КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено
Завідувач кафедри ІСТ

_____ Олександр КУЧАНСЬКИЙ

(підпис) (ім'я, ПРІЗВИЩЕ)

"___"_____2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: « Система аналізу та контролю переміщення міжобласного рейсового транспорту»

Виконав (-ла): студент (-ка) 4 курсу, групи ІР-41

(шифр групи)

Поліна МІШИНА _____

(Ім'я, ПРІЗВИЩЕ) (підпис)

Керівник к.т.н., доц. Ростислав ЛІСНЕВСЬКИЙ _____

(посада, науковий ступінь, вчене звання, Ім'я, ПРІЗВИЩЕ) (підпис)

Консультант нормо контроль Ростислав ЛІСНЕВСЬКИЙ _____

(назва розділу) (посада, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент директор ТОВ «Примавера компані» Олександр КОРОЛЬ _____

(посада, науковий ступінь, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у пояснювальна записка немає запозичень з праць інших авторів без відповідних посилань.

Здобувач освіти _____

(підпис)

Київ – 2022 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційні системи та технології
Освітній рівень Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітня програма Програмні технології інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри,
д.т.н., доцент
Олександр КУЧАНСЬКИЙ

_____
«__» _____2022 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Поліна МІШИНА

Група: ІР-41

1. **Тема кваліфікаційна робота бакалавра**: «Система аналізу та контролю переміщення міжобласного рейсового транспорту».

Затверджена протоколом засідання кафедри ІСТ №05/21_22 від 03.12.2021 року

2. **Строк подання студентом готової роботи** - «22» червня 2022 р.

3. **Вихідні дані до роботи**: дослідження транспортної галузі. Перелік необхідного апаратного забезпечення. Програмне рішення для збору та аналізу даних в галузі автоперевезення. Дані про переміщення транспортних засобів з фіксованим та чітким часом для подальшого розрахунку розкладу руху за маршрутом.

4. Зміст роботи: РОЗДІЛ 1 АНАЛІЗ ТА ОПИС СФЕР ЗАСТОСУВАННЯ СИСТЕМИ, ЩО ДОСЛІДЖУЄТЬСЯ. (Аналіз предметної області, Дослідження типів систем відстеження транспортного засобу на великих дистанціях, Вимоги до складових системи); РОЗДІЛ 2 ПРОЕКТУВАННЯ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СИТЕМИ (Створення контекстної та фізичної моделі БД, Програмні технології, що використовуються, Апаратне забезпечення, Опис

роботи системи); РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ (початок роботи, робота з датчиками, алгоритм прогнозування часу).

5. **Перелік графічного матеріалу**: графічне відображення схеми роботи GPS систем, таблиця факторів, що впливають на точність розрахунків GPS-систем, таблиця різниці між системами активного та пасивного відстеження, концептуальна модель бази даних, логічна та фізична моделі бази даних, графічні зображення модулів, робота системи у діаграмі послідовностей, архітектура системи, алгоритм роботи, графічний інтерфейс застосунку.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	01.09.2021-01.10.2021	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	03.12.2021	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2021	виконано
4. Написання І розділу кваліфікаційної роботи	19.03.2022	виконано
5. Написання ІІ розділу кваліфікаційної роботи	25.04.2022	виконано
6. Написання ІІІ розділу кваліфікаційної роботи	29.04.2022	виконано
7. Підготовка висновків і пропозицій	30.04.2022	виконано
8. Попередній захист кваліфікаційної роботи	12.05.2022	виконано
9. Перевірка на плагіат	13.05.2022-15.06.2022	виконано
10. Нормоконтроль	02.06.2022 - 06.06.2022	виконано
11. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	15.06.2022	виконано
11. Захист кваліфікаційної роботи бакалавра	23.06.2022 - 24.06.2022	

Дата видачі завдання «01» грудня 2021 р.

Керівник роботи: к.т.н., доц. Ростислав ЛІСНЕВСЬКИЙ _____
(підпис)

Завдання прийняв до виконання:
Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41

Поліна МІШИНА _____
(Власне Ім’я, ПРІЗВИЩЕ) (підпис)

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційних систем та технологій

Освітня програма «Програмні технології інтернет речей»

Кваліфікаційна робота бакалавра Поліна МІШИНА

Тема роботи: «Система аналізу та контролю переміщення міжобласного рейсового транспорту».

Мета кваліфікаційної роботи бакалавра – розробка системи збору та аналізу даних про рух рейсового транспорту, проектування і програмна реалізація системи з можливістю попереднього розрахунку часу прибуття транспортного засобу до кожної з зупинок.

Об'єкт дослідження – система фіксації та розрахунку часу на подорож для міжобласного транспорту.

Предмет дослідження – web-додаток, його функціонування, робота з даними та алгоритми розрахунку залежно від отримуваних даних.

Кваліфікаційна робота бакалавра складається зі змісту, вступу, основної частини, яка включає три розділи, висновків та списку використаних джерел. Всього 71 сторінку.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ (INTERNET OF THINGS, IOT), GPS ВІДСТЕЖЕННЯ, TRACKING SYSTEM, LOCATION TRACKING, GPS SYSTEM, VEHICLE TRACKING SYSTEM.

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technologies

Department of Information Systems and Technologies

Educational Program "Software Technologies of the Internet of Things"

Qualification work of master Polina MISHYNA.

Work topic: "Analysis and control system of interregional regular transport movement".

The purpose development of a system for collecting and analyzing data of the movement of scheduled transport, design and software implementation of the system with the possibility of preliminary calculation of the time of arrival of the vehicle to each of the stops.

The object of research system of fixing and calculating travel time for interregional transport.

The subject of research web-application, its functioning, work with data and calculation algorithms depending on the received data.

The qualification work consists of the content, introduction, main part, which includes three sections, conclusions and a list of sources used.

Total 71 pages.

KEY WORDS: INTERNET OF THINGS, IOT, GPS, TRACKING SYSTEM, LOCATION TRACKING, GPS SYSTEM, VEHICLE TRACKING SYSTEM.

ЗМІСТ

ВСТУП 7
1. АНАЛІЗ ТА ОПИС СФЕР ЗАСТОСУВАННЯ СИСТЕМИ 10
1.1 Аналіз предметної області 13
1.2 Дослідження типів систем відстеження транспортного засобу 14
1.3 Огляд аналогів 16
1.4 Вимоги до складових системи 21
Висновки до розділу 23
2. ПРОЕКТУВАННЯ РОБОТИ СИСТЕМИ 24
2.1 Створення контекстної та фізичної моделі БД 24
2.2 Програмні засоби, що використовуються 28
2.2.1 Flask 28
2.2.2 СУБД 29
2.2.3 MongoDB 30
2.2.4 API & Webhook 30
2.3 Апаратне забезпечення 31
2.3.1 Обчислювальний пристрій 31
2.3.2 GPS-трекер 33
2.3.3 Real-time clock модуль 35
2.4 Опис роботи системи та її алгоритм 37
Висновки до розділу 40
3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 41
3.1 Початок роботи 41
3.2 Робота з датчиками 45
3.3 Принцип розрахунку часу 55
Висновки до розділу 56
ВИСНОВКИ 57
ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 58
ДОДАТОК А 62
ДОДАТОК Б 68

ВСТУП

Кожний новий рік для будь-якого підприємства має основні два пункти: аналіз роботи минулого року та плани на прийдешній. Підприємці завжди у пошуках нових рішень, витків бізнесу, ідей та людей з новаторськими поглядами. У ІТ галузі під виразом “новаторський погляд” останні роки міцно закріпилася галузь IoT (англ. Internet of Things). Наразі немає меж для розвитку, вивчення та інтеграції цього напрямку. IoT – це автоматизація звичних для нас у повсякденному житті процесів. Вона охоплює як і апаратну частину рішень, так і програмну.

Сьогодення неможливо уявити без Інтернету, мережі, яка все більш і більш поглинає світ, спрощуючи життя у будь-яких питаннях, що нас цікавлять. Саме Інтернет є головною складовою, рушійною силою IoT рішень, як це і зрозуміло з назви цієї концепції. Аналізуючи багато джерел, вони вказують не маленькі цифри працюючих IoT систем на сьогодні: за 2021 рік кількість підключених пристроїв досягла 31 мільярду, а до 2030 року очікується більш як 125 мільярдів, але звичайно ці цифри кожний прогнозує та бачить по різному. Зараз починає своє розповсюдження та активну роботу технологія 5G, яка сприятиме ще більшому росту різноманітних IoT рішень у повсякденності.

Особливо примітним стрибок IoT галузі виявився приблизно у 2015-2016 роках. Неможливо не згадати про систему супутникового інтернету Starlink від Ілона Маска, який ніби диктує тренди у технологічних сферах. Можливо з цією новиною і почалася активна роботу у створенні розумних систем та пристроїв, до яких у майбутньому буде доступ з будь-якої точки світу.

Фахівці з багатьох галузей вже не можуть уявити звичну їм роботу без пристроїв та Інтернет мережі. Також, вони підтвердять, що існує ще дуже широкий спектр можливостей для аналізу та інтеграції новаторських рішень, для вирішення яких і був вигаданий (започаткувався, створене рішення/ідея) IoT. Різноманітність галузей, у яких використовується IoT вражає: метеорологія,

сільське господарство, будівництво, медицина, логістика роботи міст, космонавтика, комунальні підприємства та багато інших галузей.

Актуальність теми. Мандрівки у місті, країні та світом доступні для будь-кого будь-яким видом транспортного засобу. На сьогодні подорожі стали також невід'ємною частиною життя. Чим більше розвиваються міста, тим більше люди зацікавлені їх відвідати, що і надає змогу розширювати технічні можливості і в сфері пасажиро перевезення. Особливо у теперішній час, коли 2022 рік для України виявився занадто важким, на приємне диво патріотизм суспільства та зацікавленість країною та звичаями надзвичайно виросла. Не мала кількість людей вже планують подорожі областями України, щоб подорожувати та побачити щось нове. Звичайно значна кількість може дозволити собі подорож власною автівкою, але так само і громадський транспорт буде користуватися попитом, як економічний варіант.

Дана галузь не дуже розвинута в Україні в технічному плані, а саме відслідковування міжобласного транспорту. Пасажири спираються тільки на знання розкладу руху і досвід знайомих чи батьків про час, що необхідний для подолання певного проміжку маршруту. Це незручно, особливо для дуже довгих і тривалих маршрутів країною, тим паче для сьогоднішньої молоді, у якої життя завжди у русі та розписано ледве не щохвилинно. Також, на сьогодні треба враховувати часом пошкоджену та, навіть, знищену інфраструктуру на шляху. Міжобласні рейси мають звичку змушувати пасажирів чекати не хвилинами, а навіть годинами, так як на такі довготривалі подорожі зазвичай не передбачений транспорт у кожні декілька хвилин, як у містах, а трекінг такого виду транспорту наразі не існує та неможливий, так як не у всіх частинах областей присутній не те, що 2G, а просто банально зв'язок.

Аналізуючи численні галузі та поведінку населення (потенційних клієнтів), а також актуальність описаного спостереження вище, **метою роботи** було обрано дослідження транспортної галузі, а саме, створення аналітично-розрахункової IoT системи для контролю переміщення пасажирського

міжобласного транспорту. Дана система має можливість аналізувати час, витрачений на переміщення між кінцевими точками.

Метою дипломної роботи є розробка системи збору та аналізу даних про рух рейсового транспорту, проектування і програмна реалізація системи з можливістю попереднього розрахунку часу прибуття транспортного засобу до кожної з зупинок.

Об’єкт дослідження – система фіксації та розрахунку часу на подорож для міжобласного транспорту.

Предмет дослідження – web-додаток, його функціонування, робота з даними та алгоритми розрахунку залежно від отримуваних даних.

1. АНАЛІЗ ТА ОПИС СФЕР ЗАСТОСУВАННЯ СИСТЕМИ

Можливість відслідковування переміщення будь-чого та будь-кого набагато полегшило сьогодення для великої кількості людей. Принцип відстеження об’єктів таким, яким ми його зараз знаємо, було започатковано з винаходом Інтернету, коли можливість обміну інформацією стала значно швидшою та доступною без обмежень по всій земній кулі.[1] Згодом була розроблена та запущена система глобального позиціонування – **GPS** – працює за допомогою супутникових систем навігації. GPS відкрив багато різноманітних можливостей: вимірювання відстані, часу та визначення місцезнаходження у всесвітній системі координат WGS 84. Невдовзі ці технології почали використовувати і звичайні люди за допомогою своїх мобільних пристроїв, у які почали впроваджувати за замовчуванням GPS-приймачі.

Не зважаючи на те, що система GPS була розроблена лише для військових цілей, сьогодні ця технологія розповсюдилась на велику кількість галузей, кожна з яких має різні вимоги та потреби у використанні.

Наразі використання GPS для звичайних користувачів поєднує супутникову систему моніторингу об’єкту та картографію. Користувачі орієнтуються за допомогою візуалізованих карт місцевості, так як розуміння широт потребує додаткових знань та займає час для розуміння, тоді, коли час має велике значення у відстеженні. Система моніторингу рухомих об’єктів побудована з використанням спеціального обладнання, технологій стільникового або радіозв'язку, обчислювальної техніки, цифрових карт та GNSS-трекерів – приладів прийому-передачі даних для супутникового контролю переміщення об’єкту (див. рис.1.1). GNSS-трекери використовують GPS для чіткого визначення місця розташування.

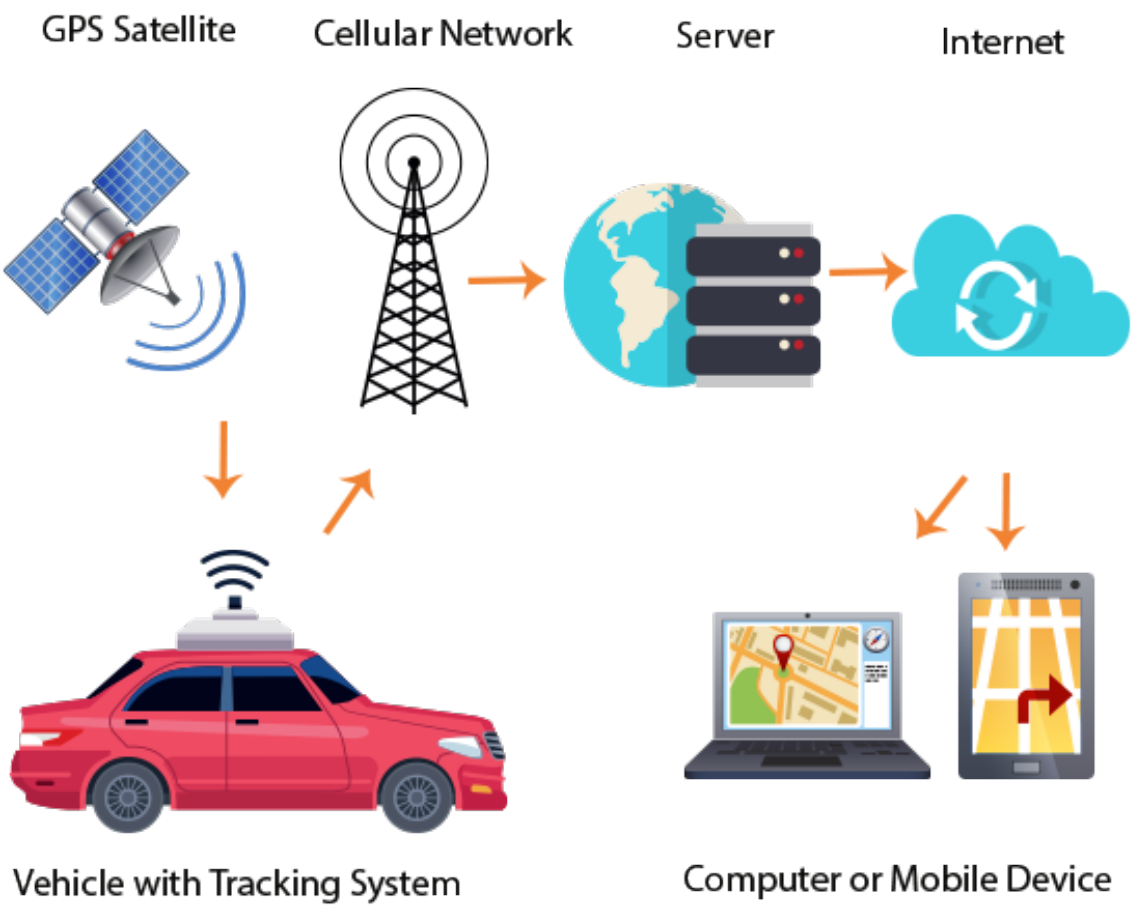


Рисунок 1.1 Відображення роботи GPS систем.

Основними задачами відстеження транспорту на основі GPS є:

- Моніторинг – включає визначення координат місця розташування транспортного засобу, його напрямку, швидкості. Система супутникового відстеження особливо корисна для навігації в незнайомих місцях.
- Контроль виконання графіків переміщення – ведення обліку переміщення транспорту та виконання завдань.
- Збір статистики для аналізу та оптимізації маршрутів – За допомогою зібраних даних про вже пройдені маршрути можна розрахувати вартість поїздки та витрачений час, це використовується для створення подібних маршрутів, які потребують менших витрат.
- Безпека – з технологією стеження власник має можливість знайти транспортний засіб, якщо його викрали.
- Ідентифікація – транспортний засіб має номер, пов’язаний з конкретним водієм або власником.

- Додатковий збір даних – у цей пункт входять дані про паливо, витрачене на маршруті, контроль заправки і злиття, контроль температури двигуна та інших вузлів транспортного засобу.

Звичайно існують і недоліки системи GPS. Перш за все – неточність. Нижче (табл.1.1) наведено складові, що впливають на похибку одного супутника при вимірюванні дальності руху об'єктів.[2]

Таблиця 1.1 – Фактори, що впливають на точність розрахунків GPS-систем.

Джерело похибки	Середньоквадратичне значення похибки, м
1	2
Нестабільність роботи генератора	6,5
Затримка у бортовій апаратурі	1,0
Невизначеність просторового стану супутника	2,0
Інші похибки космічного сегменту	1,0
Неточність ефемерид	8,2
Інші похибки наземного сегменту	1,8
Іоносферна затримка	4,5
Тропосферна затримка	3,9
Шумова помилка приймача	2,9
Багатопроменевість	2,4
Інші похибки сегменту користувача	1,0
Сумарна похибка	13,1

Також, загальним недоліком використання будь-якої радіонавігаційної системи є те, що за певних умов сигнал може не доходити до приймача або приходити зі значними спотвореннями чи затримками. Наприклад, практично неможливо визначити своє точне місцезнаходження у власному житлі, всередині залізобетонної будівлі, у підвалі або тунелі, навіть за допомогою професійних геодезичних приймачів. Так як робоча частота GPS лежить в дециметровому діапазоні радіохвиль, рівень сигналу від супутників може серйозно знизитися під щільним листям дерев або через велику хмарність.

1.1 Аналіз предметної області

Системи відстеження транспортних засобів зазвичай використовуються операторами для управління наявними транспортними засобами. Таке управління включає в себе наступні функції: відстеження, маршрутизація, диспетчеризація, бортова інформація та безпека. Деякі системи відстеження транспортних засобів постачаються в комплекті з апаратним та програмним забезпеченнями або взаємодіють з ними. Також цю технологію використовують у сферах комерційних автопарків, комунальних підприємств громадського транспорту для низки цілей, включаючи моніторинг дотримання розкладу автобусів.

На сьогодні чимало країн вже використовують системи відстеження транспортних засобів на основі GPS, більшість з них запрограмовані лише на автоматичні сповіщення про зупинки.[3] Дані, зібрані під час руху транспортного засобу своїм маршрутом, постійно надходять у комп'ютерну програму, яка порівнює фактичне місцезнаходження та час транспортного засобу з його розкладом і, у свою чергу, виводить для водія часто оновлюваний дисплей, який повідомляє, як рано чи пізніше він буде у певний час, що потенційно сприяє більш чіткому дотриманню опублікованого графіку. Такі програми також використовуються для надання пасажирам інформації в режимі реального часу про час очікування наступного автобуса чи іншого транспортного засобу на певну зупинку на основі фактичного руху транспортних засобів поблизу, а не просто надання інформації про розклад, час наступного прибуття. Транспортні системи, які надають таку інформацію, призначають кожній зупинці унікальний номер, а пасажири, які очікують, можуть отримати інформацію, ввівши номер зупинки в автоматизовану систему або додаток на веб-сайті системи громадського транспорту. Деякі транспортні компанії надають на своєму веб-сайті віртуальну карту, яка показує поточне місцезнаходження автобусів, що курсують на кожному маршруті, для інформації клієнтів, тоді як інші надають таку інформацію лише диспетчерам або іншим співробітникам.

1.2 Дослідження типів систем відстеження транспортного засобу

Як зазначалося вище – системи GPS активно використовуються як і звичайні люди у повсякденному житті, так і підприємства, яким це необхідно. Система GPS складається з трьох сегментів: космічного, управляючого та користувацького. З боку користувача необхідно мати GPS-трекер, або ж мобільний пристрій з таким вбудованим компонентом.

Для відстеження транспортних засобів зазвичай використовують перший варіант – окремий GPS-трекер – це може бути як і невеликий пристрій, так і компонент, у вигляді плати, для цілої системи, що інтегрується у автівку. Такий пристрій надає повну інформацію про точне місцезнаходження об’єкту, також є можливість відстеження руху у режимі реального часу.[4]

Існує два типи роботи систем відстеження GPS: **активний** та **пасивний**. **Активний** – як вже згадувалося, це відстеження руху об’єкту у режимі реального часу. При використанні активного GPS-трекера користувач може відстежувати кожен рух відстежуваної людини або об’єкта. Активні пристрої GPS дозволяють користувачеві переглядати швидкість, місцезнаходження та інші деталі відстеження з будь-якого місця незабаром після того, як пристрій буде розгорнуто. Такі системи регулярно надсилають дані до центральної бази даних через модем.

Пасивне відстеження – це зберігання даних про рух через певні події. Цей тип відстеження часто реалізується у вигляді звітів про те, де був об’єкт годину, три чи дванадцять годин тому. Такі дані можуть зберігатися в системі трекера або на карті пам’яті, якщо вони надані пристроєм. Інформацію можна запитувати в певний час або періодично під час поїздки.[5] Також пасивні системи передбачають і відстеження в режимі реального часу, інформація про яке автоматично надсилається на центральний трекінговий портал відповідно до налаштувань оновлення даних. Це пасивний тип системи, який широко використовується в комерційних підприємствах і програмах для оптимізації роботи автопарків.[6]

Таблиця 1.2 - Різниця між системами активного та пасивного відстеження.

Параметри	Активне GPS відстеження	Пасивне GPS відстеження
1	2	3
Використання пам’яті для даних	Займає відносно небагато місця, так як дані постійно оновлюються.	Може займати великі об’єми пам’яті, так як дані зберігаються на пристрої.
Цінова категорія	Дорогі	Дешеві
Програмне забезпечення	Мобільний додаток, або веб версія, яка сумісна з GPS-трекером.	Наявність додатку не обов’язкова завдяки налаштуванню роботи сценаріїв пристрою, що відстежує рухомий об’єкт.
Підключення до хмарного середовища	Обов’язкове для забезпечення роботи відстеження у режимі реального часу.	Не є обов’язковим.
Точність	Точне, з можливими затримками чи відхиленнями.	Точне, за наявності додаткових параметрів.
Час реакції	Негайна реакція, у режимі реального часу, але також з можливими затримками.	Залежить від комплектації системи. Частіше дані стають доступними після отримання пристрою зі збереженими даними та після їх завантаження у ПК.

Набуває популярності новий вид системи – **гібридний**. У такій системі дані зазвичай відстежуються з меншою інтенсивністю, ніж у активній. У разі надзвичайної події, коли відстежуваний об’єкт виходить поза межі певної геозони пристрій GPS автоматично перемикається у активний режим та надає оновлення про розташування об’єкту у реальному часі. **Гібридний** тип системи наразі дуже популярний для IoT рішень з такими перевагами:

- Використання мінімальної кількості супутників для визначення місцезнаходження.

- Забезпечує більш передбачувані й узгоджені дані про місцезнаходження, ніж інші рішення.
- Заощаджує приблизно 80% енергії, що використовується за допомогою стандартного визначення місцезнаходження GPS.
- Отримання даних про місце об’єкту можна отримати у будь-який час за зверненням до системи.

1.3 Огляд аналогів

Для створення інноваційних рішень перш за все необхідно розглянути ринок готових рішень за обраним направленням. Аналізуючи ринок, підприємство дізнається що саме вже активно використовується споживачами, як продукти працюють, на кого вони орієнтовані. Важливо не тільки переглянути, але й спробувати себе у якості споживача, щоб зрозуміти чого продуктам інших виробників може не вистачати, на що зорієнтуватися підприємству, щоб новий продукт став актуальним та популярним, випереджуючи наявні аналоги та створюючи конкурентоспроможний продукт.

Оскільки метою дипломної роботи є розробка саме програмного забезпечення – також особлива увага повинна приділятися і системним вимогам пристроїв. Аналіз необхідний не тільки аналогів продукту, але й технічного обладнання, з яким система буде працювати повною мірою. Звичайно визначити усі технічні характеристики обладнання може бути досить складно, за відсутністю договорів про співпрацю між підприємствами виробників, але з плином часу та розвитком системи такі проблеми вирішуються. Головною ціллю є розглянути основний функціонал продуктів, програмне забезпечення, і підтримувані технічні пристрої, що користуються попитом зараз.

ITlynx.

ТОВ «Ай Ті – Лінкс Сервіс» - підприємство, яке розробило ряд технічних рішень для надання послуг у сферах GPS-моніторингу, диспетчеризації та контролю використання палива.[8] Компанія наразі має два програмних продукти: програмно-апаратний комплекс «Інспектор» та автоматизація і

будівництво систем зрошення (рис.1.2). В даному випадку нас цікавить перший згаданий продукт та його основне технічне рішення – система GPS моніторингу «Інспектор».[9]

Які дані надає система супутникового моніторингу «Інспектор»:

- Маршрут руху.
- Швидкість пересування.
- Місця зупинок.
- Витрати палива.
- Робота різноманітних датчиків.

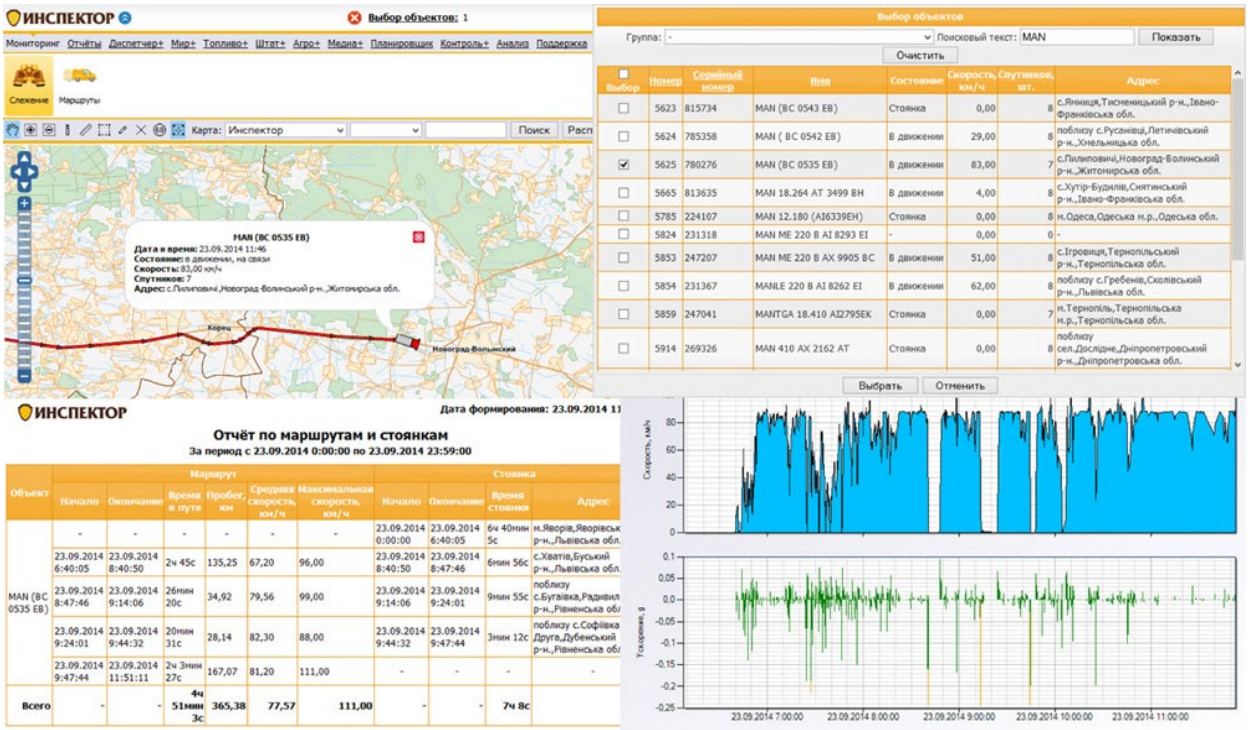


Рисунок 1.2 Зображення декількох вікон програми.

Дана система використовується виключно зі сторони адміністратора та диспетчера. «Інспектор» працює на пристроях власного виробництва. Система GPS-моніторингу має можливість інтегрування та злагодженої роботи з іншими рішеннями у програмно-апаратному комплексі «Інспектор», що надає ITlynx.

ProGps.

ProGPS – проект компанії IT INNOVATIONS, спрямований на оптимізацію роботи транспорту та надають наступні послуги: [10]

- Обладнання для GPS моніторингу – GPS-трекери.
- Системи контролю пального – датчики рівня пального та проточні витратоміри.
- Периферійне обладнання для контролю і управління додатковими механізмами.

Компанія є розробником особистого програмного продукту Fleetix (рис.1.3), яке може бути доопрацьоване під будь-які вимоги замовника, інтегровано зі сторонніми додатками.

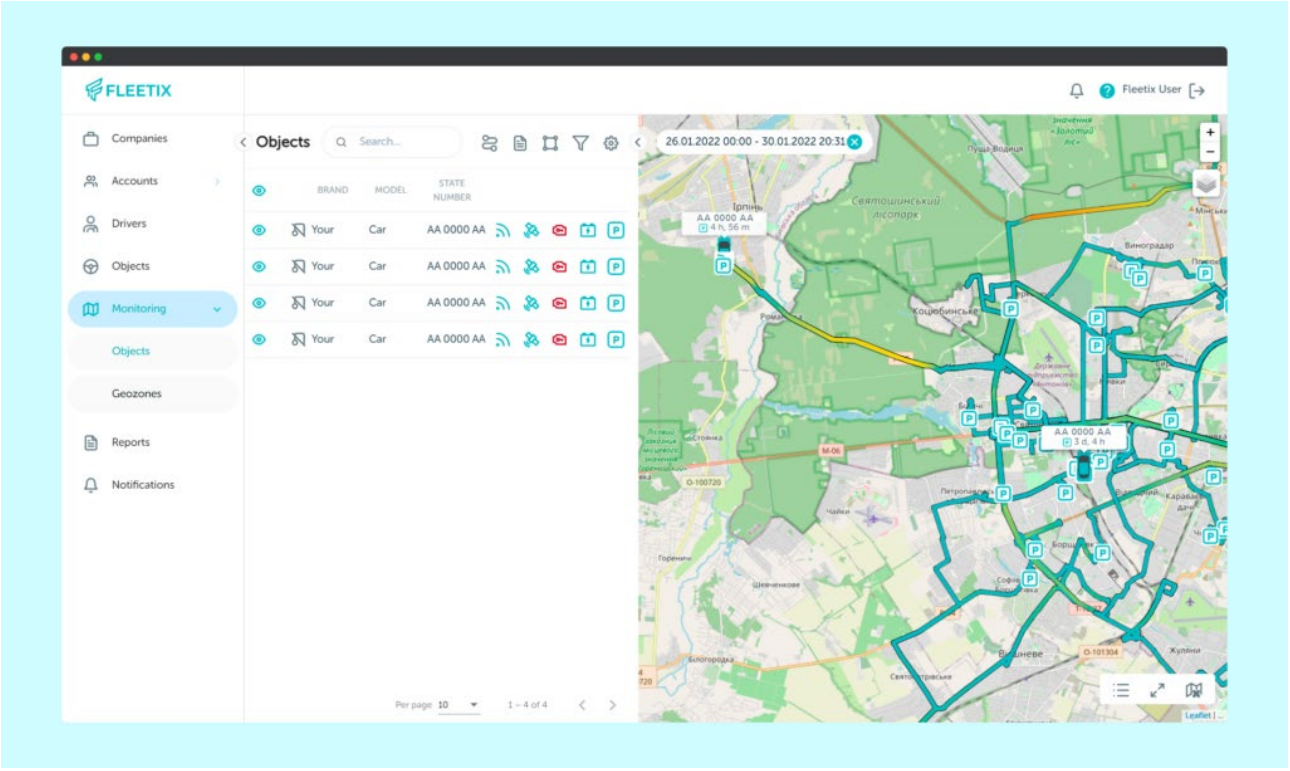


Рисунок 1.3 Вигляд програми Fleetix для адміністратора.

Команда розробників створила широкий спектр пакетів по впровадженню систем моніторингу і контролю для всіх сфер бізнесу, наприклад:

1) Агробізнес – наскрізний облік палива, створення нарядів для проведення робіт, автоматичний облік посівних площ та робочого часу.
2) Будівництво - комплексний облік робочого часу водіїв і фіксація мотогодин і використаного пального для будь-якого типу транспорту.
3) Пасажирський транспорт – повноцінна система диспетчеризації громадського транспорту і інформація про місцезнаходження транспорту для пасажирів.
4) Комунальний транспорт – контроль якості виконання завдань з обслуговування міста для будь-якої техніки і надання звітності жителям.

Компанія має широкий вибір технічного обладнання з яким працюють для будь-якого транспорту.[11] В залежності від обраних послуг системи, надається вибір якісних комплектуючих для подальшої роботи. Загалом система виконує наступні поставлені задачі:

- Онлайн моніторинг транспорту.
- Фіксація зливів і заправок пального.
- Облік фактичних витрат пального.
- Ідентифікація водіїв.
- Облік мотогодин будь-якого типу техніки.
- Фіксація замісу і вивантаження бетоновозів.
- Фотомоніторінг.
- Підключення спецтехніки до CAN-шини.
- Контроль геозон.
- Відстеження швидкісних режимів.
- Фіксація вивантаження для самоскида.
- Інтеграція з системами 1С і іншими обліковими системам.

NaviTrack.

Компанія NaviTrack є виробником обладнання, розробником програмного забезпечення та оператором ринку систем у сфері моніторингу, віддаленого

контролю та раціонального використання стаціонарних та рухомих об'єктів, різного роду палива, рідких середовищ та рідин на базі супутникових навігаційних систем GPS.[12] Компанія пропонує різні варіанти систем GPS моніторингу та контролю, де присутня можливість самостійно обирати комплектацію.

На даний момент у NaviTrack є три програмні продукти:

- Online.navitrack – програмний продукт для моніторингу рухомих об'єктів у режимі реального часу рухомих об'єктів, на яких встановлено обладнання для моніторингу (рис.1.4). Включає в себе наступні модулі: мапа, спостереження, графіки, геозони, водії, звітність, тривожна консоль.
- Fcs.navitrack – програмний продукт обліку руху палива у резервуарах.
- Мобільні програми Navitrack – мобільні програми для платформ IOS та Android для моніторингу рухомих об'єктів у режимі реального часу.

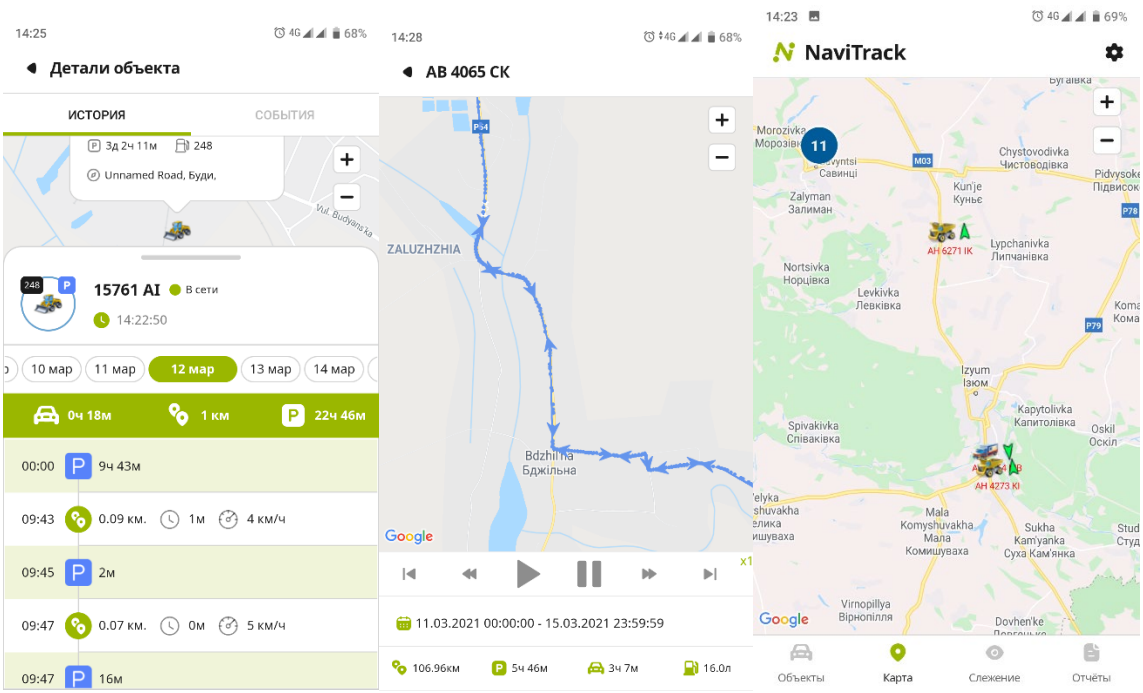


Рисунок 1.4 Вигляд мобільного додатку.

Основні види діяльності компанії полягають у створенні універсальних рішень та розробці багатофункціонального обладнання на базі супутникових

технологій. NaviTrack використовують власні технології для забезпечення роботи систем автоматизованого GPS моніторингу за допомогою власно розроблених платформ та програмного забезпечення.

1.4 Вимоги до складових системи

Згідно з поставленими цілями, результатом роботи буде готовий додаток для моніторингу переміщення міжобласного транспорту. Компоненти системи поділяються на *програмне* та *апаратне* забезпечення. Для коректної роботи додатку є перелік мінімально необхідних компонентів системи.

До програмних відносяться:

1) Бекенд (англ. back-end) – відноситься до розробки на стороні сервера. Він фокусується на базах даних, сценаріях, архітектурі веб-сайту. Містить дії за кадром, які відбуваються під час виконання будь-якої дії на веб-сайті. Це може бути вхід в обліковий запис або покупка в інтернет-магазині. Код, написаний розробниками серверної частини, допомагає браузерам спілкуватися з інформацією бази даних.

   Бекенд розробка зосереджується на:

   - Мови програмування та сценаріїв, такі як PHP, Python та C#.
   - Автоматизовані рамки тестування.
   - Масштабованість та доступність мережі.
   - Управління базами даних і перетворення даних.
   - Безпека та методи резервного копіювання даних.

2) Бази даних – це організований збір інформації, який структурується, або даних, які зберігаються у електронному форматі у комп’ютерній системі. БД контролюється системою управління базами даних (СУБД). Дані в поширених типах БД, які актуальні на сьогодні, зазвичай моделюються у вигляді рядків та стовпців в таблицях, щоб організувати обробку та запити даних більш ефективними. З такими даними легко взаємодіяти: змінювати, оновлювати, створювати та контролювати. На

теперішній час більшість БД використовують мову структурованих запитів (англ. SQL – structured query language).

3) Фронтенд (англ. front-end) – те, що бачить користувач, іншими словами веб-сторінка, з якою клієнт взаємодіє. Фронт-енд розробка охоплює HTML, CSS, JavaScript та інше кодування веб-сайту або програми, яке дозволяє користувачеві бачити його та взаємодіяти з ним. Робота фронтенд розробника включає кодування програм і веб-сайтів, а потім налаштування їх, щоб вони мали привабливий вигляд та виконували конкретні завдання. Хоча розробники, як правило, мають більш технічні та аналітичні направлення думок, розробнику інтерфейсу також потрібна певна креативність, щоб створити приємний для ока сайт.

   Фронтенд включає в себе концепції та компоненти:

   - Мови дизайну та розмітки, такі як HTML, CSS та JavaScript.
   - Пошукова оптимізація (SEO).
   - Тестування зручності та доступності.
   - Інструменти графічного дизайну та редагування зображень.
   - Веб-продуктивність і сумісність з браузером.

Мінімально необхідна апаратна частина системи:

1) Обчислювальний пристрій – це може бути контролер або мікрокомп’ютер, підійде будь-який пристрій, що містить у собі з модулем Wi-Fi або Bluetooth, або має можливість інтеграції із ними.
2) Подібний мікроконтролер, як зазначалося вище, потрібний буде і для рухомого об’єкту, але матиме невелику відмінність у вигляді GPS-модуля, що буде використовуватись для більш точного трекінгу транспорту на маршруті.
3) Система живлення – у ролі генератора може бути powerbank або ж сам прилад може бути вбудований в авто, яке буде надавати енергію.

4) Real-time clock модуль (годинник реального часу) – електронна схема, призначена для обліку хронометричних даних, являє собою систему з автономного джерела живлення та пристрою, що враховує.

Висновки до розділу

В даному розділі вивчено поняття та принцип роботи GPS систем, з чого вони складаються та як саме вони застосовуються у теперішній час в галузі відстеження транспорту. Розглянуто також недоліки у роботі системи, фактори, що спричиняють появі похибок та наведено приблизні цифри погрішностей – наведено у табл. 1.1.

Досліджені типи систем відстеження: *активний*, *пасивний* та *гібридний* – їх особливості у роботі, складові системи, що необхідні для. У табл. 1.2 наведено порівняльну характеристику між активним типом системи та пасивним.

Проаналізовано існуючі рішення, їх функціонал, розглянуті готові системи рішень. Оглянуті особливості кожної з систем, що пропонують виробники, та їх інтерфейси.

Наведено приклад мінімально необхідних складових власної системи для забезпечення її роботи. Система аналізу та контролю рейсового транспорту складається з апаратної та програмної частин. Мінімальне апаратне забезпечення потребує наявність обчислювального пристрою, систем живлення, GPS-трекеру та карти пам’яті. Програмне забезпечення складається з розробки бекенду та фронтенду, а також створення бази даних.

2. ПРОЕКТУВАННЯ РОБОТИ СИСТЕМИ

2.1 Створення контекстної та фізичної моделі БД

Сучасні системи займають неймовірно великий обсяг пам’яті, які потребують адміністрування та управління. З розвитком системи збільшуються і обсяги інформації, що збирається, аналізується та зберігається. Винахід та розвиток нових систем вимагає наявність певних навичок в організації управління отриманими даними. Обов’язковим є передбачення подальшого росту системи та накопичення більших обсягів інформації. Уся така інформація заноситься у бази даних, які передбачають її збереження та використання. Тому важливим першим кроком при проектуванні та реалізації будь-якого проекту є саме побудова бази даних, її нормалізація та планування її подальшого масштабування.

Для побудови логічної моделі даних було обрано схема «сутність-зв’язок». ER-діаграми (англ. Entity Relationshi) – це різновид блок-схем, де мають місце «сутності» (об’єкти, люди, системи і тому подібне), які пов’язані між собою всередині системи. Такий вид діаграм частіше за все використовується для проектування та відкладення реляційних баз даних у різноманітних сферах, такі як освітня, дослідницька чи розробка програмного забезпечення. ER-моделі прості для розуміння та у побудові, використовують простий набір символів, включаючи прямокутники, ромби, кола, з’єднувальні лінії для відображення зв’язків та атрибутів.

Основна робота проводиться над визначенням взаємин сутності та зв’язків. Сутність позначається прямокутником (ще має назву *іменник*), зв’язок – ромбом (*дієслово*), атрибути – еліпс.

У побудованій схемі для проекту сутностями є «Автобус», «Маршрут», «Зупинка» та «Класифікатор автобусів». Кожна сутність має певну кількість атрибутів. Ці сутності пов’язані між собою зв’язками. Сутність «Автобус» має атрибути: Державний номер, Тип, Середня швидкість, Кількість місць, Чи в

дорозі та Маршрут – усі ці атрибути використовуються для опису кожного автобусу, а також, характеристик для подальшого аналізу та планування змін у системі та у майбутній роботі. Сутність «Класифікатор автобусів» має наступні атрибути: Марка, Модель, Максимальний пробіг, Норма витрат – для детального опису характеристик кожного з автобусів. Сутність «Маршрут» вміщує в себе атрибути: Номер маршруту, Тип, Кількість зупинок, Відстань, Чи в дорозі – для організації та відстеження наявних автобусів на певному маршруті. Сутність «Зупинка» складається з Id зупинок та Назви зупинок – для їх ідентифікації на маршруті.

Кожна з сутностей має зв’язки одна з одною, що доповнюють та надають чітке розуміння планування майбутньої бази даних. Таким чином сутність «Автобус» має зв’язок, а точніше класифікується, за допомогою сутності «Класифікатор автобусів». Сутність «Маршрут» має зв’язок, складається з «Зупинок». Основними ж складовими бази даних є саме сутності «Автобус» та «Маршрут», які пов’язані між собою зв’язком, що позначає рух.

Повна модель «сутність-зв’язок» відображена на рис. 2.1.

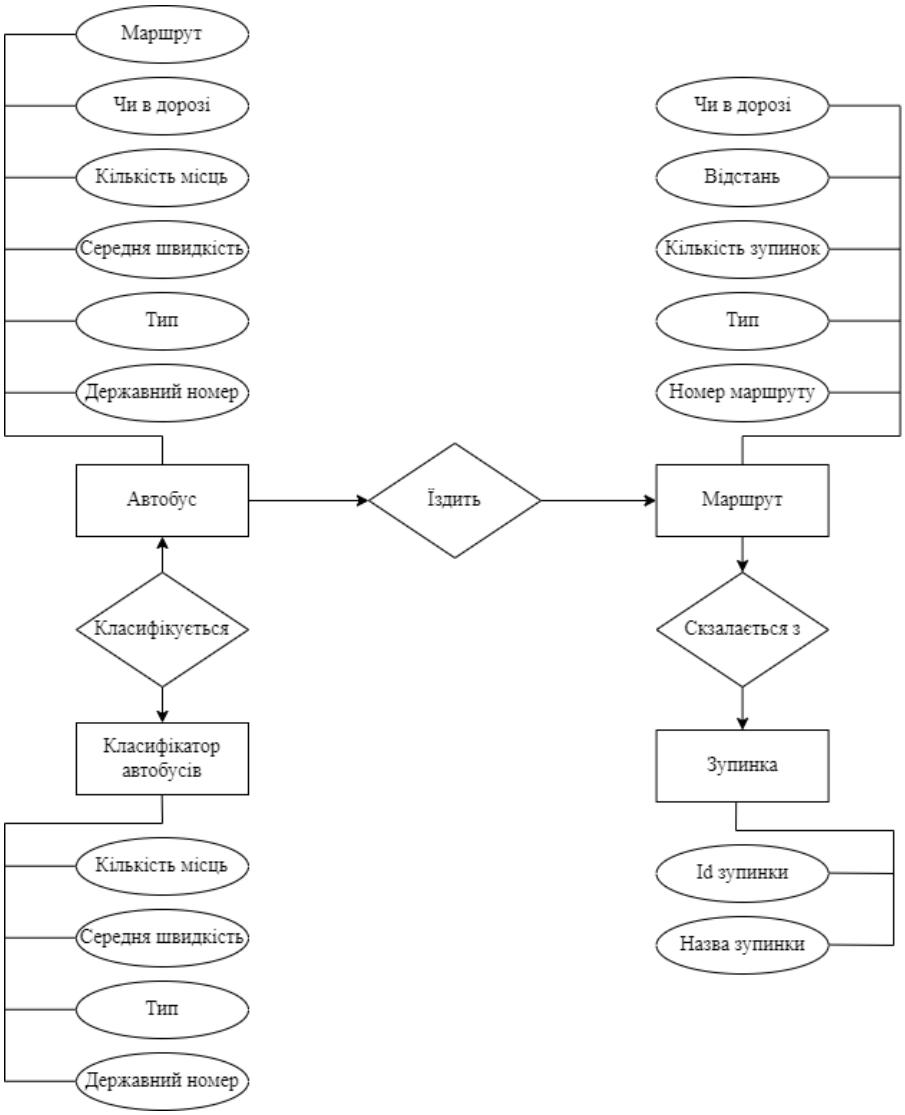


Рисунок 2.1 ER-діаграма бази даних.

Завдяки попередній побудові ER-діаграм, наступним кроком буде відображення її у вигляді таблиць, з яких і складається баз даних. Фізична модель даних візуально представляє структуру даних та їх тип.

Фізична модель (див. рис.2.2) містить у собі 4 таблиці:

1. Logs ().
2. Bus (Автобус).
3. Route (Маршрут).
4. Bus_stop (Зупинка).

Кожна таблиця складається з полів, що являють собою запис наборів значень певного типу. Таким чином:

Таблиця logs складається з:

- id – тип даних integer.
- stop_id – тип даних integer.
- route_id – тип даних integer.
- bus_id – тип даних integer.
- arrive_time – тип даних datetime.

Таблиця bus:

- id – тип даних integer.
- route_id – тип даних integer.
- speed – тип даних integer.
- current stop– тип даних integer.
- syatus – тип даних bool.

Таблиця route:

- id – тип даних integer.
- name– тип даних string.

Таблиця bus_stop:

- id – тип даних integer.
- route_id – тип даних integer.
- name– тип даних string.
- type– тип даних string.
- prev_bus_stop_id – тип даних int.
- next_bus_stop_id – тип даних int.
- waiting_time_in_minutes – тип даних int.
- distance_to_next_stop – тип даних int.

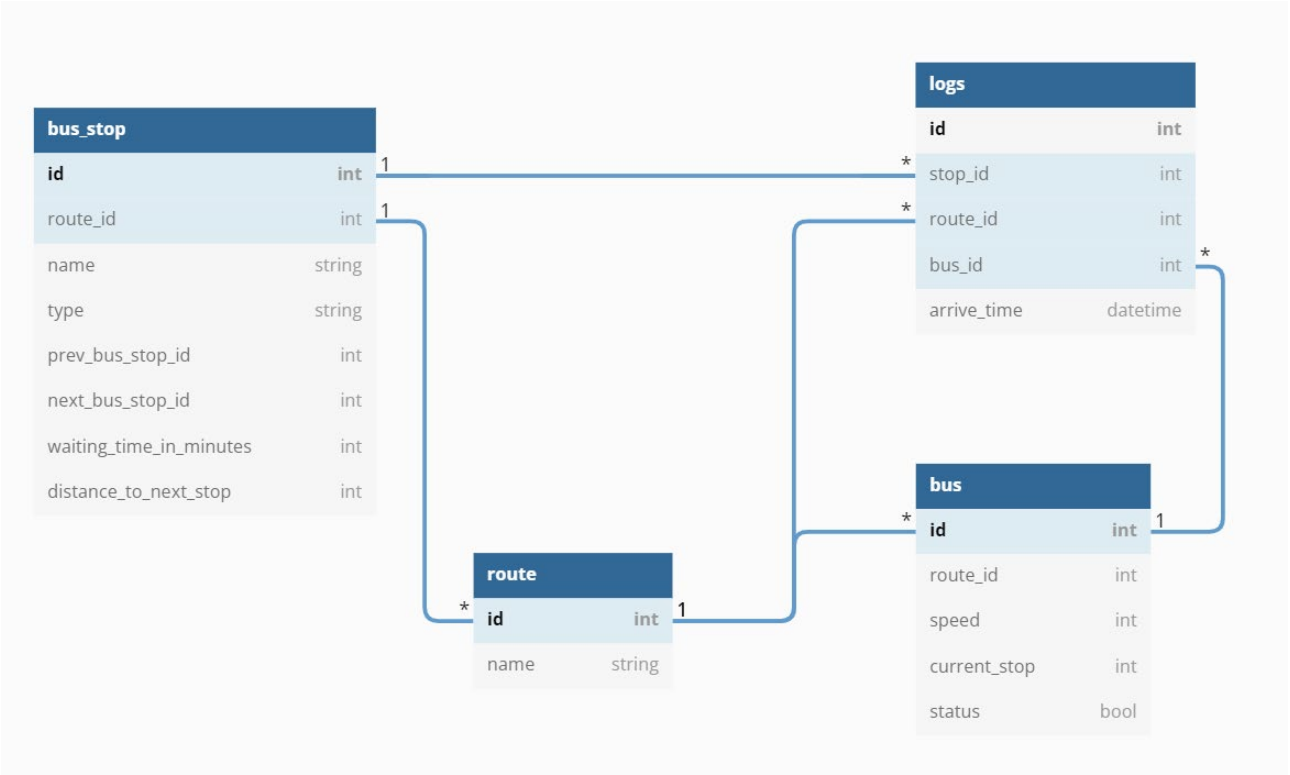


Рисунок 2.2 Фізична модель БД системи аналізу та контролю переміщення транспорту

2.2 Програмні засоби, що використовуються

Розглянувши у попередньому розділі (див.2.5) загальну інформацію про необхідні складові системи, в цьому розділі буде описано конкретні програмні та апаратні елементи системи, що будуть використовуватися у роботі.

2.2.1 Flask

Flask – це веб-фреймворк, написаний на мові Python.[20] Класифікується як мікровеб-фреймворк, оскільки надає інструменти, бібліотеки та технології, які дозволяють створювати веб-додаток. Мікрофреймворк, як правило, є фреймворком, який майже не залежить від зовнішніх бібліотек. Плюси використання цієї технології полягають у тому, що фреймворк легкий, є мало залежностей для оновлення та спостереження за помилками безпеки.[21] Деякі особливості та функції, які роблять Flask ідеальною платформою для розробки веб-додатків:

- Надає розробнику повний контроль над рішеннями щодо створення програми на етапі розробки (впровадження).

- Прості та гнучкі конфігурації.
- Забезпечує інтегровану підтримку модульного тестування.
- Flask використовує Jinja2 у якості системи шаблонізації.

На відміну від Django, який є повноцінним веб-фреймворком, Flask – легкий фреймворк, поставляється з деякими стандартними функціями і дозволяє розробникам додавати будь-яку кількість бібліотек або плагінів для розширення. Просте освоєння для новачків.[22]

2.2.2 СУБД

Система управління базами даних (СУБД) – це програмне забезпечення, яке взаємодіє з кінцевим користувачем, додатками та самою базою даних для збору й аналізу даних. Програмне забезпечення СУБД додатково охоплює основні засоби, надані для адміністрування бази даних. Загальну суму бази даних, СУБД і пов'язаних програм можна назвати системою баз даних. БД відноситься до набору пов'язаних даних та способу їх організації. Саме СУБД забезпечує доступ до цих даних.

Існуючі СУБД забезпечують різні функції, які дозволяють керувати базою даних та її даними, які можна класифікувати на чотири основні функціональні групи, які також мають назву CRUD-операції:[23] [24]

- Визначення даних – створення, зміна та видалення визначень, які визначають організацію даних.
- Оновлення – вставка, зміна та видалення фактичних даних.
- Вилучення – надання інформації у формі, яку можна безпосередньо використовувати або для подальшої обробки іншими програмами. Отримані дані можуть бути доступні у формі, яка в основному зберігається в базі даних, або в новій формі, отриманій шляхом зміни або об'єднання наявних даних із бази даних.
- Адміністрування – Реєстрація та моніторинг користувачів, забезпечення безпеки даних, моніторинг продуктивності, підтримка

цілісності даних, управління паралельністю та відновлення інформації, яка була пошкоджена якоюсь подією, наприклад, несподіваним збоєм системи. Вибір СУБД базується на вимогах до системи.

2.2.3 MongoDB

MongoDB — це кросплатформна програма баз даних з відкритим вихідним кодом, орієнтована на документи. MongoDB, класифікована як програма баз даних NoSQL, використовує документи, подібні до JSON, з додатковими схемами.[25]

Особливості:[26]

- Кросплатформність – СУБД розроблена мовою програмування C++, тому легко інтегрується під будь-яку операційну систему (Windows, Linux, MacOS та інші).
- Формат даних – MongoDB використовує власний формат зберігання інформації.
- Документ – якщо реляційні БД використовують рядки, то MongoDB – документи, які зберігають значення і ключі.
- СУБД здійснює пошук за спеціальними запитами. Користувач може створити діапазонний запит та миттєво отримати відповідь.
- Індексація – технологія застосовується до будь-якого поля у документі на розсуд користувача. Проіндексована інформація обробляється швидше.
- MongoDB може використовуватися як хмарне рішення для кінцевого клієнта.

2.2.4 API & Webhook

API (англ. Application Programming Interface) – прикладний програмний інтерфейс. Простими словами це набір інструментів, які дозволяють комп’ютерній програмі взаємодіяти з іншою програмою (API).[27] API спрощує процес програмування при створенні додатку, абстрагуючи базову реалізацію та надаючи лише ті об’єкти та дії, які необхідні розробнику. Для зв’язку додатків

між собою розробнику немає потреби знати їх внутрішній пристрій. Завдяки API стає можливим отримувати всю необхідну інформацію (наприклад нові пости, новини користувачів) у сторонній додаток.

Webhook також використовується для того, щоб різні системи мали змогу обмінюватися інформацією між собою.[28] Але цей механізм має інший принцип роботи. Цей принцип полягає в тому, що як тільки якась подія сталася – про це відразу ж надсилається певне повідомлення. Якщо у випадку з API додаток, якому потрібне оновлення інформації, постійно відправляє запити, то у випадку використання.[29] Webhook додаток чекає певне повідомлення про подію.

2.3 Апаратне забезпечення

У попередньому розділі (див.2.5) було перераховано мінімальні вимоги до апаратної частини системи. В даному розділі приведено чіткі приклади пристроїв, що необхідні для забезпечення роботи системи.

2.3.1 Обчислювальний пристрій

Як вже було зазначено, обчислювальним пристроєм може бути будь-який мікроконтролер або комп’ютер з наявними модулями бездротового зв’язку. Прикладами таких контролерів можуть бути будь-які мікросхеми на базі ESP8266. Прикладами таких є:

- NodeMcu – платформа на основі ESP8266 (рис.2.3) для створення різних пристроїв Інтернету речей (IoT).[30] Модуль вміє відправляти та отримувати інформацію в локальну мережу або інтернет за допомогою Wi-Fi. Недорогий модуль часто використовується для створення систем розумного будинку або роботів Arduino, що керуються на відстані.

Рисунок 2.3 Вигляд NodeMcu ESP8266.

- Witty Cloud. Wi-Fi модуль ESP8266 (рис.2.4). Witty побудований на базі популярного чипу ESP8266-12 та призначений для створення пристроїв керування і збору інформації через Wi-Fi мережу.[31] Модульна конструкція дозволяє максимально гнучко формувати конфігурацію кінцевого пристрою. Як мінімальне оснащення пропонуються змонтовані на модулі датчик освітленості та RGB світлодіод. Живити модуль можна як через microUSB роз'єм (5В), так і безпосередньо від джерела живлення 3.3В.

Рисунок 2.4 Вигляд NodeMcu ESP8266.

- Мікроконтролер D1 mini. Плата WeMos D1 mini – це аналог плат Witty Cloud і NodeMCU v3 (рис.2.5), що має найменші серед них габарити і використовує модуль ESP8266 версії ESP-12F.[32] Для роботи WeMos D1 mini не потрібен зовнішній мікроконтролер або інший керуючий пристрій, так як крім Wi-Fi модуля ESP-12F вже

вбудований 32-бітний мікроконтролер з тактовою частотою 80 МГц, а також чіп флеш-пам'яті на 4МБ.

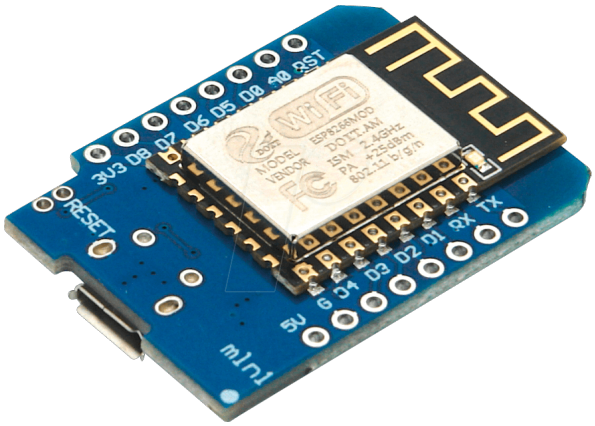


Рисунок 2.5 Вигляд D1 mini ESP8266.

- Raspberry Pi — це невеликий одноплатний комп’ютер, чиї інтегровані входи та виходи дозволяють застосовувати його до широкого кола проектів.[33] Вони варіюються від базових завдань (наприклад, написання тексту) до більш складних, таких як використання Pi (рис.2.6) як мультимедійного центру або сервера NAS. Комп’ютер також може бути використаний у сфері IoT (Internet of Things) або проектів домашньої автоматизації.

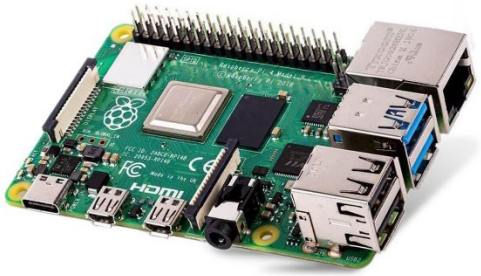


Рисунок 2.6 Вигляд Raspberry Pi 4 B.

2.3.2 GPS-трекер

GPS-трекер – це електронний пристрій, який визначає координати об'єкта. Але щоб місце розташування можна було подивитися на карті, потрібне спеціальне програмне забезпечення, яке отримує, обробляє та показує координати система GPS-моніторингу. За допомогою неї користувачі можуть

бачити розташування об'єктів у режим реального часу. Даний модуль влаштовується у плату, що знаходиться в автобусі.

- Trema GPS модуль ATGM336H – є навігаційним пристроєм (рис.2.7), що дозволяє визначити свої координати по широті, довготі та висоті.[35] Додатково модуль здатний визначити поточну дату, час, швидкість та напрямок пересування. Модуль отримує дані на основі інформації, що надходить із супутників навігаційних систем GPS. Він самостійно обробляє отриману інформацію та передає дані по шині UART у вигляді текстових повідомлень у форматі протоколу NMEA 0183, відрізняється низьким енергоспоживанням та високою чутливістю.

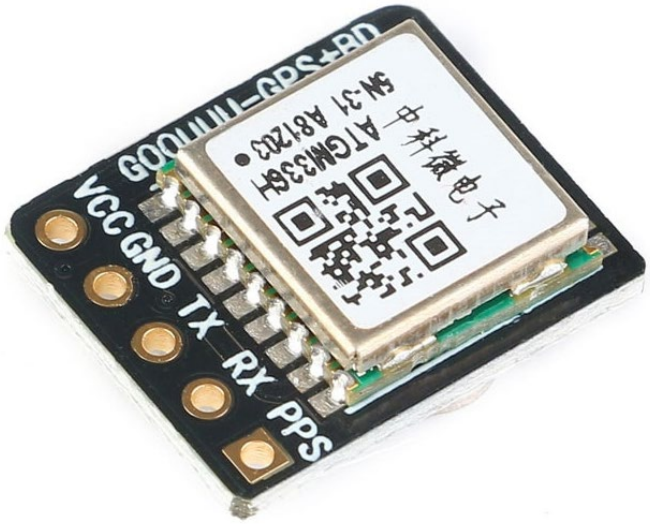


Рисунок 2.7 Вигляд Trema GPS модуль ATGM336H.

- Модуль GPS з антеною для Arduino APM2 Ublox NEO-6M (рис.2.8) – це модуль, який допоможе забезпечити повноцінний GPS зв'язок із пристроєм.[36] Ця модель сумісна з Arduino, Ardupilot та багатьма іншими мікроконтрольними системами. За допомогою цього пристрою обмін даними здійснюється через UART. APM2 Ublox NEO-6M відрізняється низьким енергоспоживанням, компактними розмірами та бюджетною вартістю. Ця модель має 50 каналів позиціонування з більш ніж 2 мільйонами ефективних кореляторів.

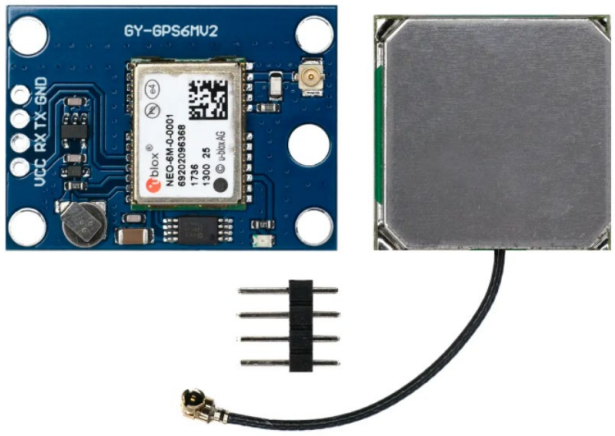


Рисунок 2.8 Вигляд модуля GPS з антеною для Arduino APM2 Ublox NEO-6M.

- GPS модуль GY-NEO6MV2 – автономний GPS-приймач (рис.2.9). Гнучкий та економічний, пропонує численні варіанти підключення в мініатюрі.[37] Модуль працює як з ПК, так і з багатьма платами різних виробників. Приймач пропонує користувачу не тільки потоне положення на мапі, але також і відображає додаткові дані, такі як швидкість, курс, висота та кількість видимих супутників.

Рисунок 2.9 Вигляд модуля GPS с антеною.

2.3.3 Real-time clock модуль

Годинник реального часу (RTC) — це електронний пристрій (найчастіше у вигляді інтегральної схеми), який вимірює хід часу.

Хоча стеження за часом можливе без RTC, його використання має ряд переваг:

1) Низьке енергоспоживання (важливо при функціонуванні від альтернативного джерела живлення).
2) Звільняє основну систему від навантаження та виконання важливих за часом завдань.
3) Часом точніший за інші методи.

RTC не слід плутати з обчисленнями в реальному часі, які мають трибуквену абревіатуру, але не мають прямого відношення до часу доби.

- Модуль DS1307 (рис.2.10) з годинником реального часу.[38] Модуль постачається повністю зібраним та із запрограмованим поточним часом (перед використанням у проекті встановіть свій часовий пояс). З літієвою батареєю (CR2032-210mAh) модуль може працювати не менше ніж 5 років без додаткового джерела живлення 5В.

Рисунок 2.10 Вигляд модулю DS1307.

- Модуль на DS3231 – цей модуль годинника реального часу відрізняється від аналогічних модулів тим, що він побудований на унікальному чіпі DS3231 (рис.2.11).[39] Його унікальність полягає в дуже високій точності годинника. Це було досягнуто шляхом розміщення кристалічного резонатора в корпусі мікросхеми та забезпечення температурної компенсації та цифрової частотної корекції генератора, що задає.

Рисунок 2.11 Вигляд модулю DS3231.

2.4 Опис роботи системи та її алгоритм

Для реалізації роботи системи визначаються наступні цілі перед розробкою:

- Оптимізація робочого процесу.
- Збір статистики.
- Аналіз витраченого часу на кожний з маршрутів за різних обставин.
- Прогнозування часу прибуття до наступної точки зупинки.
- Підвищення ефективності пересування за різними маршрутами.

Кожен маршрут, як зазначалося у концептуальній схемі, має зупинки, певний кілометраж між ними, час на подолання цього шляху при певній швидкості та державний номер автобусу, що курсує — вся ця інформація буде доступною для користувача.

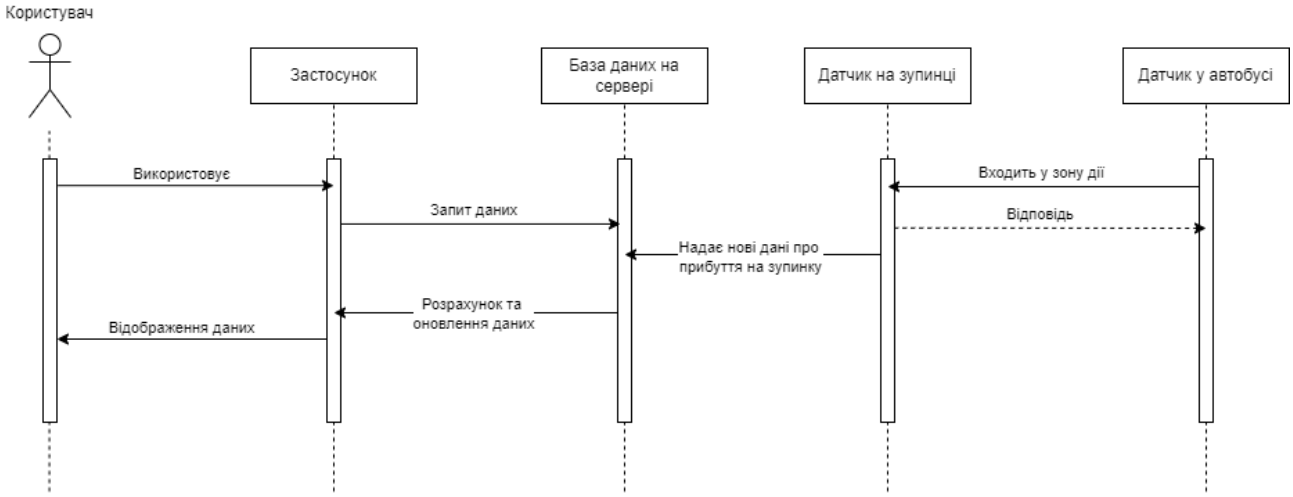


Рисунок 2.12 Відображення роботи системи у діаграмі послідовностей.

Застосунок передбачає можливість вибору маршруту серед наявних. Після вибору, користувачу надається інформація коли прибуде найближчий автобус та скільки часу займе його подорож (рис.2.12).

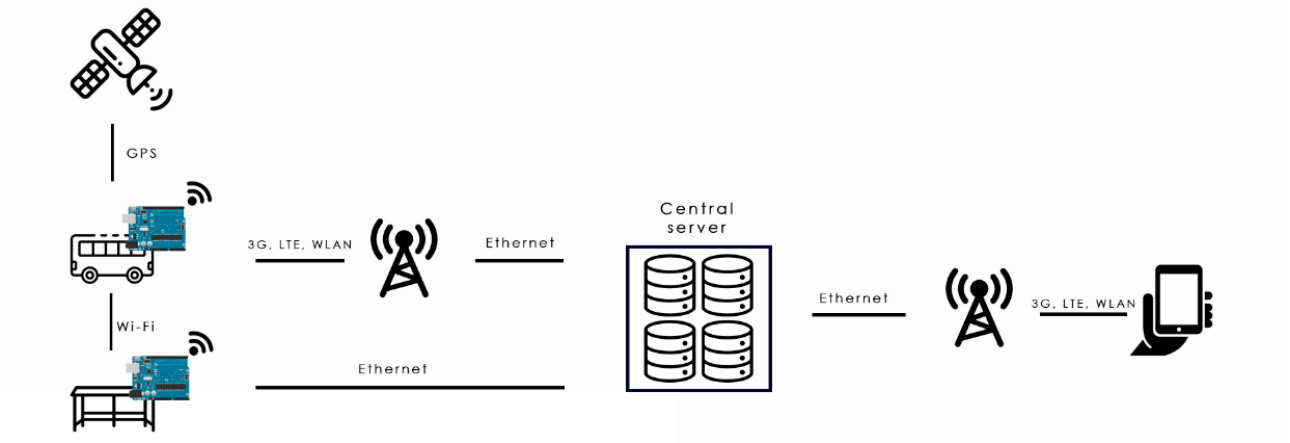


Рисунок 2.13 Архітектура системи.

Прибуваючи до зони дії датчику на зупинці, цей датчик подає сигнал про прибуття датчику у автобусі (рис.2.13). Отримавши сигнал-відповідь, записуються дані до бази даних інформація до таблиці logs, а саме: id запису, id автобусу, id зупинки, id маршруту та час прибуття даного автобусу. Система записує нові дані про прибуття та корегує дані з приблизного часу прибуття (прогнозовані) на чіткий час. Далі, маючи дані про кілометраж маршруту та необхідний час на його подолання, система підраховує час на подальший маршрут, окремо для кожної зупинки. Нові підраховані дані замінюють попередні у базі даних та оновлюються у додатку користувача (рис.2.14).

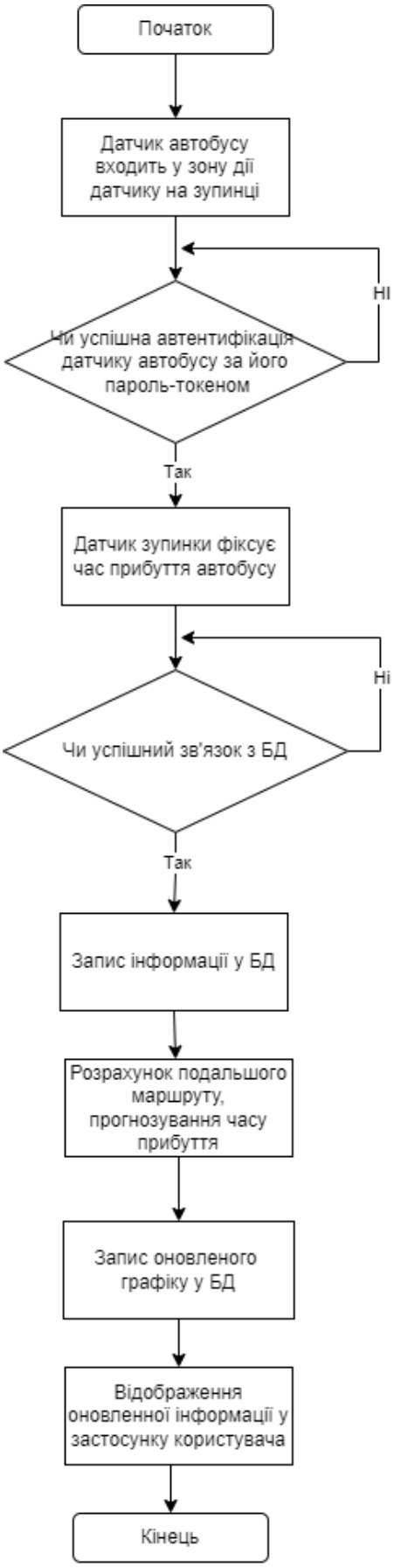


Рисунок 2.14 Алгоритм роботи системи.

Особливістю даної системи є її комбінований тип. Крім відстеження у реальному часі (яке має неточності та періодичні втрати зв’язку), завдяки фіксації часу прибуття на зупинках, система завжди отримує чіткі актуальні дані та оновлює прогнозування часу незалежно чи доступний автобус онлайн, чи ні. Таким чином користувачу завжди доступна актуальна інформація про автобус, маршрут між зупинками та необхідний час на його подолання.

Висновки до розділу

В даному розділі розроблено логічну та фізичну моделі даних. Логічна представлена у вигляді ER-діаграми та має чотири сутності «Автобус», «Маршрут», «Зупинка» та «Класифікатор автобусів», кожна з яких пов’язані між собою та маюсь унікальні атрибути, що визначають їх унікальність. У фізичній моделі БД наведено таблиці, їх поля та типи даних, що застосовуються у системі.

Також наведено та описано загальні поняття програмних технологій, що використовуються для розробки системи. Серед них: мова програмування Python, фреймворк Flask, програма баз даних MongoDB, технології API та Webhook. Аналогічно розглянуто мінімально необхідне апаратне забезпечення для системи.

Відображено схематично архітектуру системи аналізу, її алгоритм роботи, побудовані логічна та фізична моделі баз даних, зображена взаємодія з користувачем у вигляді діаграми послідовностей, описана особливість роботи системи.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Початок роботи

Мінімальним застосунком Flask може бути наступне (рис.3.1):

```
1   from flask import Flask
2
3   app = Flask (__name__)
4
5   @app.route("/")
6
7   def hello_world():
8       return "<p>Hello, World!</p>"
```

Рисунок 3.1 Приклад мінімального застосунку Flask.

Спочатку імпортується імпортується Flask клас, екземпляром якого буде додаток. Далі створюється екземпляр цього класу. Першим аргументом є назва модуля або пакета програми. **__name__** – це зручна абревіатура для цього, яка підходить для більшості випадків. Таким чином Flask знає, де шукати ресурси, такі як шаблони та статичні файли. Декоратор **route()** необхідний, щоб сказати Flask, який URL має виконувати функція. Функція повертає повідомлення, яке необхідно відобразити в браузері користувача. Тип вмісту за замовчуванням – HTML, тому HTML у рядку відтворюється браузером.

Маршрутизація та URL-адреси. Сучасні веб-додатки використовують значущі URL-адреси, щоб допомогти користувачам. Користувачі, швидше за все, вподобають сторінку й повернуться, якщо сторінка використовує змістовну URL-адресу, яку вони можуть запам’ятати та використовувати для безпосереднього відвідування сторінки. Для зв’язку функції з URL-адресою використовується декоратор **route().**

```
14  @app.route('/')
15  def index():
16      bus_list = bus_routes.find()
17      return render_template('index.html', bus_list=bus_list)
18
19  @app.route('/get_stations', methods=['GET','POST'])
20  def get_stations():
21      if request.method == 'POST':
22          bus_id = request.form.get('buses')
23          print("bus_routes_stations_id" + bus_id)
24
```

Рисунок 3.2 Маршрутизація у додатку.

Прикладом маршрутизації у рис.3.2 є `@app.route('/')` – перехід на головну сторінку та @app.route(**'/get_stations'..**) перехід на сторінку get_stations.

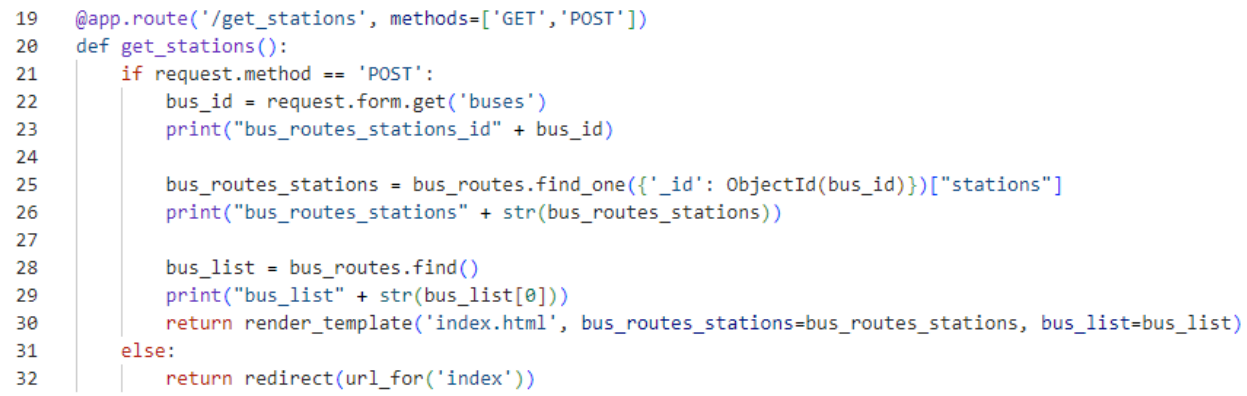
HTTP-методи. Веб-програми використовують різні методи HTTP під час доступу до URL-адрес. За замовчуванням маршрут відповідає лише на запити GET. Також можливо використовувати аргумент методу декоратора **route()** для обробки різних методів HTTP. Існує п’ять методів: GET, POST, HEAD, OUT, DELETE.

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

Рисунок 3.3 HTTP-метод у додатку.

На рис.3.3 відображено використання одразу двох HTTP-методів, а саме **GET** і **POST**. Для отримання даних надсилається повідомлення GET і сервер повертає дані. POST використовується для надсилання даних форми HTML на сервер. Щоб обробляти запити GET і POST, в прикладі вони додані це в метод app.route() декоратора. Яким би не був бажаний запит, змінюється він в декораторі.

В залежності від методу ж можливість виведення різних типів даних. Наприклад (див.рис. 3.4):



```
19  @app.route('/get_stations', methods=['GET','POST'])
20  def get_stations():
21      if request.method == 'POST':
22          bus_id = request.form.get('buses')
23          print("bus_routes_stations_id" + bus_id)
24
25          bus_routes_stations = bus_routes.find_one({'_id': ObjectId(bus_id)})["stations"]
26          print("bus_routes_stations" + str(bus_routes_stations))
27
28          bus_list = bus_routes.find()
29          print("bus_list" + str(bus_list[0]))
30          return render_template('index.html', bus_routes_stations=bus_routes_stations, bus_list=bus_list)
31      else:
32          return redirect(url_for('index'))
```

Рисунок 3.4 Приклад HTTP-методу у функції додатку.

В додатку присутні два приклади обробки методів. Перша (рядок 19) працює на обмеження, в get_stations() пропускається лише два види запитів: GET, тобто звичайний перехід за посиланням /get_stations або, при переході через submit форми, – буде використовуватися POST метод, який передаватиме параметри (наприклад id автобусу, обраного користувачем), крім переходу на сайт. Також, нижче (рядок 21) наведено ще одне використання методу POST – розділення дій на різні методи.

if request.method == 'POST'– при умові, якщо відомо, що користувач зробив запит з форм, та введені параметри точно існують, - завдання може бути оброблене та виведене на сторінці всі списки станцій, за обраним маршрутом. Якщо використовується GET метод (рядок 31) – перехід на сторінку відбувся не з форми, відповідно немає необхідності робити запит до бд за наявними даними, оскільки не було обрано пара метри у формі – як результат, замість виведення помилки, користувача поверне на головну сторінку (рядок 32).

Шаблони рендерінгу. Створення HTML з Python є насправді досить громіздким, оскільки потрібно самостійно перевірити HTML, щоб забезпечити безпеку програми. Через це Flask автоматично налаштовує механізм шаблонів Jinja2. Для візуалізації шаблону можна використовувати метод render_template(). Необхідно лише вказати ім’я шаблону та змінні, які треба передати механізму шаблонів, як аргументи ключового слова. Flask буде шукати шаблони в папці template. На рис.3.5 на прикладі додатку наведено приклад:

```
14  @app.route('/')
15  def index():
16      bus_list = bus_routes.find()
17      return render_template('index.html', bus_list=bus_list)
```

Рисунок 3.5 Приклад методу render_template () у додатку.

Шаблони Jinja – це простий текстовий файл. Jinja генерує будь-який текстовий формат, наприклад HTML, XML, CSV. Такі шаблони містять **змінні** або **вирази**, які замінюються значеннями при відображенні шаблону, та **теги**, які керують логікою шаблонів. Нижче, на рис.3.6 наведено приклад такого шаблону.

```
<div class="form-group">
    <div class="input-group flex">

        <select name="buses" class="form-control bus_routes"  id="buses">
            {% for bus in bus_list %}
                <option value="{{ bus._id }}"> {{ bus.name }}| {{ bus.start }} - {{ bus.end }} </option>
            {% endfor %}
        </select>

       <span class="input-group-btn bus_routes-btn-wrap">
          <button type="submit" class="btn btn-default bus_routes-btn" id="add-btn">SEARCH</button>
       </span>

    </div>
</div>
```

Рисунок 3.6 Приклад шаблону додатку.

Конфігурація синтаксису виглядає наступним чином (рядки 37-39):

- {% ... %} – для заяв;
- {{ ... }} – для друку виразів у вихідний текст шаблону;
- {# ... #} – для коментарів, не включені у вихідний шаблон шаблону.

Необхідним є **підключення бази даних до проекту** (рис.3.7). Перш за все імпортується flask_pymongo. Далі створюється об’єкт Flask - app = Flask(__name__), який буде використовуватися для ініціалізації клієнта MongoDB. Конструктор PyMongo (імпортований flask_pymongo) приймає об’єкт застосунку Flask та рядок URl бази даних. app.config['MONGO_URI'] – пов’язує додаток з екземпляром MongoDB.

```
1   from flask import Flask, render_template, request, jsonify, url_for, redirect
2   from flask_pymongo import PyMongo
3   from bson.objectid import ObjectId
4   import json
5   import os
6   from flask_httpauth import HTTPBasicAuth
7   from pymongo import MongoClient
8
9
10  app = Flask(__name__)
11
12  app.config['MONGO_URI'] = os.getenv ('MONGO_RI_CONFIG')
```

Рисунок 3.7 Підключення MongoDB.

3.2 Робота з датчиками

Будь-яка робота з даними повинна передбачати безпеку передачі та шифрування даних. У додатку використовується декілька методів:

1. Створення динамічної маршрутизації під кожний датчик, тобто, окреме посилання на кожний пристрій (див. рис. 3.8), для цього додається <tracker_id> (ім'я змінної) до кожного маршруту /add_tracking_info/<tracker_id>. За необхідності, можна використовувати перетворювач, щоб вказати тип аргументу, наприклад <converter:variable_name>, але у даній системі це обмеження буде на стороні бази даних, оскільки у айді об`єкта у MongoBD має свою особливу структуру.

```
169 @app.route('/add_tracking_info/<tracker_id>', methods=['POST'])
170 @auth.login_required
171 def add_tracking_info(tracker_id):
172     if(trackers.count_documents({'_id': ObjectId(tracker_id)}) > 0 ):
173         requests = request.json
174         station = requests.get('station')
175         timestamp  = requests.get('timestamp')
176         print (id, station, timestamp)
177         if None not in (id, station, timestamp):
178             mongo.db.trackings.insert_one(requests)
179             return { "status" : 200, "desctiption" : "ok"}
180
181     return { "status" : 505, "desctiption" : "bus not exist"}
```

Рисунок 3.8 Функція перевірки наявності автобусів.

2. Обмеження лише на метод POST.

3. Використання базової автентифікації HTTP для захисту маршруту ' / '. Кожна зупинка має свій унікальний токен-пароль, занесений у базу даних (рис.3.9), який також і перевіряється перед автентифікацією (рис.3.10).

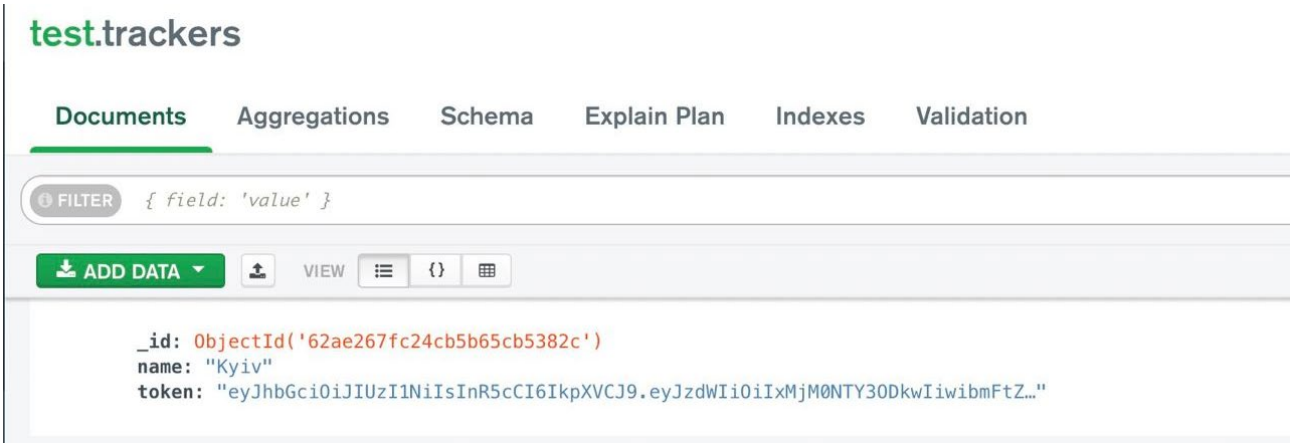


Рисунок 3.9 Приклад запису зупинки та її токену.

```
159  auth = HTTPBasicAuth()
160
161  @auth.verify_password
162  def verify_password(username, password):
163      token = trackers.find_one({'_id': ObjectId(username)})['token']
164      if token == password:
165          return username
```

Рисунок 3.10 Перевірка токену.

Нижче, (рис.3.11) наведено приклад запиту, який буде робити датчик, зроблений з програмного забезпечення postman. Для цього необхідно обрати базову авторизацію, вказати username, як id датчику, та пароль, що зберігається у базі.

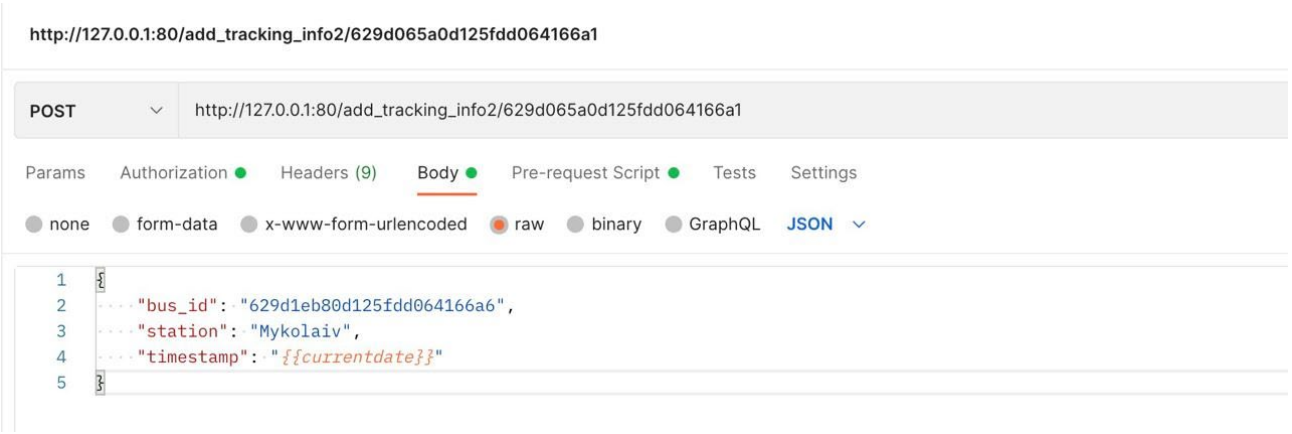


Рисунок 3.11 Запит з postman.

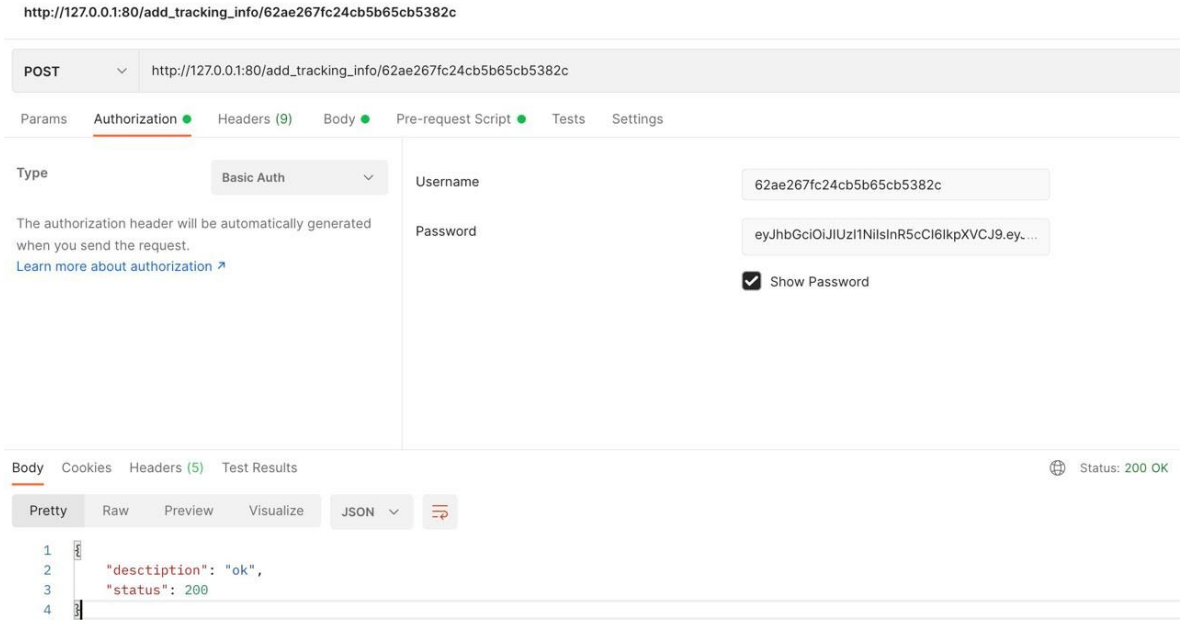


Рисунок 3.12 Успішний запит про зупинку у БД, повертає "status: 200".

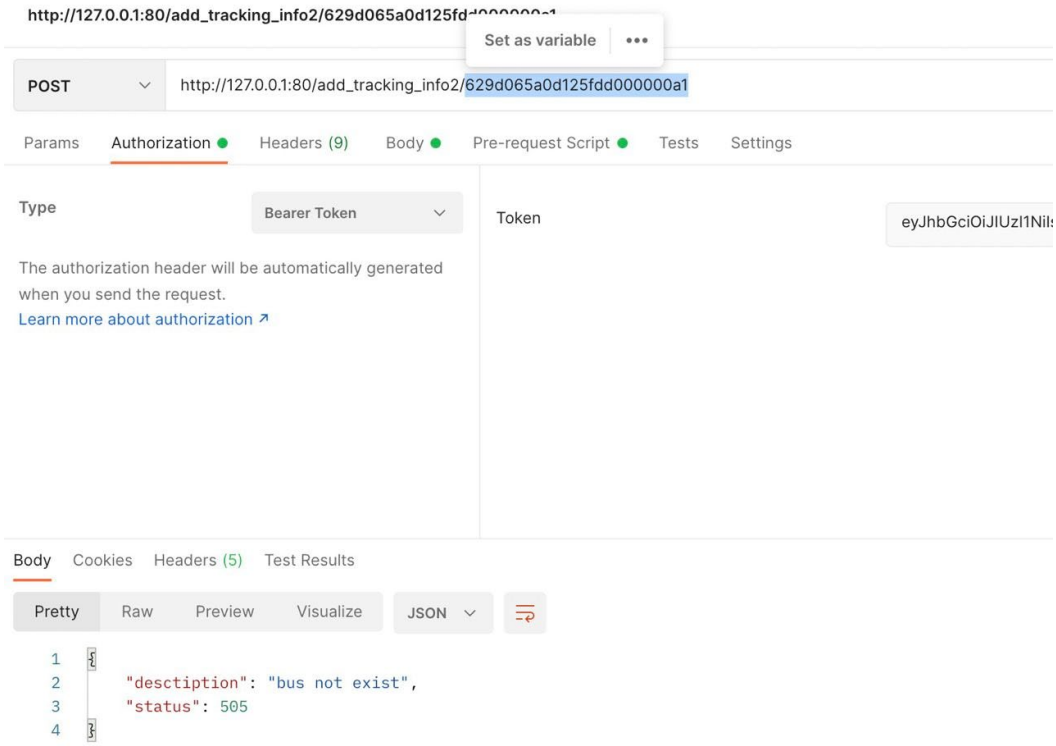


Рисунок 3.13 Невдалий запит про зупинку у БД "status: 505".

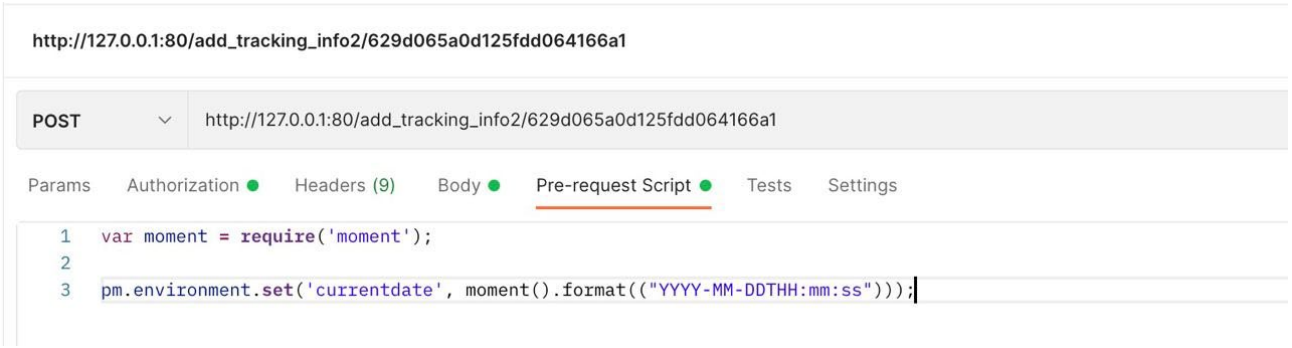


Рисунок 3.14 Запис часу прибуття автобусу на зупинку.

Після перевірки існування автобусу, отримуються змінні та перевіряються чи не порожні вони (рис.3.15).

```
170  @auth.login_required
171  def add_tracking_info(tracker_id):
172      if(trackers.count_documents({'_id': ObjectId(tracker_id)}) > 0 ):
173          requests = request.json
174          station = requests.get('station')
175          timestamp  = requests.get('timestamp')
176          print (id, station, timestamp)
177          if None not in (id, station, timestamp):
178              mongo.db.trackings.insert_one(requests)
179              return { "status" : 200, "desctiption" : "ok"}
180
181      return { "status" : 505, "desctiption" : "bus not exist"}
```

Рисунок 3.15 Функція add_tracking_info.

Далі перевірені дані заносяться у базу даних (див.рис.3.16).

```
178              mongo.db.trackings.insert_one(requests)
179              return { "status" : 200, "desctiption" : "ok"}
```

Рисунок 3.16 Занесення даних у БД.

4. Наявність датчика який робить запит у базі датчиків

5. Перевірка на непусті значення (рис.3.17).

```
173          requests = request.json
174          station = requests.get('station')
175          timestamp  = requests.get('timestamp')
176          print (id, station, timestamp)
177          if None not in (id, station, timestamp):
```

Рисунок 3.17 Перевірка на наповненість значень.

Взаємодія з додатком. Відкриваючи web-додаток користувач потрапляє на головну сторінку (рис. 3.18). На даній сторінці є випадаючий список з переліком маршрутів. Обираючи бажаний, користувач натискає кнопку Search для пошуку автобусів, що наразі курсують за обраним маршрутом.

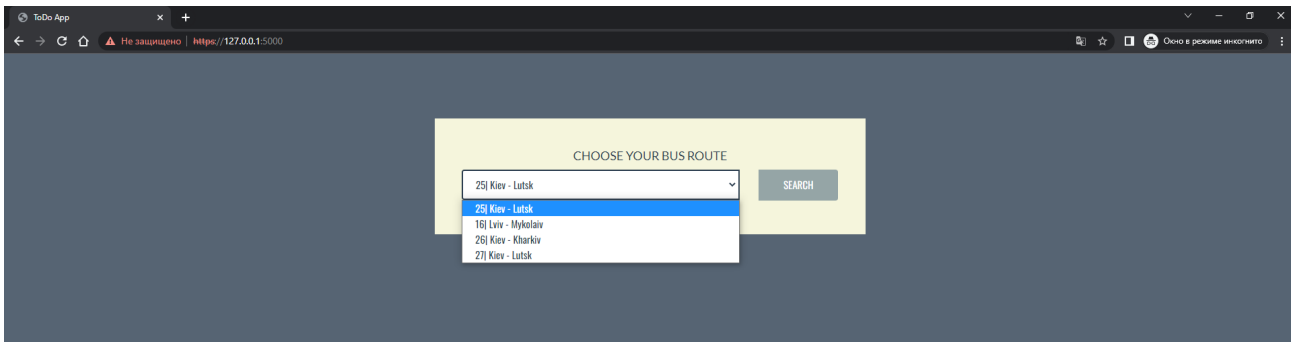


Рисунок 3.18 Головна сторінка.

```
<form action="{{ url_for('get_stations') }}" method="POST" role="form">
    <div class="form-group">
        <div class="input-group flex">

            <select name="buses" class="form-control bus_routes"  id="buses">
                {% for bus in bus_list %}
                    <option value="{{ bus._id }}">
                        {{ bus.name }}| {{ bus.start }} - {{ bus.end }}
                    </option>
                {% endfor %}
            </select>

            <span class="input-group-btn bus_routes-btn-wrap">
                <button type="submit" class="btn btn-default bus_routes-btn" id="add-btn">
                    SEARCH
                </button>
            </span>
        </div>
    </div>
</form>
```

Рисунок 3.19 Головна сторінка у VSCode.

Після натискання кнопки Search, розгортається наступна сторінка /get_tracking_info, де розгортається повний список маршруту, з часом перебування на зупинках (рис.3.20).

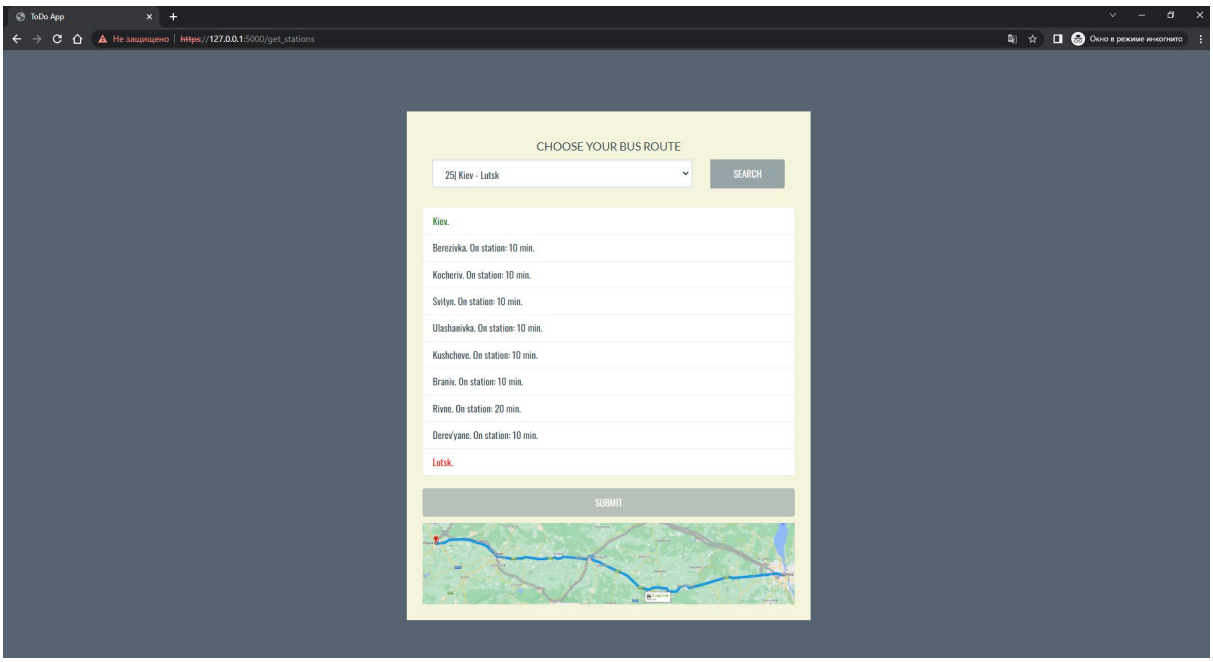


Рисунок 3.20 Сторінка з зупинками на обраному маршруті.

На цьому етапі користувач має можливість обирати свій шлях в межах обраного маршруту. Програмно передбачено (ри.3.21), що користувачу необхідно обрати не менше і не більше двох зупинок, щоб активувалася кнопка Submit. Обирати зупинки можна в будь-якому порядку – система розпізнає де початок та де кінець маршруту за допомогою умови if (index_of(stations[0]) > index_of(stations[1])), так як кожна зупинка має свій порядковий індекс та сам маршрут є одностороннім. В довільному виборі початком руху буде обрана зупинка з найменшим індексом, а кінцева – з найбільшим з обраних.

```
@app.route('/get_tracking_info', methods=['GET','POST'])
def get_tracking_info(bus_id):
    if request.method == 'POST':
        stations = request.form.get('stations')

        bus_routes_stations = bus_routes.find_one({'_id': ObjectId(bus_id)})["stations"]
        print("bus_routes_stations" + str(bus_routes_stations))

        bus_list = bus_routes.find()
        print("bus_list" + str(bus_list[0]))

        if(index_of(stations[0]) > index_of(stations[1])):
            start = stations[1]
            end = stations[0]
        else:
            start = stations[1]
            end = stations[0]
```

Рисунок 3.21 Функція get_tracking_info.

Після підтвердження обраних зупинок, відкривається наступна сторінка /get_tracking_info та автобус (рис.3.21). Також форма передає зупинки stations = request.form.get('stations').

{% if bus_routes_stations %} – якщо зупинкі існують – форма їх надає (див рис3.21). Кожний рядок зупинки має свою функцію " onClick="GFG_click(this.id)", яка викликається при кожному натисканні на кнопку.

```
{% if bus_routes_stations %}
    <form action="{{ url_for('get_tracking_infos') }}" method="POST" id="routes_form" role="form">

        <ul class="list-group t20">
            {% for station in bus_routes_stations.stations %}

                <li class="list-group-item {{'end-station' if station.type == 'end' }} {{'start-station' if station.type
                id="{{ station.name }}" onClick="GFG_click(this.id)">
                    {{ station.name }}.

                    {% if station.type == 'mid' %}
                        On station: {{ station.waiting_time }} min.
                    {% endif %}
                </li>
            {% endfor %}
        </ul>

        <input id="prodId" name="bus_id" type="hidden" value="{{ bus_id }}">
        <span class="input-group-btn btn-submit-rout">
            <button type="submit" class="btn btn-default bus_routes-btn" id="route-btn" >SUBMIT</button>
        </span>
        <img src="{{ bus_routes_stations.route_url }}" class="img-map" style="width: 100%;">
    </form>
{% endif %}
```

Рисунок 3.21 Форма списку зупинок.

При кожному натисканні на зупинку, функція (рис.3.22) перевіряє скільки елементів у списку counter_route = start_and_end.length, якщо їх менше двох – елемент стає обраним (addClass("chosed")) та записується у список. Відповідно, при натисканні на вже обраний елемент – стан обраного з нього знімається (removeClass("chosed")) та видаляється зі списку. Якщо після попередніх дій елементів у списку два – кнопка для підтвердження запиту активується, якщо умови не виконані – кнопка залишається неактивною.

```
function GFG_click(clicked) {
    counter_route = start_and_end.length ;

    var clicked_id = "#" + clicked

    if(counter_route < 2 && !($(clicked_id).hasClass("chosed"))){
        $(clicked_id).addClass("chosed");
        start_and_end.push(clicked);
        counter_route++;
    }
    else if($(clicked_id).hasClass("chosed")){
        $(clicked_id).removeClass("chosed");
        removeItems(start_and_end, clicked)
        counter_route--;
    }

    console.log (counter_route)
    if(counter_route >= 2 ){
        document.getElementById('route-btn').disabled=false;
        console.log("11");
    }
    else{
        document.getElementById('route-btn').disabled=true;
        console.log("24");
    }


    console.log(start_and_end)
    console.log("ID = " + clicked);
}
```

Рисунок 3.22 Функція GFG_click.

Після підтвердження форми кнопкою, відправляється запит до функції get_tracking_infos (рис.3.23). Ця функція отримує bus_id та станції, що обирались у формі попередньо.

```
{% if bus_routes_stations %}
    <form action="{{ url_for('get_tracking_infos') }}" method="POST" id="routes_form" role="form">
```

Рисунок 3.23 Запит до функції get_tracking_infos.

Подальша обробка полягає в наступному: за допомогою bus_id отримуються всі маршрути та зупинки обраного маршруту (рис.3.24). Це необхідно, щоб виводити на екран користувача загальну інформацію про зупинки разом з обраними параметрами.

```
76      bus_routes_stations = bus_routes.find_one({'_id': ObjectId(bus_id)})["stations"]
77      print("bus_routes_stations" + str(bus_routes_stations))
78
79      bus_list = bus_routes.find()
80      print("bus_list" + str(bus_list[0]))
```

Рисунок 3.24 Вивід списку зупинок користувачу.

Для пошуку та вибору найближчого автобусу на маршруті обирається той, що відмітився на найближчій зупинці нещодавно. У БД (logs) фіксується час прибуття автобусу на зупинку (рис.3.25). У БД вказується id об’єкту в базі, id автобусу, назва зупинки та час прибуття.

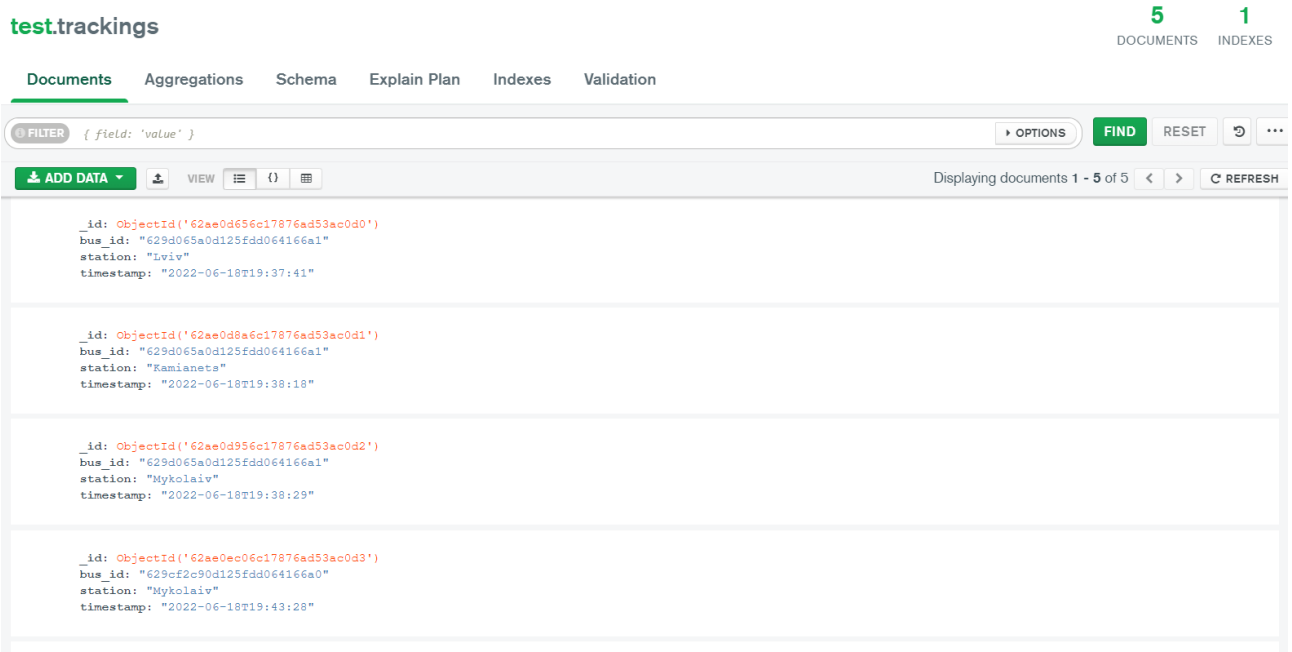


Рисунок 3.25 Записи прибуття автобусів на зупинки.

Список усіх зупинок передається у find_near_bus_list (рис.3.26) та проводиться пошук найближчого за часом та місцем запису – "$elemMatch" : find_near_bus_list.

```
90      find_near_bus_list = bus_routes_stations[:start]
91      route_list = bus_routes_stations[start:end]
92
93      nearest_station = trackings.aggregate([{
94          "$elemMatch" :  find_near_bus_list
95          }
96      ])
```

Рисунок 3.26 Пошук найближчого автобусу.

3.3 Принцип розрахунку часу

Якщо знайдений найближчий запис зупинки той самий, що і обрана зупинка – вивід часу очікування буде дорівнювати нулю (рис.3.27).

```
if(nearest_station["name"] == start):
    stations_to_wait = []
```

Рисунок 3.27 Умова очікування.

В іншому випадку, якщо між користувачем та зупинкою є якась дистанція, - ці зупинки записуються у stations_to_wait. Змінна time_to_bus_came – необхідний час до прибуття автобусу. Функція прораховує усі зупинки, які необхідно чекати до прибуття автобусу, які включають в себе час самої поїздки – route_list[station].drive_time – між зупинками та час простою на зупинках – route_list[station].waiting_time.

```
else:
    stations_to_wait = bus_routes_stations[nearest_station:]

time_to_bus_came = 0
for station in range(len(stations_to_wait)-1):
   time_to_bus_came += stations_to_wait[station].waiting_time
   time_to_bus_came += stations_to_wait[station].drive_time
```

Рисунок 3.28 Розрахунок часу очікування на автобус.

Аналогічно проходить розрахунок часу на сам маршрут, з моменту посадки на автобус – time_to_drive = 0 (рис.3.29).

```
time_to_drive = 0
for station in range(len(route_list)-1):
   time_to_drive += route_list[station].waiting_time
   time_to_drive += route_list[station].drive_time
```

Рисунок 3.29 Розрахунок часу поїздки.

Також, користувачеві виводиться сума обох розрахованих даних для повної інформації по часу, що необхідний на поїздку (рис.3.30).

```
final_waiting_time = time_to_bus_came + time_to_drive;
```

Рисунок 3.30 Розрахунок суми часу очікування та часу поїздки.

Далі усі необхідні дані та розрахунки виводяться у додатку користувача (рис.3.31), (рис.3.32).

```
return render_template('index.html', bus_routes_stations=bus_routes_stations, final_waiting_time = final_waiting_time ,
                       bus_list=bus_list, time_to_bus_came = time_to_bus_came,
                       time_to_drive = time_to_drive, start = start, end = end, nearest_station = nearest_station )
```

Рисунок 3.31 Вивід перелічених даних у додатку користувача.

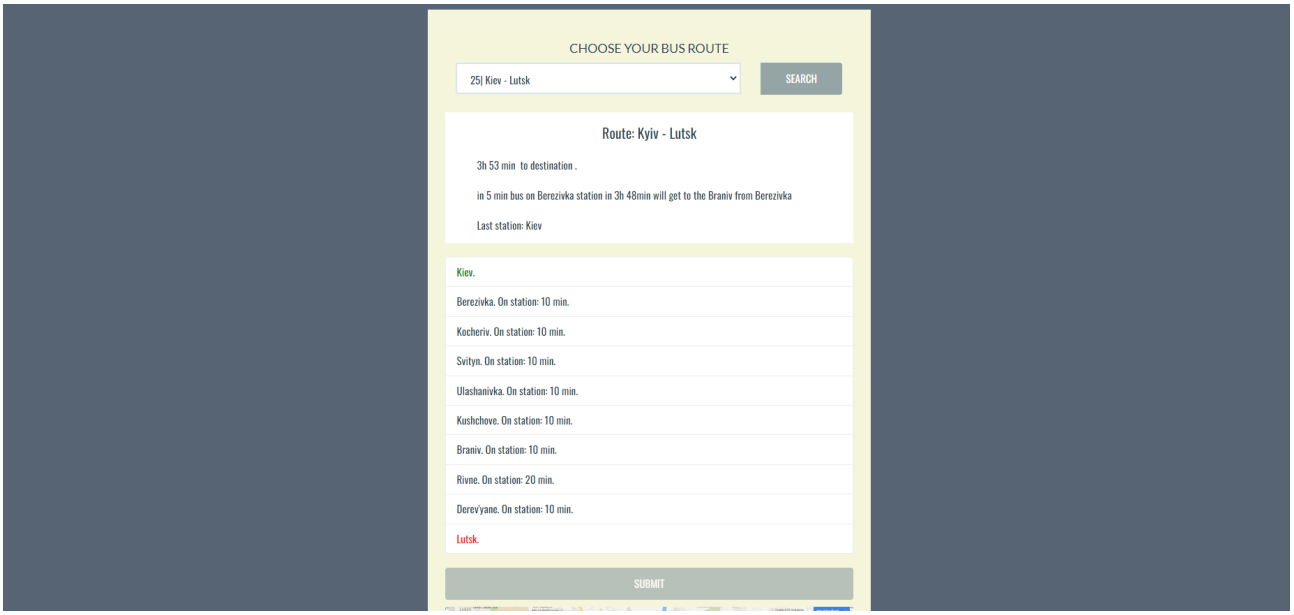


Рисунок 3.32 Вигляд виводу у web-додатку.

Висновки до розділу

У даному розділі програмно реалізовано та описано користувацький web-додаток. Оглянуто на прикладах системи використання технологій. Реалізована робота системи з MongoDB та даними у БД, які записуються та використовуються в подальшому для прогнозування часу на поїздку. Створено користувацький інтерфейс з можливістю вибору шляху на основі наявного маршруту. Розроблений алгоритм розрахунку часу та його відображення у додатку користувача.

ВИСНОВКИ

У результаті виконання бакалаврської роботи реалізовано систему аналізу та контролю переміщення міжобласного рейсового транспорту. Було:

1. Проаналізовано транспортну область та поняття GPS-трекінгу.
2. Наведено приклади мінімально необхідних складових для системи аналізу та контролю переміщення міжобласного рейсового транспорту для забезпечення роботи апаратної та програмної частин.
3. Побудовано архітектуру роботи системи аналізу. Створено алгоритм роботи системи. Побудовані логічна та фізична моделі баз даних та зображена взаємодія з користувачем у вигляді діаграми послідовностей.
4. Наведено та описано загальні поняття програмних технологій, що використовуються для розробки системи.
5. Створено web-додаток на мові Python та за допомогою веб-фреймворку Flask. Написано програмний код для роботи системи аналізу та контролю переміщення міжобласного рейсового транспорту. Розроблено інтерфейс користувача з можливістю вибору маршруту та відображенням прогнозованого часу на мандрівку.

При виконанні бакалаврської роботи усі поставлені задачі було виконано та реалізовано у вигляді користувацького web-додатку, який забезпечує фіксування часу прибуття транспортних засобів на кожній з зупинок впродовж маршруту та актуальне прогнозування часу для обраного маршруту.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. The History of Shipment Tracking URL: https://turvo.com/articles/shipment-tracking/ (дата звернення: 22.01.2022)
2. GPS Accuracy Tracking URL: https://www.gps.gov/systems/gps/performance/accuracy/ (дата звернення: 22.01.2022)
3. Vehicle Tracking Device Market Report Coverage URL: https://www.gminsights.com/industry-analysis/vehicle-tracking-market (дата звернення: 22.01.2022)
4. What is GPS? URL: https://www.arvento.com/en/what-is-a-vehicle-tracking-system (дата звернення: 22.01.2022)
5. What is GPS tracking and how it work URL: https://www.mixtelematics.com/blog/what-is-gps-tracking-and-how-does-it-work (дата звернення: 23.01.2022)
6. Difference Between Active and Passive GPS work URL: https://www.differencebetween.com/difference-between-active-and-vs-passive-gps/ (дата звернення: 28.01.2022)
7. Active GPS tracking vs Passive GPS tracking URL: https://www.rfwireless-world.com/Terminology/Difference-between-active-GPS-tracking-and-passive-GPS-tracking.html (дата звернення: 28.01.2022)
8. ITlynx URL: http://www.it-lynx.com/ (дата звернення: 25.01.2022)
9. Сторінка системи «Інспектор» URL: http://www.it-lynx.com/products/hardware-software-complex-inspector/gps-monitoring-system-inspector/
10. ProGPS URL: https://progps.com.ua/uk/ (дата звернення: 25.01.2022)
11. Підтримуване обладнання від ProGPS URL: https://progps.com.ua/uk/obladnannya/ (дата звернення: 25.01.2022)
12. Офіційна сторінка NaviTrack URL: https://navitrack.com.ua/ru/ (дата звернення: 26.01.2022)

13. A Beginner's Guide to Back-End Development URL: https://www.upwork.com/resources/beginners-guide-back-end-development#fundamentals (дата звернення: 3.02.2022)
14. How to Start Freelancing as a Front-End Developer & Find Work URL: https://www.upwork.com/resources/freelance-front-end-developer (дата звернення: 5.02.2022)
15. What Is The Difference Between Front-End And Back-End Development? URL: https://www.conceptatech.com/blog/difference-front-end-back-end-development (дата звернення: 5.02.2022)
16. Front end and back end URL: https://www.techtarget.com/whatis/definition/front-end (дата звернення: 5.02.2022)
17. What Is a Database? URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 10.02.2022)
18. Database (DB) URL: https://www.techtarget.com/searchdatamanagement/definition/database (дата звернення: 10.02.2022)
19. CASE-засоби для розробки інформаційних систем URL: https://it.wikireading.ru/39337 (дата звернення: 12.02.2022)
20. Web framework Flask URL: https://flask.palletsprojects.com/en/2.1.x/ (дата звернення: 12.02.2022)
21. What is Flask Python URL: https://pythonbasics.org/what-is-flask-python/ (дата звернення: 12.02.2022)
22. Flask vs Django in 2022: Which Framework to Choose? URL: https://hackr.io/blog/flask-vs-django# (дата звернення: 12.02.2022)
23. Create, read, update and delete? URL: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete (дата звернення: 23.02.2022)
24. What is CRUD? URL: https://w ww.codecademy.com/article/what-is-crud (дата звернення: 23.02.2022)
25. MongoDB URL:. https://www.mongodb.com/ (дата звернення: 13.02.2022)

26. Thinking about using MongoDB? URL: https://habr.com/ru/company/otus/blog/565700/ (дата звернення: 14.02.2022)
27. What is an API? (Application Programming Interface) URL: https://www.mulesoft.com/resources/api/what-is-an-api (дата звернення: 20.02.2022)
28. What’s a Webhook? URL: https://sendgrid.com/blog/whats-webhook/ (дата звернення: 20.02.2022)
29. APIs vs. Webhooks: What’s the difference? URL: https://www.mparticle.com/blog/apis-vs-webhooks (дата звернення: 20.02.2022)
30. Wi-Fi модуль NodeMCU V3 ESP8266 URL: https://arduino.ua/prod1492-wi-fi-modyl-nodemcu-esp8266 (дата звернення: 23.02.2022)
31. Wi-Fi модуль ESP8266 Witty Cloud URL: https://arduino.ua/prod1417-wi-fi-modyl-esp8266-witty (дата звернення: 23.02.2022)
32. Мікроконтролер D1 mini ESP8266 URL: https://beegreen.com.ua/plata-wemos-d1-mini-wifi-na-baze-esp8266-arduino-avr-12894 (дата звернення: 23.02.2022)
33. Плата міні-комп'ютера Raspberry Pi 3 Model URL: https://arduino.ua/prod1449-raspberry-pi-3-b (дата звернення: 23.02.2022)
34. Робота з GPS модулем URL: https://wiki.iarduino.ru/page/GPS-module/ (дата звернення: 04.05.2022)
35. Trema GPS модуль ATGM336H URL: http://www.kosmodrom.com.ua/el.php?name=ATGM336H-Modul (дата звернення: 04.05.2022)
36. Модуль GPS з антеною для Arduino APM2 Ublox NEO-6M URL: https://www.robostore.com.ua/modul-gps-ublox-neo-6m-s-antennoy-arduino-apm2/ (дата звернення: 05.05.2022)
37. HiLetgo GY-NEO6MV2 NEO-6M GPS URL: https://www.amazon.com/HiLetgo-GY-NEO6MV2-Controller-Ceramic-Antenna/dp/B01D1D0F5M/ (дата звернення: 05.05.2022)

38. Real Time Clock (DS1307) URL: https://arduino.ua/prod22-real-time-clock-modyl-s-batareikoi-ds1307 (дата звернення: 09.05.2022)
39. Real Time Clock DS3231 URL: https://radiostore.com.ua/p92774809-real-time-clock.html (дата звернення: 09.05.2022)
40. REST API Crash Course - Introduction + Full Python API Tutorial URL: https://youtu.be/qbLc5a9jdXo (дата звернення: 13.05.2022)
41. Flask Quickstart URL: https://flask.palletsprojects.com/en/1.1.x/quickstart/ (дата звернення: 15.05.2022)
42. Integrating MongoDB with Flask Using Flask-PyMongo https://stackabuse.com/integrating-mongodb-with-flask-using-flask-pymongo/ (дата звернення: 17.05.2022)
43. Flask-HTTPAuth URL: https://flask-httpauth.readthedocs.io/en/latest/ (дата звернення: 20.05.2022)
44. Running Your Flask Application Over HTTPS URL: https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https (дата звернення: 21.05.2022)
45. Jinja URL: https://palletsprojects.com/p/jinja/ (дата звернення: 21.05.2022)
46. Template Designer Documentation URL: https://jinja.palletsprojects.com/en/3.1.x/templates/ (дата звернення: 24.05.2022)

ДОДАТОК А

Презентація захисту

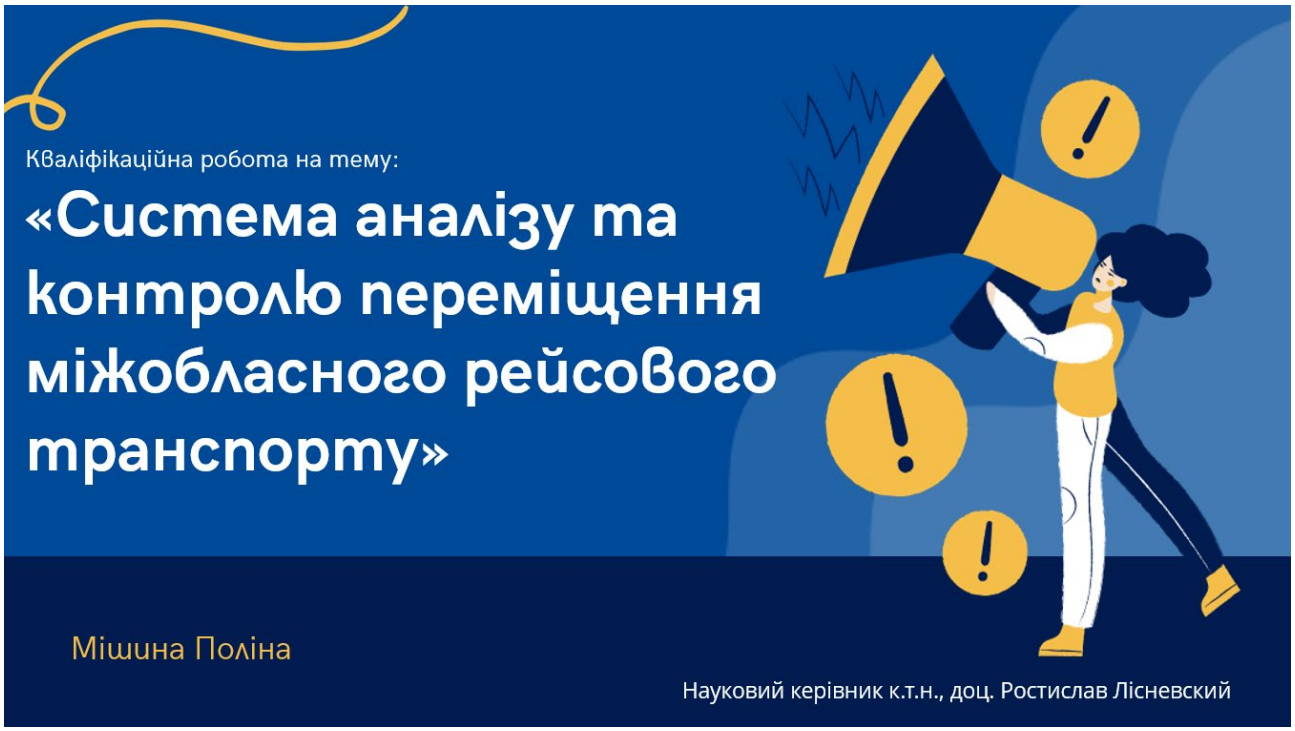


Рисунок А.1. Слайд 1

Рисунок А.2. Слайд 2

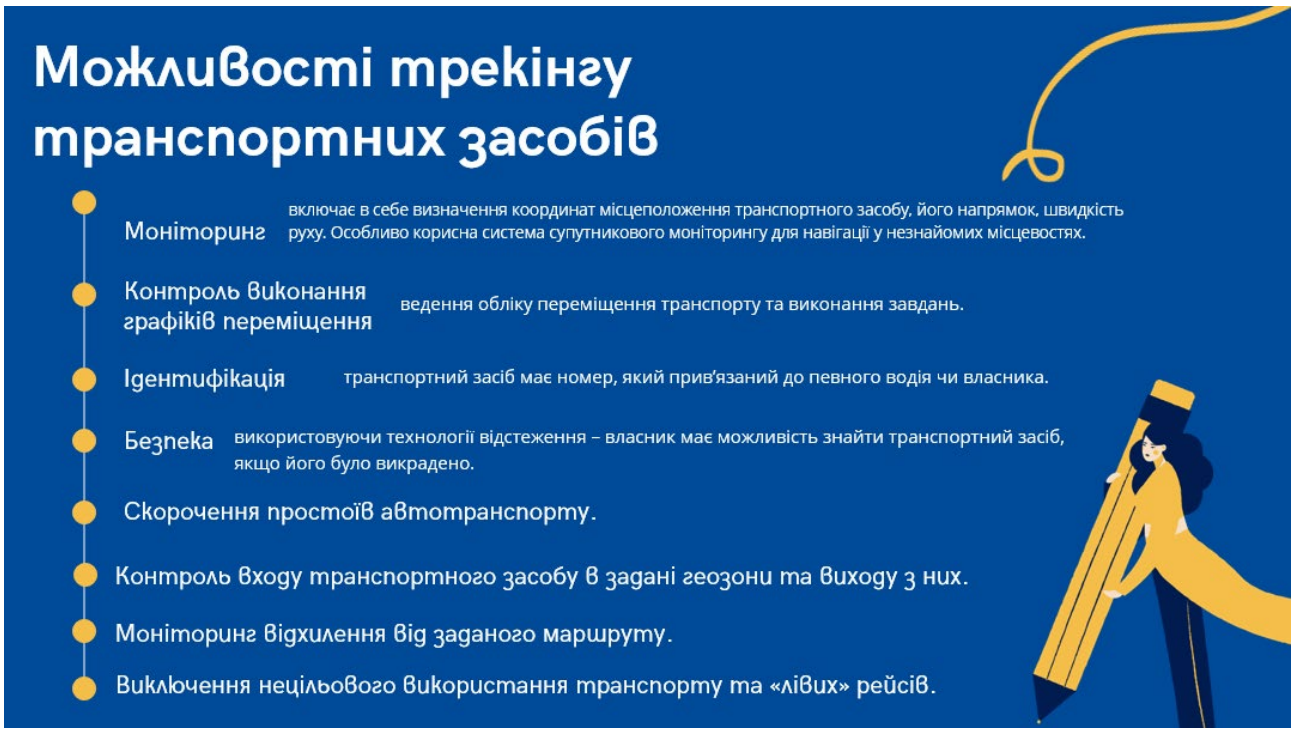


Рисунок А.3. Слайд 3

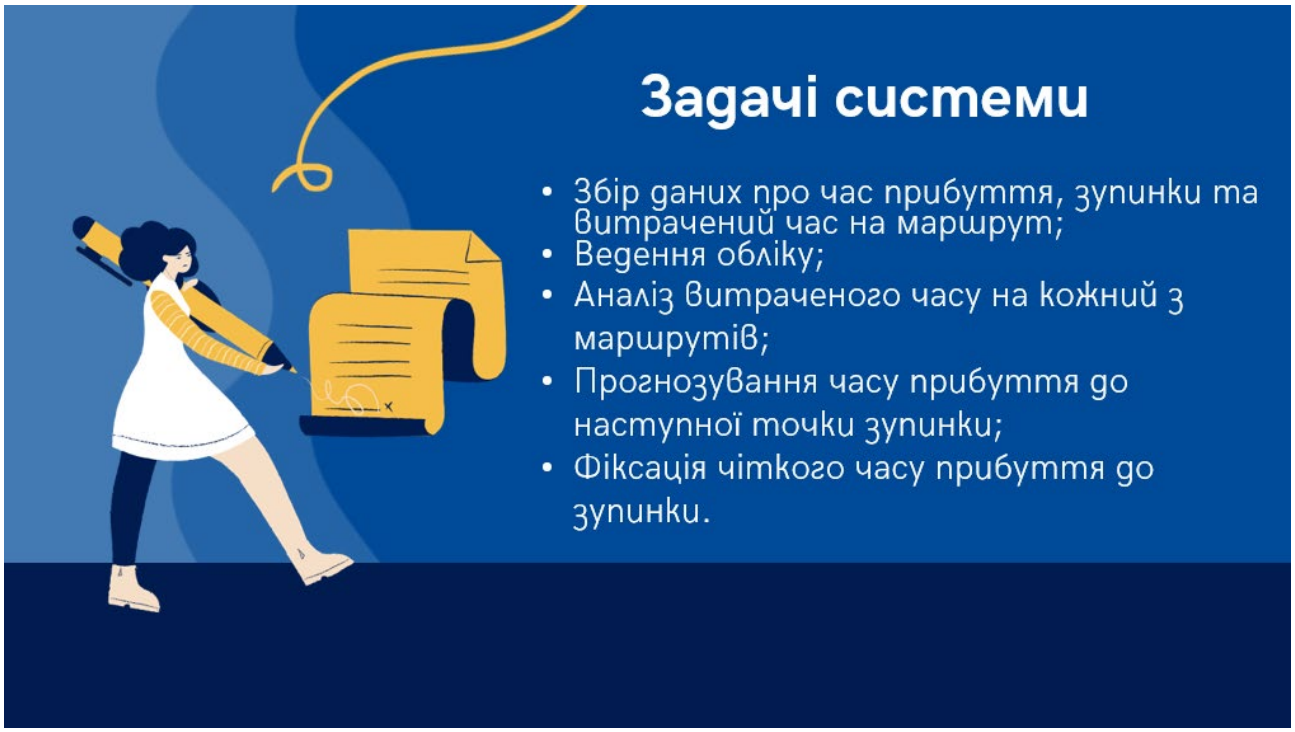


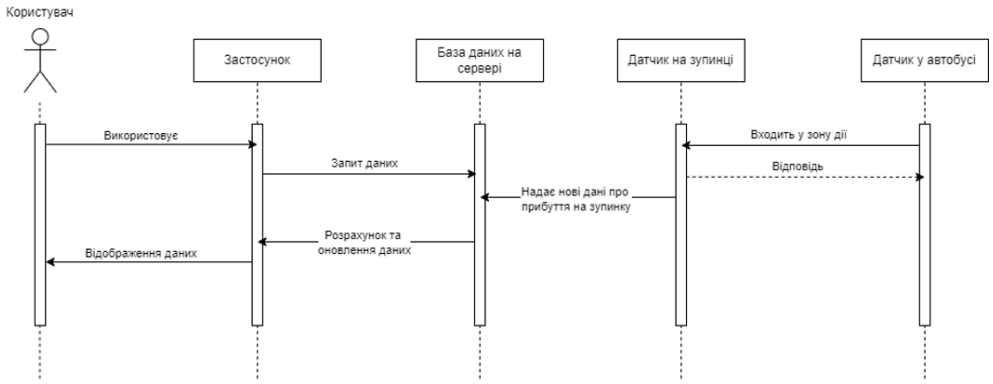
Рисунок А.4. Слайд 4

Рисунок А.5. Слайд 5

Рисунок А6. Слайд 6

Опис та алгоритм роботи системи

У вигляді блок-схеми



У вигляді діаграми послідовності

Рисунок А.7. Слайд 7

Архітектура системи

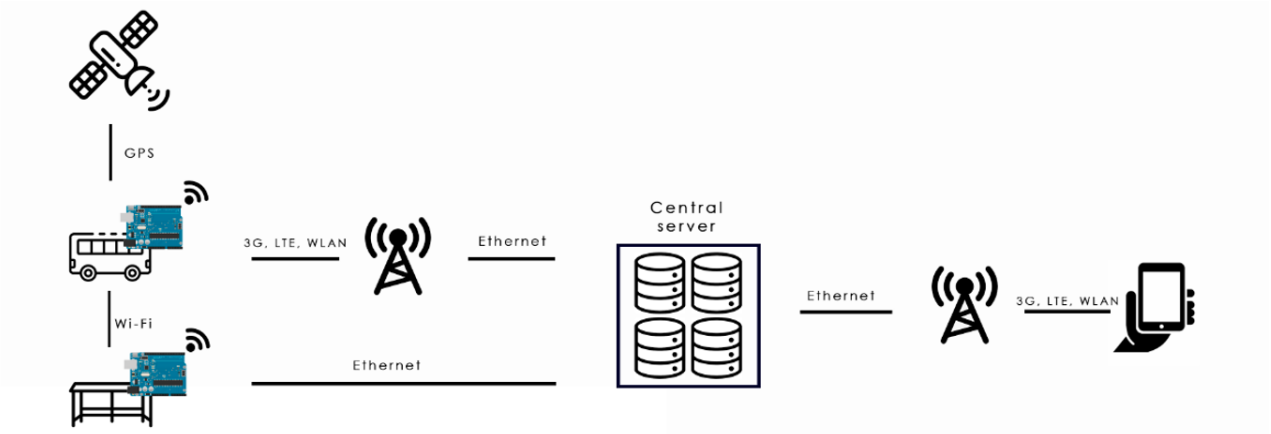


Рисунок А.8. Слайд 8

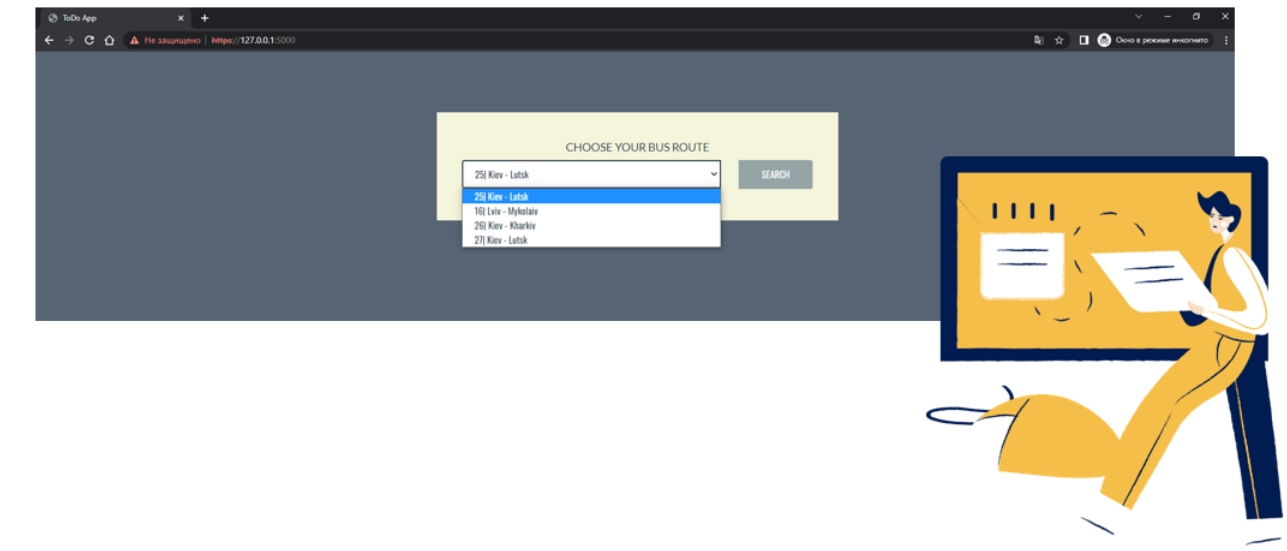


Рисунок А.9. Слайд 9

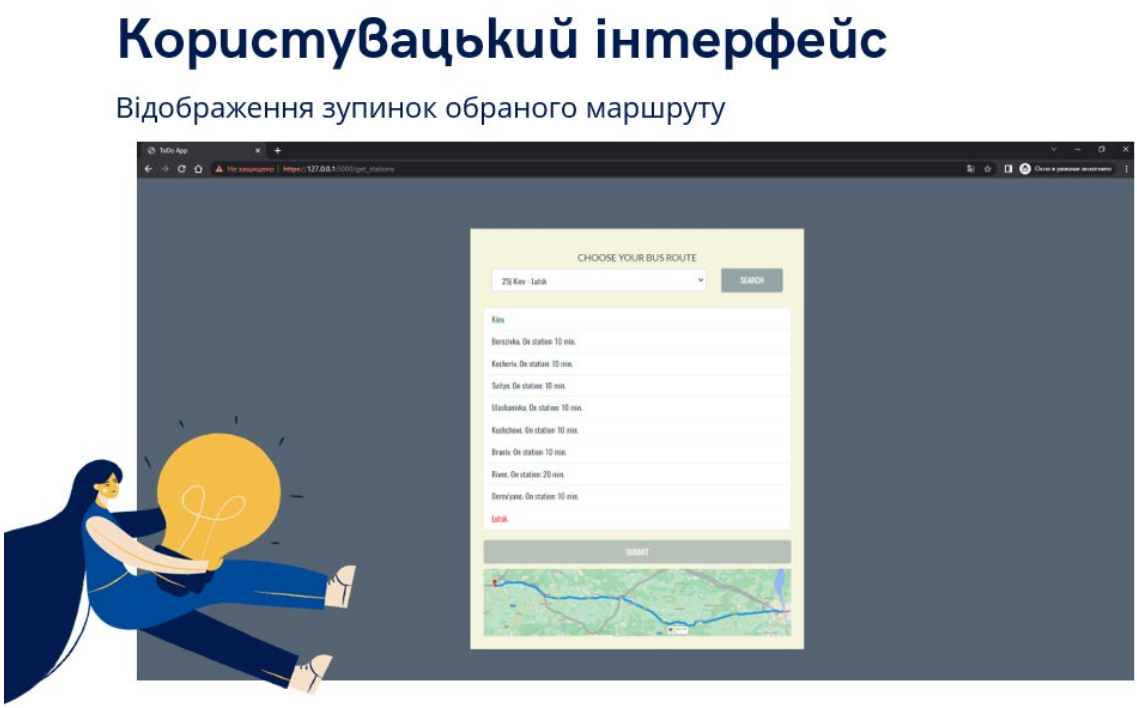


Рисунок А.10. Слайд 10

Користувацький інтерфейс

Прорахований час за обраним маршрутом

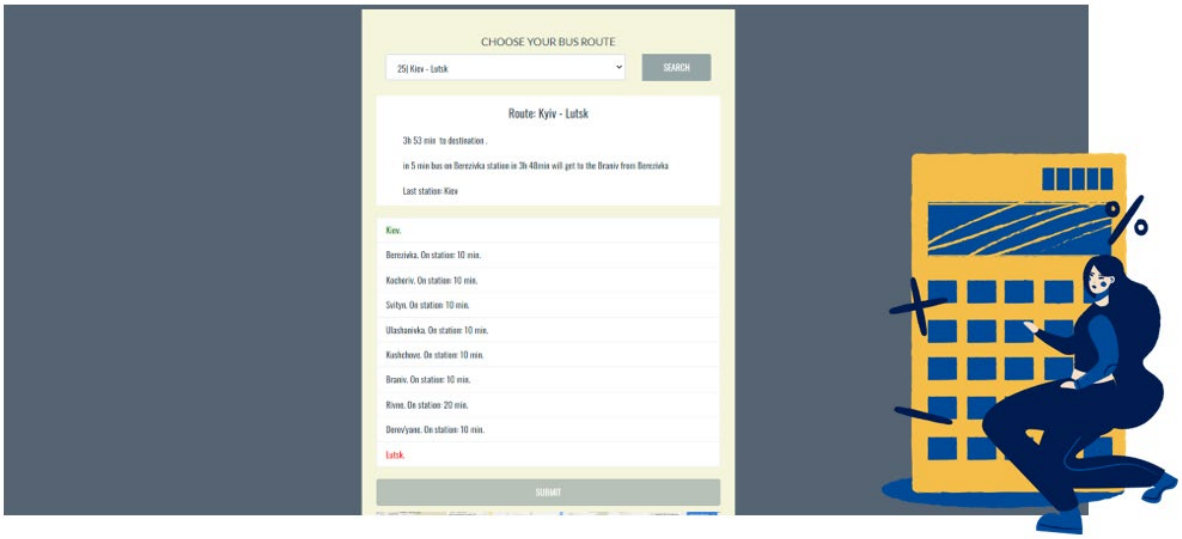


Рисунок А.11. Слайд 11

Висновки

В результаті кваліфікаційної роботи було:

- Проаналізовано транспортну область та поняття GPS-трекінгу.
- Наведено приклади мінімально необхідних складових для системи аналізу та контролю переміщення міжобласного рейсового транспорту для забезпечення роботи апаратної та програмної частин.
- Побудовано архітектуру роботи системи аналізу. Створено алгоритм роботи системи. Побудовані логічна та фізична моделі баз даних та зображена взаємодія з користувачем у вигляді діаграми послідовностей.
- Наведено та описано загальні поняття програмних технологій, що використовуються для розробки системи.
- Створено web-додаток на мові Python та за допомогою веб-фреймворку Flask. Написано програмний код для роботи системи аналізу та контролю переміщення міжобласного рейсового транспорту. Розроблено інтерфейс користувача з можливістю вибору маршруту та відображенням прогнозованого часу на мандрівку.

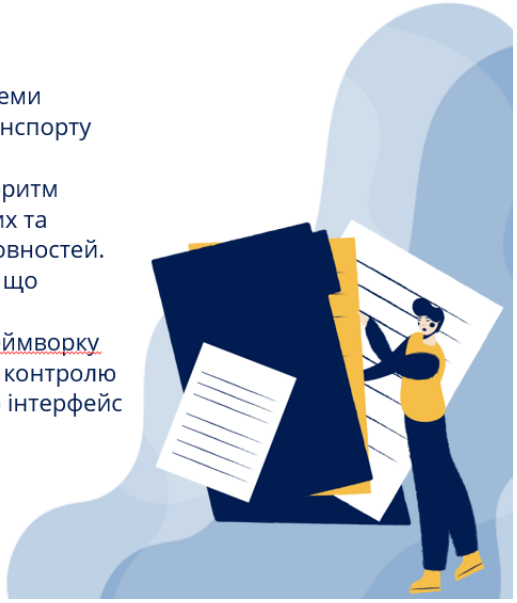


Рисунок А.12. Слайд 12

ДОДАТОК Б

Фрагмент коду програми

```
from flask import Flask, render_template, request,
jsonify, url_for, redirect
from flask_pymongo import PyMongo
from bson.objectid import ObjectId
import json
import os
from flask_httpauth import HTTPBasicAuth
from pymongo import MongoClient
app = Flask(__name__)
mongo = PyMongo(app)
bus_routes = mongo.db.bus_routes
trackings = mongo.db.trackings
trackers = mongo.db.trackers

# collection = mongo['trackings']
@app.route('/')
def index():
 bus_list = bus_routes.find()
return render_template('index2.html', bus_list=bus_list)
@app.route('/get_stations', methods=['GET','POST'])
def get_stations():
if request.method == 'POST':
bus_id = request.form.get('buses')

print("bus_routes_stations_id" + bus_id)
bus_routes_stations = bus_routes.find_one({'_id':
ObjectId(bus_id)})
print("bus_routes_stations" + str(bus_routes_stations))

bus_list = bus_routes.find()
print("bus_list" + str(bus_list[0]))
return render_template('index2.html',
bus_routes_stations=bus_routes_stations,
bus_list=bus_list, bus_id=bus_id)
else:
return redirect(url_for('index'))


def index_of(my_list, station):
 index = next((i for i, item in enumerate(my_list) if
item.id == station), -1)
 return (index)

def minutes_to_hours(time_in_minutes):
hours = time_in_minutes // 60
minutes = time_in_minutes % 60
 return "{} h {} min".format(hours, minutes)

@app.route('/get_tracking_info',
methods=['GET','POST'])
def get_tracking_info():
if request.method == 'POST':
bus_id = request.form.get('bus_id')
stations = request.form.get('stations')

bus_routes_stations = bus_routes.find_one({'_id':
ObjectId(bus_id)})["stations"]
print("bus_routes_stations" + str(bus_routes_stations))

bus_list = bus_routes.find()
print("bus_list" + str(bus_list[0]))

if(index_of(stations[0]) > index_of(stations[1])):
start = stations[1]
end = stations[0]
else:
start = stations[1]
end = stations[0]

find_near_bus_list = bus_routes_stations[:start]
 route_list = bus_routes_stations[start:end]

nearest_station = trackings.aggregate([{
"$elemMatch" :  find_near_bus_list}])

if(nearest_station["name"] == start):
stations_to_wait = []
else:
stations_to_wait =
bus_routes_stations[nearest_station:]

time_to_bus_came = 0
for station in range(len(stations_to_wait)-1):
time_to_bus_came +=
stations_to_wait[station].waiting_time
time_to_bus_came +=
stations_to_wait[station].drive_time

time_to_drive = 0
for station in range(len(route_list)-1):
time_to_drive += route_list[station].waiting_time
time_to_drive += route_list[station].drive_time
```

```
final_waiting_time = time_to_bus_came +
time_to_drive;
final_waiting_time =
minutes_to_hours(final_waiting_time)
time_to_drive = minutes_to_hours(time_to_drive)
time_to_bus_came =
minutes_to_hours(time_to_bus_came)


return render_template('index.html',
bus_routes_stations=bus_routes_stations,
final_waiting_time = final_waiting_time ,
bus_list=bus_list, time_to_bus_came =
time_to_bus_came,
time_to_drive = time_to_drive, start = start, end = end,
nearest_station = nearest_station )
else:
return redirect(url_for('index'))


@app.route('/find_bus1/<bus_id>')
def find_bus1(bus_id):
 todo_item = bus_routes.find_one({'_id':
ObjectId(bus_id)})
 # todo_item['complete'] = True
 # bus_routes.insert(todo_item)
bus_id = request.form.get('stations')
 print(bus_id)
   return redirect(url_for('index'))

@app.route('/find_bus', methods=['GET','POST'])
def find_bus():
 bus_id = request.form.get('stations')
 print(bus_id)
 return redirect(url_for('index'))

@app.route('/delete_all')
def delete_all():
todos.delete_many({})
return redirect(url_for('index'))


if __name__ == '__main__':
app.run(ssl_context='adhoc')

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible"
content="IE=edge">
<meta name="viewport" content="width=device-
width, initial-scale=1">
<title>ToDo App</title>

<!-- Bootstrap CSS -->
<link rel="stylesheet" href="{{ url_for('static',
filename='flatly.min.css') }}" />
<link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}" />

<!-- HTML5 Shim and Respond.js IE8 support of
HTML5 elements and media queries -->
<!-- WARNING: Respond.js doesn't work if you view
the page via file:// -->
<!--[if lt IE 9]>
<script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.2/html
5shiv.min.js"></script>
<script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/resp
ond.min.js"></script>
<![endif]-->
</head>
<body>

<div class="container-fluid" id="wrapper">
<div class="row">
<div class="col-lg-4 col-lg-offset-4" id="content">
<h2>CHOOSE YOUR BUS ROUTE</h2>
<form action="{{ url_for('get_stations') }}"
method="POST" role="form">
<div class="form-group">
<div class="input-group flex">

<select name="buses" class="form-control
bus_routes"  id="buses">
{% for bus in bus_list %}
<option value="{{ bus._id }}">
{{ bus.name }}| {{ bus.start }} - {{ bus.end }}
</option>
{% endfor %}
</select>
<span class="input-group-btn bus_routes-btn-wrap">
<button type="submit" class="btn btn-default
bus_routes-btn" id="add-btn">
SEARCH
</button>
</span>
</div>
</div>
```

```
</form>


{% if final_waiting_time %}
<div class="wraper">
<div class="route_name">Route: {{start}} - {{end}}
</div>
<div class="flex">
<div class="timetable">
<div class="timetable-main">
<div class="timetable-main-
number">{{final_waiting_time}} </div> min to
destination.
</div>
in <div class="timetable-main-number">
{{time_to_bus_came}} </div> min bus on  {{start}}
station in <div class="timetable-main-number">
{{time_to_drive}} </div> will get to the {{end}} from
{{start}}
</div>
<div class="routing">
On station: {{nearest_station}}
</div>
 </div>
</div>
{% endif %}

{% if bus_routes_stations %}
<form action="{{ url_for('get_tracking_infos') }}"
method="POST" id="routes_form" role="form">

<ul class="list-group t20">
{% for station in bus_routes_stations.stations %}
<li class="list-group-item {{'end-station' if station.type
== 'end' }} {{'start-station' if station.type == 'start' }}
station "
id="{{ station.name }}"
onClick="GFG_click(this.id)">
{{ station.name }}.
{% if station.type == 'mid' %}
 On station: {{ station.waiting_time }} min.
{% endif %}
</li>
{% endfor %}
</ul>

<input id="prodId" name="bus_id" type="hidden"
value="{{ bus_id }}">
<span class="input-group-btn btn-submit-rout">
<button type="submit" class="btn btn-default
bus_routes-btn" id="route-btn" >SUBMIT</button>
</span>
<img src="{{ bus_routes_stations.route_url }}"
class="img-map" style="width: 100%;">
</form>
{% endif %}

</div>
</div>

</div>
</body>
   <script>

function GFG_click(clicked) {
counter_route = start_and_end.length ;
var clicked_id = "#" + clicked
if(counter_route < 2 &&
!($(clicked_id).hasClass("chosed"))){
$(clicked_id).addClass("chosed");
start_and_end.push(clicked);
counter_route++;}
else if($(clicked_id).hasClass("chosed")){
$(clicked_id).removeClass("chosed");
removeItems(start_and_end, clicked)
counter_route--;}
console.log (counter_route)
if(counter_route >= 2 ){
document.getElementById('route-btn').disabled=false;
console.log("11");}
else{
document.getElementById('route-btn').disabled=true;
console.log("24");}
console.log(start_and_end)
console.log("ID = " + clicked);
}
```