF';
              let drawFrom = cameraPositions.find((camPos) => camPos.name ==
route[0])
              ctx.moveTo(drawFrom.pos[0], drawFrom.pos[1]); // x, y
              for (let i = 1; i < route.length; i++) {
                 let drawTo = cameraPositions.find((camPos) => camPos.name ==
route[i])
                 let { pos: [toX, toY] } = drawTo;
                 ctx.lineTo(toX, toY);
              }
              ctx.stroke();
              console.log("Done")
            }
```

```
} // end of "/service"

if ( window.location.pathname === "/camera-about") {

  $('#loginModal').modal('show');

  let nIntervId;

  function runUpdate() {
    // check if an interval has already been set up
    if (!nIntervId) {
    nIntervId = setInterval(updateData, 1000);
    }
  }
  function updateData() {
    let url = "http://mgybarev.pythonanywhere.com/update-img-list"
                    $.ajax(url, {
      type: 'GET',
      contentType : 'application/json',
      crossDomain:true,
      success: buildGallery
    });
    let speedUrl = "http://mgybarev.pythonanywhere.com/getspeed"
                    $.ajax(speedUrl, {
      type: 'GET',
      contentType : 'application/json',
      crossDomain:true,
      success: function(response) {
        $("#camSpeed").html(response.camspeed);
      }
    });
  }
  runUpdate()
  let buildGallery = function(response) {
    console.log("POST REQUEST  /getframe -> " + response);
    let galContainer =  $("#galleryContainer");
    galContainer.empty();
    for (let i = 0; i < response.carimages.length; i+=2) {
      console.log(i);
      let base = '/static/rec-images/';
      let template = '<div class="col-lg-4 single-gallery">';
      let url = base + response.carimages[i]
      template += `
        <a href="${url}" class="img-gal">
        <img class="img-fluid" src="${url}" alt=""></a>`;
      if(i+1 < response.carimages.length) {
        url = base + response.carimages[i+1]
```

```
                    template += `
                    <a href="${url}" class="img-gal">
                    <img class="img-fluid" src="${url}" alt=""></a>`;
                }
                template += '</div>';
                console.log(template);
                let domTemplate = $.parseHTML(template);
                galContainer.append(domTemplate);
            }
            $('.img-gal').magnificPopup({
                type: 'image',
                gallery: {
                    enabled: true
                }
            });
        }
        $("a[name='exp-btn']").on("click", function(e) {
            console.log("Frame sending...");// $(this) is document instance
            e.stopPropagation();
            let status = e.target.text
            let data = { "num" : e.target.text.split(' ')[2] };
            e.target.text = "Working ..."
            let url =
"https://SunnyMediumpurpleAutoresponder.mgybarev.repl.co/getframe";
                                $.ajax(url, {
                type: 'POST',
                contentType : 'application/json',
                data: JSON.stringify(data),
                crossDomain:true,
                success: (response) => {
                    e.target.text = status;
                    buildGallery(response);
                }
            });

        });

    } // end of "/camera-about"

});

//------- Header Scroll Class  js --------//
$(window).scroll(function() {
    if ($(this).scrollTop() > 100) {
        $('#header').addClass('header-scrolled');
    } else {
        $('#header').removeClass('header-scrolled');
```

```
    }
});
//------- Google Map  js --------//
if (document.getElementById("map")) {
    google.maps.event.addDomListener(window, 'load', init);
    function init() {
        var mapOptions = {
            zoom: 11,
            center: new google.maps.LatLng(40.6700, -73.9400), // New York
            styles: [{
                "featureType": "water",
                "elementType": "geometry",
                "stylers": [{
                    "color": "#e9e9e9"
                }, {
                    "lightness": 17
                }]
            }, {
                "featureType": "landscape",
                "elementType": "geometry",
                "stylers": [{
                    "color": "#f5f5f5"
                }, {
                    "lightness": 20
                }]
            }, {
                "featureType": "road.highway",
                "elementType": "geometry.fill",
                "stylers": [{
                    "color": "#ffffff"
                }, {
                    "lightness": 17
                }]
            }, {
                "featureType": "road.highway",
                "elementType": "geometry.stroke",
                "stylers": [{
                    "color": "#ffffff"
                }, {
                    "lightness": 29
                }, {
                    "weight": 0.2
                }]
            }, {
                "featureType": "road.arterial",
                "elementType": "geometry",
                "stylers": [{
```

```
        "color": "#ffffff"
    }, {
        "lightness": 18
    }]
}, {
    "featureType": "road.local",
    "elementType": "geometry",
    "stylers": [{
        "color": "#ffffff"
    }, {
        "lightness": 16
    }]
}, {
    "featureType": "poi",
    "elementType": "geometry",
    "stylers": [{
        "color": "#f5f5f5"
    }, {
        "lightness": 21
    }]
}, {
    "featureType": "poi.park",
    "elementType": "geometry",
    "stylers": [{
        "color": "#dedede"
    }, {
        "lightness": 21
    }]
}, {
    "elementType": "labels.text.stroke",
    "stylers": [{
        "visibility": "on"
    }, {
        "color": "#ffffff"
    }, {
        "lightness": 16
    }]
}, {
    "elementType": "labels.text.fill",
    "stylers": [{
        "saturation": 36
    }, {
        "color": "#333333"
    }, {
        "lightness": 40
    }]
}, {
```

```
                "elementType": "labels.icon",
                "stylers": [{
                    "visibility": "off"
                }]
            }, {
                "featureType": "transit",
                "elementType": "geometry",
                "stylers": [{
                    "color": "#f2f2f2"
                }, {
                    "lightness": 19
                }]
            }, {
                "featureType": "administrative",
                "elementType": "geometry.fill",
                "stylers": [{
                    "color": "#fefefe"
                }, {
                    "lightness": 20
                }]
            }, {
                "featureType": "administrative",
                "elementType": "geometry.stroke",
                "stylers": [{
                    "color": "#fefefe"
                }, {
                    "lightness": 17
                }, {
                    "weight": 1.2
                }]
            }]
        };
        var mapElement = document.getElementById('map');
        var map = new google.maps.Map(mapElement, mapOptions);
        var marker = new google.maps.Marker({
            position: new google.maps.LatLng(40.6700, -73.9400),
            map: map,
            title: 'Snazzy!'
        });
    }
}
//------- Mailchimp js --------//
$(document).ready(function() {
    $('#mc_embed_signup').find('form').ajaxChimp();
});
});
```