НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004-047.36:629.434

«ПОГОДЖЕНО»

Декан факультету інформаційних технологій
Глазунова О.Г.,
д.пед.н.,професор

_____

« » _____2023 р

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»

Завідувач кафедри комп'ютерних наук
Голуб Б.Л., к.тех.н., доцент

_____

« » _____2023 р

МАГІСТЕРСЬКА РОБОТА

На тему: Система моніторингу громадським транспортом міста

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітньо-професійна програма Інформаційні управляючі системи та технології
(назва)

Робота на здобуття кваліфікації магістра

Керівник магістерської роботи
к.тех.н., доцент / Бородкіна І.Л. /
(вчене звання і ступінь) (підпис) (ПІБ)

Виконав _____ / Мазуренко А.О. /
(підпис) (ПІБ студента)

КИЇВ – 2023

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
к.тех.н., доцент Голуб Б.Л.
(вчене звання і ступінь) (підпис) (ініціали і прізвище)

«_____» _____ 2023 р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Мазуренко Андрія Олександровича
(прізвище,ім'я,по-батькові)

Спеціальність 122 Компютерні науки
(шифр і назва)

Освітньо-професійна програма Інформаційні управляючі системи та технології

Тема магістерської роботи: Система моніторингу громадським транспортом міста затверджено наказом ректора НУБіП від « 30 » грудня 2022 р. № 1940-С

Термін подання завершеної роботи на кафедру 5 листопада 2023 року
(рік, місяць, число)

Вихідні дані до магістерської роботи:

Розробка комплексної системи моніторингу громадським транспортом міста, яка включає в себе сучасні методи збору, обробки та аналізу даних.Перелік питань, що підлягають дослідженню:

1. Дослідження бізнес-процесів у сфері міського громадського транспорту.

2. Вибір методології та технічних рішень для системи моніторингу та управління.

3. Проектування та розробка комплексної системи моніторингу та управління міським громадським транспортом

4. Тестування та валідація розробленої системи моніторингу та управління

5. Дослідження та інтеграція методів та технологічних рішень для оптимізації міського громадського транспорту

6. Інтеграція декількох алгоритмів оптимізації для покращення ефективності маршрутів у системі громадського транспорту.

5. Перелік графічного матеріалу (за потребами): постер

Дата видачі завдання "17'' грудня 2022 р.
Керівник магістерської роботи Бородкіна І.Л. .
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання Мазуренко А.О..
(підпис) (прізвище та ініціали студента)

РЕФЕРАТ

Робота обсягом 111 сторінок, містить 30 рисунків, 17 таблиці та 30 літературних посилань.

В сучасному світі, з ростом урбанізації та насиченням міст транспортними засобами, актуальним стає питання ефективного моніторингу громадського транспорту. Важливість цього виражається не тільки в потребі забезпечення комфорту пасажирів, але і в забезпеченні безпеки, точності слідкування за маршрутами та оперативності управління транспортною інфраструктурою.

Метою даної роботи є розробка комплексної системи моніторингу громадським транспортом міста, яка включає в себе сучасні методи збору, обробки та аналізу даних. В роботі було проведено глибокий аналіз бізнес-процесів у сфері громадського транспорту, огляд сучасних технологічних рішень, а також методів забезпечення інформаційної безпеки.

Розроблена система базується на використанні сучасних технологій: від датчиків GPS для точного визначення геолокації до застосування машинного навчання для аналізу та прогнозування руху транспортних засобів. Система реалізована на мові програмування C# і використовує базу даних MS SQL Server для зберігання та обробки інформації.

Результати тестування та впровадження системи показали її високу ефективність: забезпечення точного слідкування за маршрутами, швидке реагування на зміни умов руху, а також надійну синхронізацію даних між серверною частиною та базою даних. Такий підхід може стати модельним для масштабування системи на рівень регіону або країни.

КЛЮЧОВІ СЛОВА:

ГРОМАДСЬКИЙ ТРАНСПОРТ, МОНІТОРИНГ, УРБАНІЗАЦІЯ, БЕЗПЕКА, МАШИННЕ НАВЧАННЯ, GPS, C#, MS SQL SERVER, ТЕХНОЛОГІЧНІ РІШЕННЯ, ІНФОРМАЦІЙНА БЕЗПЕКА.

ЗМІСТ

ВСТУП

У сучасному світі транспортна інфраструктура відіграє ключову роль у соціально-економічному розвитку країн та регіонів. Громадський транспорт є одним із найважливіших компонентів транспортної системи, який безпосередньо впливає на якість життя громадян. В Україні, як і в багатьох інших країнах, система громадського транспорту стикається з рядом проблем: нерегулярність руху, перевантаженість, відсутність чіткої інформації про розклади та маршрути, а також проблеми екології та енергоефективності.

З урахуванням зростаючого обсягу пасажироперевезень, актуальність створення ефективної системи моніторингу громадським транспортом міста є безсумнівною. Це дозволить не тільки підвищити рівень комфорту пасажирів, але і забезпечити більш ефективне використання ресурсів, зокрема пального, що є актуальною проблемою для економіки України.

Розвиток інформаційних технологій відкриває широкі можливості для імплементації автоматизованих систем моніторингу, що базуються на використанні GPS-трекінгу, IoT-сенсорів, Big Data та інших сучасних технологій. Хоча єдині стандарти для таких систем ще не сформовані, вже існують вдалі приклади їх впровадження в ряді розвинутих країн.

Шляхом критичного аналізу існуючих рішень можна констатувати, що більшість з них або не призначені для масштабування, або фокусуються на розв'язанні лише окремих аспектів проблеми. Таким чином, є актуальною потреба в комплексному підході до створення системи моніторингу громадського транспорту, яка би враховувала специфіку українських міст, і була б зорієнтована на максимальну ефективність та доступність.

Важливо підкреслити, що успішна реалізація такої системи може мати позитивний вплив на транспортну безпеку, екологічну ситуацію, а також може стати каталізатором для розвитку смарт-міст в Україні. Враховуючи стратегічні напрямки розвитку нашої країни, зокрема в контексті інтеграції з Європейським Союзом, це дослідження є не тільки актуальним, але й

необхідним для подальшого прогресу в області транспортних технологій та урбаністики.

Мета роботи полягає у розробці комплексної системи для моніторингу та управління міським громадським транспортом з метою оптимізації транспортних потоків, підвищення рівня безпеки та комфорту пасажирів, а також інтеграції сучасних технологічних рішень.

Виходячи із поставленої мети, можна виділити такі основні задачі:

– дослідження бізнес-процесів предметної області. Вивчення основних процесів діяльності міського громадського транспорту, акторів та їхніх функцій, а також структури основних бізнес-процесів;

– вибір методології та технічних рішень. Оцінка та вибір платформ для розробки, мов програмування, систем управління базами даних, а також аналіз вимог та моделювання прецедентів;

– проектування та розробка системи. Створення структури бази даних, розробка алгоритмів та програмного коду, реалізація серверної частини, конфігурація та оптимізація системи;

– тестування та валідація. Проведення комплексних тестів для перевірки функціональності, надійності та ефективності системи, а також аналіз отриманих результатів для подальшої оптимізації.

Кожна з цих задач є складовою частиною загальної мети і вимагає детального дослідження та практичної реалізації. Виконання цих задач дозволить досягнути поставленої мети та забезпечить успішну імплементацію системи моніторингу та управління міським громадським транспортом.

Об'єктом даного дослідження є система міського громадського транспорту, що включає в себе комплекс взаємопов'язаних елементів: транспортні засоби, інфраструктуру, пасажирські потоки, а також механізми управління та моніторингу. Ця система породжує ряд проблемних ситуацій, таких як перевантаженість, неефективність маршрутів, проблеми з безпекою пасажирів та екологічні виклики, які стають предметом дослідження.

Предметом дослідження є методи та технологічні рішення для моніторингу та управління міським громадським транспортом. Конкретно, це включає в себе алгоритми оптимізації маршрутів, механізми для забезпечення безпеки пасажирів, технічні засоби для моніторингу стану транспортних засобів, а також розробка програмного забезпечення для централізованого управління та моніторингу.

З метою досягнення конкретних задач, поставлених в дослідженні було обрано наступні методи дослідження:

– аналітичний метод. Використовувався для дослідження існуючих систем моніторингу та управління міським громадським транспортом, а також для аналізу бізнес-процесів у предметній області. Цей метод допоміг у виявленні ключових проблем, які потребують рішення;

– експериментальний метод. Використовувався для тестування прототипу системи з метою виявлення можливих помилок, оцінки ефективності та надійності розробленої системи;

– статистичний метод. Застосовувався для аналізу отриманих в ході експерименту даних. Статистичний аналіз дозволив зробити висновки про ефективність та надійність системи, а також ідентифікувати аспекти, що потребують подальшої оптимізації.

– метод критичного аналізу. Використовувався для огляду схожих на розроблювану систему рішень, а також для оцінки методів забезпечення безпеки в системах управління громадським транспортом.

Наукова новизна одержаних результатів полягає в інтеграції декількох алгоритмів оптимізації для досягнення більшої ефективності маршрутів, що є відмінним від традиційних однокритеріальних методів.

1 ПРОЕКТНІ АСПЕКТИ СТВОРЕННЯ КОМПЛЕКСУ ДЛЯ МОНІТОРИНГУ ТА УПРАВЛІННЯ МІСЬКИМ ГРОМАДСЬКИМ ТРАНСПОРТОМ

1.1 Дослідження бізнес-процесів предметної області

Сучасні міста стикаються з низкою викликів у сфері громадського транспорту, серед яких перевантаженість, недостатній рівень безпеки, а також неефективність маршрутів. Розробка комплексної системи моніторингу та управління міським громадським транспортом вимагає глибокого розуміння бізнес-процесів, які лежать в основі цієї діяльності. Відсутність такого розуміння може призвести до створення системи, що не враховує конкретні потреби та особливості діяльності перевізників, органів управління та пасажирів. Тому першочерговим завданням є детальний аналіз та дослідження ключових бізнес-процесів у даній предметній області. Це дозволить в подальшому вибудувати систему, яка максимально відповідає потребам всіх зацікавлених сторін.

1.1.1 Основні процеси діяльності

Основні бізнес-процеси в сфері міського громадського транспорту відіграють ключову роль у забезпеченні ефективної, безпечної та комфортної перевезення пасажирів. Ці процеси охоплюють планування маршрутів, управління транспортними засобами, забезпечення безпеки пасажирів та персоналу, а також моніторинг і контроль якості обслуговування. Успішна організація цих елементів є важливим фактором для підтримки високого рівня сервісу та забезпечення стабільності транспортної системи міста.

Нижче детально розглянуто кожен з основних процесів:

Планування маршрутів та графіків

Планування маршрутів та графіків здійснюється на основі глибокого аналізу даних, які включають інформацію про пасажиропотоки, часові рамки найбільшої пасажирської активності, потреби жителів в транспортних послугах, а також характеристики та стан транспортної інфраструктури. Цей

процес не лише допомагає в оптимізації маршрутів для найбільш ефективного використання транспортних засобів, але також відіграє важливу роль у зниженні витрат пального, зменшенні часу в дорозі для пасажирів і, в кінцевому рахунку, у поліпшенні якості життя місцевих жителів.

Особлива вага приділяється створенню таких графіків руху транспортних засобів, які б максимально відповідали потребам пасажирів, забезпечуючи при цьому ефективне використання транспортних ресурсів. У цьому контексті проводиться постійний моніторинг пасажиропотоків, що дозволяє швидко реагувати на зміни в пасажирській активності та забезпечує гнучкість у плануванні маршрутів та графіків [1].

Управління транспортними засобами

Управління транспортними засобами є складним і багатоаспектним процесом, що має безпосередній вплив на роботу всієї транспортної системи міста. Цей процес включає планування та розподіл транспортних ресурсів, що охоплює визначення необхідної кількості транспортних засобів для кожного маршруту, розробку графіків їх технічного обслуговування та встановлення критеріїв для заміни старого транспорту новим.

Технічний контроль та обслуговування транспортних засобів є іншим важливим аспектом, який безпосередньо впливає на безпеку пасажирів та персоналу. Систематичний технічний огляд, планове та непланове ремонтні роботи, а також перевірка всіх систем безпеки є необхідними компонентами ефективного управління.

Крім того, економічна ефективність управління парком транспортних засобів тісно пов'язана з оптимізацією використання ресурсів, зокрема пального, запчастин та робочого часу персоналу. Це вимагає від організацій, що управляють громадським транспортом, впровадження сучасних методів моніторингу та аналізу даних для постійної оптимізації робочих процесів [2].

Забезпечення безпеки пасажирів та персоналу

Забезпечення безпеки пасажирів та персоналу налічує не тільки технічні, але і організаційні аспекти. Технічні аспекти перш за все включають регулярне

та якісне обслуговування транспортних засобів, перевірку робочого стану гальмівних систем, сигналізації та інших критичних компонентів. Важливим є впровадження сучасних систем безпеки, таких як автоматичні системи контролю швидкості, системи відеоспостереження в салонах та на зупинках, а також системи раннього виявлення аварійних ситуацій.

Організаційні аспекти зосереджуються на підготовці персоналу та їх постійному професійному розвитку. Водії проходять спеціалізовані курси з безпеки дорожнього руху, а також регулярно піддаються медичним перевіркам та моніторингу їхнього фізичного та психоемоційного стану. Це дозволяє вчасно виявляти можливі проблеми, що можуть вплинути на безпеку пасажирів.

Крім того, акцентується увага на розробці та впровадженні протоколів реагування на надзвичайні ситуації, що включає в себе не лише швидке реагування служб безпеки, але і готовність пасажирів до дій в критичних ситуаціях. Всі ці заходи спрямовані на мінімізацію ризиків та забезпечення високого рівня безпеки пасажирів та персоналу [3].

Забезпечення безпеки пасажирів та персоналу

Забезпечення безпеки пасажирів та персоналу є важливою складовою управління міським громадським транспортом. З одного боку, великий наголос кладеться на технічне обслуговування транспортних засобів. Регулярні технічні перевірки, а також своєчасний ремонт критичних систем — гальм, рульового управління, сигналізації — спрямовані на мінімізацію ризику аварійних ситуацій.

З іншого боку, організаційні аспекти не менш важливі. Персонал, зокрема водії, проходять спеціалізовану підготовку з питань безпеки дорожнього руху. Моніторинг стану водіїв включає регулярні медичні перевірки та оцінку їх психофізіологічного стану, що дозволяє виявити та вчасно усунути фактори, що можуть негативно вплинути на безпеку перевезень.

Також важливим є впровадження систем відеоспостереження не тільки всередині транспортних засобів, але і на зупинках, що дозволяє контролювати ситуацію в реальному часі та реагувати на надзвичайні ситуації.

Всі ці заходи спільно створюють комплексну систему, що має на меті забезпечити високий рівень безпеки пасажирів та персоналу в умовах постійної динаміки та змінності умов експлуатації міського громадського транспорту [4].

Моніторинг та контроль якості перевезень

Моніторинг та контроль якості перевезень в управлінні міським громадським транспортом забезпечують систематичність і прозорість всіх процесів. Постійний збір даних про рух транспортних засобів створює не лише можливість детального відслідковування маршрутної діяльності, але і формує оперативний інструментарій для швидкого реагування на непередбачені обставини. Це може бути затор на дорозі, аварія або інші надзвичайні ситуації, які вимагають негайних дій.

Систематичний аналіз заповненості транспортних засобів стає ключовим для раціонального планування маршрутів та графіків руху. Інформація про заповненість дозволяє не лише оптимізувати кількість транспортних засобів на конкретних ділянках, але і зменшує час очікування пасажирів на автобусних зупинках та станціях метро.

Дотримання графіків руху транспорту є ще одним аспектом, який підлягає постійному контролю. Відхилення від розкладу може привести до перевантаження ділянок маршруту, що в кінцевому результаті призведе до зниження якості обслуговування та загальної задоволеності пасажирів.

Однією з ключових компонентів в системі контролю якості є збір зворотного зв'язку від пасажирів. Аналіз відгуків та скарг забезпечує важливу інформацію про слабкі місця в роботі транспортної системи. Це, в свою чергу, надає можливість для внесення коректив у планування та експлуатацію транспортних засобів, що спрямовано на забезпечення високої якості перевезень.

Кожен з цих процесів є взаємопов'язаним і має свою специфіку, яка повинна бути врахована при розробці комплексної системи моніторингу та управління міським громадським транспортом. Розуміння цих процесів є критично важливим [5].

1.1.2 Актори і їхні функції

В системі моніторингу громадським транспортом міста можна виділити кілька ключових акторів, кожен з яких відіграє свою специфічну роль в організації та контролі перевезень.

– водії громадського транспорту. Основна функція цієї групи полягає в безпосередньому управлінні транспортними засобами. Вони відповідають за дотримання маршрутів, графіків, а також за безпеку пасажирів під час перевезення;

– диспетчери. Ці спеціалісти координують роботу транспортних засобів у реальному часі. Вони мають доступ до систем моніторингу і в разі потреби можуть коригувати маршрути для оптимізації руху;

– технічний персонал. Ця група включає механіків, інженерів та інших спеціалістів, які відповідають за технічний стан транспортних засобів, їх обслуговування та ремонт;

– аналітики та планувальники. Ці спеціалісти займаються аналізом даних з системи моніторингу, плануванням оптимальних маршрутів та графіків, а також оцінкою ефективності системи в цілому;

– пасажири. Хоча пасажири не є безпосередньо включені в систему управління, їх роль не можна недооцінювати. Вони є основними користувачами системи та джерелом важливого зворотнього зв'язку для подальшого її вдосконалення;

– органи місцевого самоврядування. Ці інституції зазвичай фінансують та регулюють діяльність громадського транспорту, встановлюють правила та нормативи, і вони зацікавлені в ефективності та надійності системи моніторингу.

Кожен з акторів взаємодіє з системою моніторингу на різних рівнях і в різних аспектах, формуючи складну, але гармонійну структуру, що забезпечує ефективність та надійність перевезень у громадському транспорті [6].

1.1.3 Структура основних бізнес-процесів

Структура бізнес-процесів в системі моніторингу громадським транспортом міста відображає її багатоаспектну природу та різноманітність завдань, які необхідно вирішувати для забезпечення ефективної діяльності. Основні елементи цієї структури можна поділити на чотири основні блоки:

– планування та координація. Цей блок включає в себе процеси аналізу пасажиропотоків, планування маршрутів та графіків, а також розподіл транспортних ресурсів. Він є виходом для всіх подальших дій і залучає в себе аналітиків, планувальників та органи місцевого самоврядування;

– операційне управління. У цьому блоку реалізуються процеси безпосереднього управління транспортними засобами. Це включає в себе роботу диспетчерів, водіїв та технічного персоналу, які координують рух транспорту, забезпечують його технічний стан та реагують на надзвичайні ситуації;

– моніторинг та контроль. В цьому сегменті фокусується на зборі, аналізі та використанні даних для оцінки ефективності системи. Процеси моніторингу дотримання графіків, збору зворотного зв'язку від пасажирів та аналізу заповненості транспортних засобів відіграють ключову роль в цьому блоку;

– забезпечення безпеки. Ключовими процесами в цьому блоку є обслуговування та ремонт транспортних засобів, підготовка персоналу, системи відеоспостереження та інші технічні та організаційні міри для забезпечення безпеки пасажирів та персоналу.

Всі ці блоки не ізольовані один від одного, а взаємопов'язані через ряд ключових процесів та взаємодій між акторами (рис. 1.1). Така структура забезпечує гнучкість та адаптивність системи, що є критично важливим для

ефективного управління складним і динамічним середовищем управління громадським транспортом.

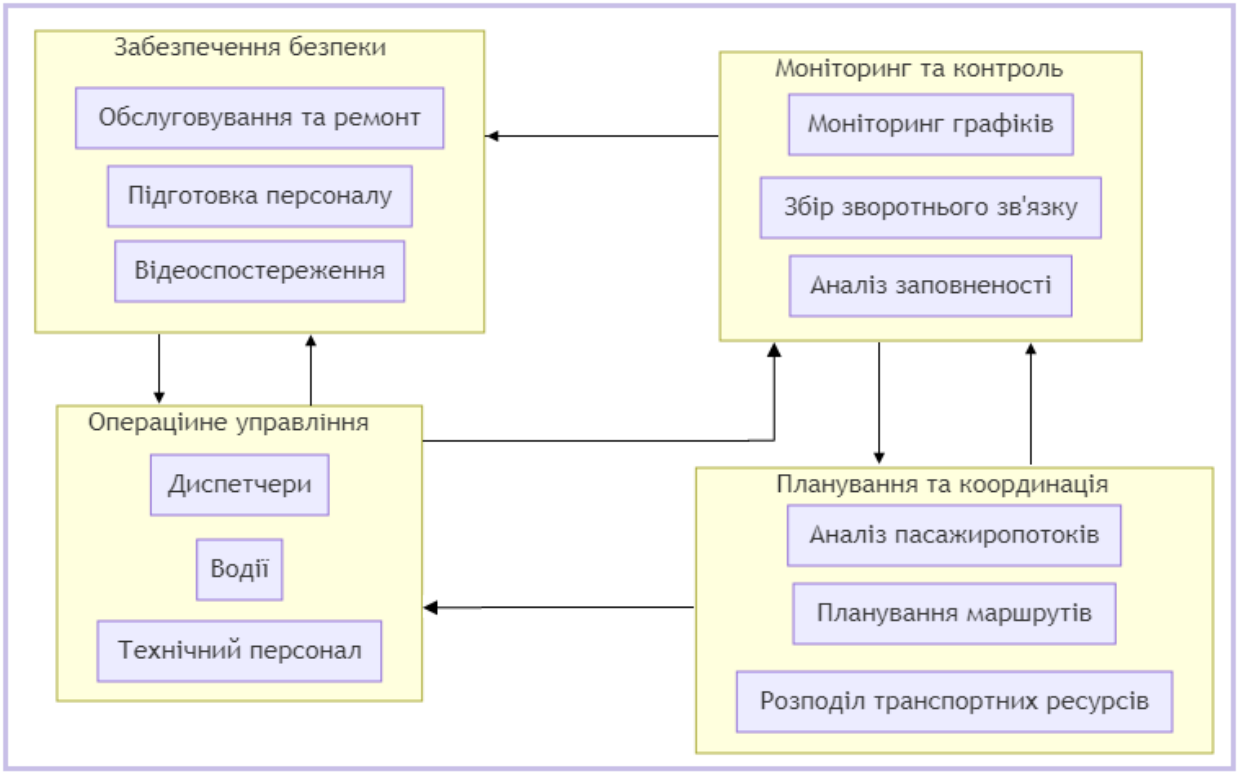

Рисунок 1.1 – Схема взаємодії основних бізнес-процесів

Взаємодія між основними блоками структури бізнес-процесів в системі моніторингу громадським транспортом відбувається наступним чином:

– планування та координація. Цей блок забезпечує аналіз пасажиропотоків, планування маршрутів та графіків, а також розподіл транспортних ресурсів. Він взаємодіє з блоком «Операційне управління» для передачі інформації про плановані маршрути та графіки. Також цей блок може користуватися даними з блоку «Моніторинг та контроль» для корекції планів;

– операційне управління. Цей блок зосереджується на безпосередньому управлінні транспортними засобами. Він взаємодіє з блоком «Планування та координація» для отримання планованих маршрутів та графіків. Цей блок також може надсилати дані про реальний стан руху та надзвичайні ситуації до блоку «Моніторинг та контроль»;

– моніторинг та контроль. Цей блок зосереджується на зборі, аналізі та використанні даних для оцінки ефективності системи. Він може отримувати

дані від блоку «Операційне управління» та взаємодіє з блоком «Планування та координація» для корекції планів на основі зібраних даних;

– забезпечення безпеки. Цей блок фокусується на технічних та організаційних мірах для забезпечення безпеки пасажирів та персоналу. Він може взаємодіяти з блоком «Операційне управління» для реагування на надзвичайні ситуації та з блоком «Моніторинг та контроль» для аналізу даних про безпеку.

Ці блоки не ізольовані один від одного, а взаємопов'язані через ряд ключових процесів та взаємодій між акторами. Така структура забезпечує гнучкість та адаптивність системи, що є критично важливим для ефективного управління складним і динамічним середовищем управління громадським транспортом[7].

1.2 Огляд схожих на розроблювану систему рішень

В сучасних міських умовах, системи моніторингу громадського транспорту є не лише актуальними, але й невід'ємними елементами ефективного управління транспортними потоками.

GeoLocate Public Transport

Система «GeoLocate Public Transport» розроблена з метою моніторингу та управління громадським транспортом в реальному часі. Ця система спрямована на підвищення ефективності транспортних послуг, зменшення часу очікування для пасажирів та оптимізацію маршрутів.

Основні функції системи:

– моніторинг в реальному часі. Відображення актуального місцезнаходження транспортних засобів;

– аналіз маршрутів. Оптимізація маршрутів на основі аналізу даних;

– інформаційні повідомлення. Автоматична відправка повідомлень пасажирам про зміни в графіку;

– звітність та аналітика. Генерація звітів для аналізу ефективності системи.

На рис. 1.2 представлено інтерфейс користувача системи GeoLocate Public Transport.

Рисунок 1.2 – Інтерфейс користувача «GeoLocate Public Transport»

Ця система є комплексним рішенням, що включає в себе ряд технологій та функцій для ефективного моніторингу та управління громадським транспортом.

Переваги системи включають:

– точність моніторингу. Використання GPS-технологій забезпечує високу точність відстеження місцезнаходження транспортних засобів;

– оптимізація маршрутів. Система дозволяє динамічно оптимізувати маршрути на основі зібраних даних, що підвищує ефективність транспортної системи;

– зручність для пасажирів. Мобільний додаток надає корисну інформацію для пасажирів, зокрема про час прибуття транспорту;

– аналітика та звітність. Інтегровані інструменти для аналізу даних та генерації звітів.

Недоліки:

– вартість установки та обслуговування. Встановлення GPS-модулів та хмарного сховища може бути коштовним;

– залежність від інтернет-з'єднання. Для ефективної роботи системи необхідне стабільне інтернет-з'єднання, що може бути проблематичним в деяких регіонах;

– проблеми з конфіденційністю. Збір та аналіз геолокаційних даних може порушувати питання конфіденційності.

Отже, система «GeoLocate Public Transport» є комплексним рішенням для моніторингу та управління громадським транспортом. Завдяки використанню GPS-технологій та хмарних рішень, система забезпечує високу точність моніторингу та можливість динамічної оптимізації маршрутів. Мобільний додаток робить систему зручною для кінцевих користувачів, надаючи їм актуальну інформацію про рух транспорту.

Однак, система має деякі недоліки, такі як висока вартість установки та потенційні проблеми з конфіденційністю. Загалом, «GeoLocate Public Transport» може бути ефективним рішенням для міст і регіонів, які прагнуть модернізувати свою транспортну інфраструктуру та підвищити якість обслуговування пасажирів [8].

TransitTrack

Система «TransitTrack» розроблена для моніторингу громадського транспорту в реальному часі з метою підвищення ефективності та комфорту пасажирських перевезень (рис. 1.3). Система спрямована на зменшення часу очікування пасажирів та поліпшення координації між різними видами транспорту.

Основні функції системи включають:

– моніторинг в реальному часі. Відображення актуального місцезнаходження транспортних засобів за допомогою GPS;

– інтеграція з розкладами. Система автоматично синхронізується з офіційними розкладами, надаючи точну інформацію про час прибуття;

– повідомлення для пасажирів. Автоматична відправка повідомлень про зміни в розкладі, відхилення від маршруту або надзвичайні ситуації;

– аналітичні інструменти. Збір та аналіз даних про пасажиропотік, заповненість транспорту та інші ключові показники.

Система «TransitTrack» є ефективним інструментом для управління громадським транспортом, який може значно поліпшити якість обслуговування пасажирів та оптимізувати роботу транспортної інфраструктури.

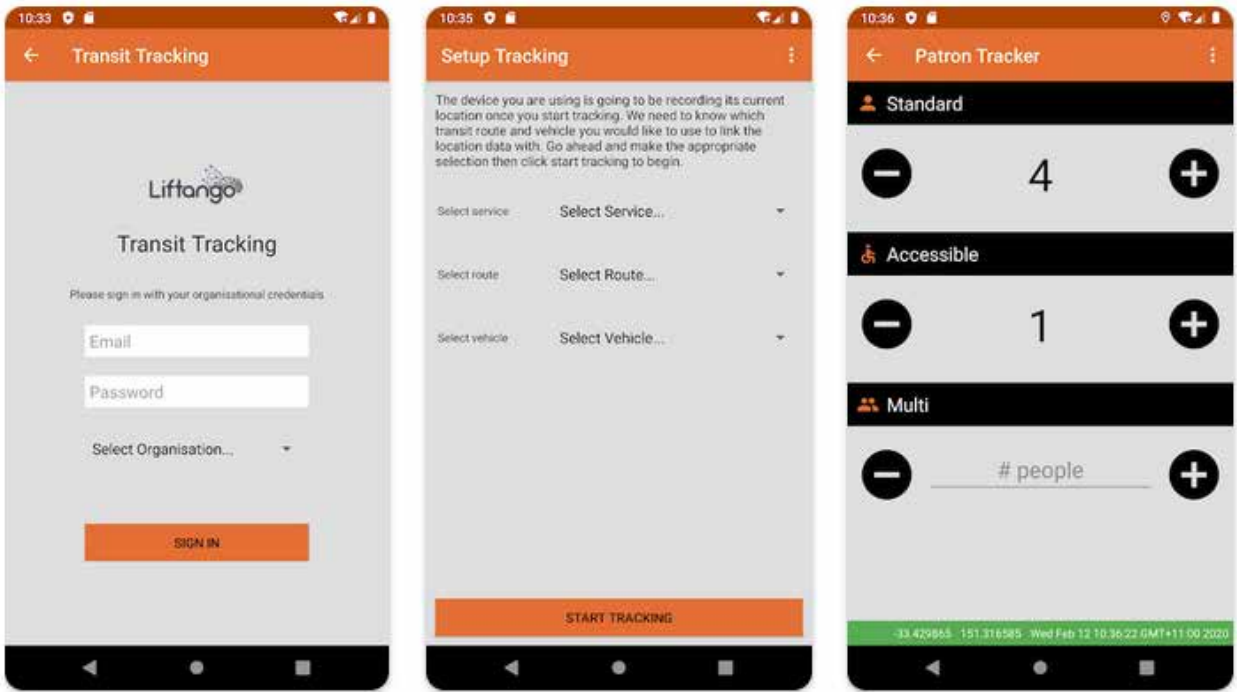

Рисунок 1.3 – Інтерфейс користувача «TransitTrack»

Переваги системи складають:

– інтеграція з розкладами. Автоматична синхронізація з офіційними розкладами забезпечує точність і актуальність інформації для пасажирів;

– багатофункціональність. Система не тільки відстежує місцезнаходження транспорту, але і забезпечує аналітичні інструменти для оптимізації роботи;

– комунікація з пасажирами. Можливість автоматичної відправки повідомлень про зміни в розкладі підвищує рівень сервісу;

– гнучкість. Система може бути адаптована для різних видів громадського транспорту, включаючи автобуси, трамваї, метро.

До недоліків можна віднести:

– залежність від технічної інфраструктури. Наявність GPS-модулів та стабільного інтернет-з'єднання є обов'язковою;

– складність налаштування. Інтеграція з існуючими розкладами та системами може бути трудомісткою;

– вартість. Ліцензійні витрати та витрати на обслуговування можуть бути високими, залежно від обсягу та функціональності системи.

Отже, система «TransitTrack» представляє собою інтегроване рішення для моніторингу та управління громадським транспортом. Завдяки широкому спектру функцій — від відстеження місцезнаходження транспортних засобів до аналітичних інструментів — система може значно підвищити ефективність та якість обслуговування пасажирів. Однак, необхідно враховувати технічні та фінансові аспекти при її впровадженні. Зокрема, висока вартість та потреба в стабільному інтернет-з'єднанні можуть бути обмежуючими факторами. Загалом, «TransitTrack» є перспективним рішенням для сучасних систем громадського транспорту [9].

Moovit

Moovit є універсальним мобільним додатком для планування подорожей на громадському транспорті (рис. 1.4). Цей застосунок призначений для широкого спектру користувачів, від мешканців великих міст до туристів, і має на меті полегшити навігацію та оптимізувати використання громадського транспорту.

Функції даного додатку складають:

– розклади та маршрути. Додаток надає актуальні розклади та оптимальні маршрути для різних видів громадського транспорту;

– живий моніторинг. Відстеження місцезнаходження транспортних засобів в реальному часі;

– сповіщення. Автоматичні повідомлення про зміни в розкладі, затримки, або надзвичайні ситуації;

– інтерактивна карта. Карта міста з позначеннями зупинок, станцій та інших ключових точок;

– персональні налаштування. Збереження улюблених маршрутів та зупинок для швидкого доступу;

– соціальні функції. Можливість ділитися інформацією та зворотнім зв'язком з іншими користувачами.

Також Moovit є одним з найбільш популярних та функціональних додатків для громадського транспорту, який враховує потреби різних категорій користувачів та надає комплексний набір інструментів для ефективної навігації.

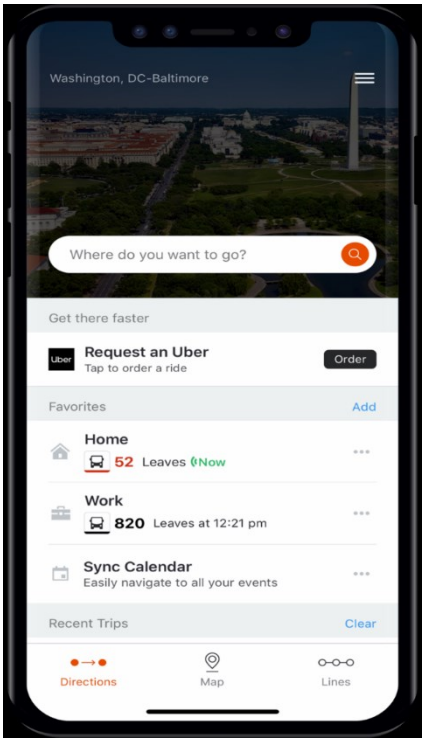

Рисунок 1.4 – Інтерфейс користувача «Moovit»

Переваги додатку «Moovit» становлять:

– універсальність. Підтримка великої кількості міст та видів громадського транспорту;

– інтуїтивний інтерфейс. Зручний та зрозумілий дизайн, який полегшує використання додатку;

– реальний моніторинг. Можливість відстежувати місцезнаходження транспорту в реальному часі;

– персональні налаштування. Збереження улюблених маршрутів та зупинок для швидкого доступу.

– соціальні функції. Інтеграція з соціальними мережами та можливість обміну інформацією.

До недоліків «Moovit» відносять:

– обмежений функціонал в офлайн режимі. Більшість функцій додатку недоступні без Інтернет-з'єднання, що може бути критичним у певних ситуаціях;

– комплексність інтерфейсу. Для нових користувачів може бути складно зрозуміти всі можливості додатку, що ускладнює його використання;

– точність даних. Іноді може бути невірна інформація через зміни в розкладах або технічні проблеми;

– реклама. В безкоштовній версії присутній рекламний контент, який може бути незручним для користувача;

– споживання ресурсів. Додаток може бути витратним з точки зору батареї та мобільних даних.

Отже, Moovit є високоефективним та універсальним додатком для планування та моніторингу подорожей на громадському транспорті. Завдяки широкому спектру функцій, він забезпечує користувачам гнучкість та зручність при виборі оптимальних маршрутів. Однак, необхідність постійного інтернет-з'єднання та наявність реклами в безкоштовній версії можуть стати незначними недоліками. Загалом, даний додаток є цінним інструментом для ефективного використання громадського транспорту [10].

Після аналітичного огляду ринку програмних рішень для моніторингу громадського транспорту, можна констатувати, що кожна з розглянутих систем має свій набір сильних сторін та обмежень. Для зручності порівняння та об'єктивної оцінки цих характеристик, було створено зведену таблицю аналізу (табл. 1.1), яка відображає ключові параметри кожного з розглянутих програмних продуктів.

Таблиця 1.1 – Аналіз аналогічних програмних рішень

Опції	GeoLocate Public Transport	TransitTrack	Moovit	Розроблене ПЗ
Інтерфейс програми	Спрощений	Комплексний	Інтуїтивний	Інтуїтивний
Функціонал	Широкий	Широкий	Широкий	Необхідний для більшості користувачів
Технічні вимоги	Низькі	Середні	Високі	Низькі
Вартість	Умовно безкоштовний	Умовно безкоштовний	Безкоштовний/ Платний без реклами	Безкоштовний
Українська локалізація	Немає	Немає	є	є

Виходячи із проведеного аналізу аналогічних програмних рішень для моніторингу громадського транспорту, можна зробити висновок, що кожна з існуючих систем має свої переваги та недоліки. Однак, розроблене програмне забезпечення не лише відповідає ключовим потребам користувачів, але й пропонує ряд унікальних характеристик, які виділяють його на тлі конкурентів. Зокрема, це стосується таких функціональних можливостей:

– інтуїтивний інтерфейс з фокусом на потребах користувача. Розроблене ПЗ буде мати інтуїтивний інтерфейс, який буде зосереджений на функціоналі, необхідному для більшості користувачів. Це дозволить забезпечити високий рівень користувацького досвіду;

– оптимізований функціонал. Замість широкого спектру функцій, ПЗ буде зосереджено на ключових потребах користувачів, що зробить його більш ефективним і менш перевантаженим;

– низькі технічні вимоги. ПЗ буде працювати на пристроях з низькими технічними характеристиками, що робить його доступним для широкого кола користувачів;

– повна українська локалізація. Розроблене ПЗ буде повністю локалізоване для українських користувачів, що є важливим фактором для популяризації продукту на вітчизняному ринку;

– безкоштовність. ПЗ буде повністю безкоштовним, що зробить його більш привабливим для користувачів в порівнянні з аналогами, які пропонують умовно безкоштовні або платні версії.

Всі ці унікальні характеристики роблять розроблене в рамках роботи ПЗ не просто альтернативою існуючим рішенням, а продуктом, який враховує специфіку та потреби українського ринку та користувачів.

1.3 Дослідження методів забезпечення безпеки систем управління громадським транспортом

Система моніторингу громадського транспорту надає можливість збору, обробки та аналізу великої кількості даних в реальному часі. Однак це також ставить питання про безпеку передачі даних від транспортних засобів до центральної бази даних. Для цього необхідно розглянути методи забезпечення безпеки передачі даних:

Протокол HTTPS з двосторонньою аутентифікацією

В даному варіанті використовується протокол HTTPS (HTTP Secure), який є розширенням стандартного протоколу HTTP з додаванням підтримки шифрування на основі протоколу SSL/TLS. Для додаткової безпеки застосовується двостороння аутентіфікація за допомогою цифрових сертифікатів, яка забезпечує не тільки шифрування даних, але і перевірку автентичності обох учасників комунікації.

Застосування протоколу HTTPS з двосторонньою аутентифікацією на основі сертифікатів є однією з надійних технік забезпечення безпеки при передачі даних. В цьому контексті, двостороння аутентифікація означає, що як

сервер, так і клієнт (в нашому випадку, транспортний засіб) підтверджують свою легітимність за допомогою цифрових сертифікатів.

Використання HTTPS забезпечує шифрування даних на основі протоколу SSL/TLS, що робить неможливим їх перехоплення або модифікацію третіми особами. Двостороння аутентифікація на основі сертифікатів додає ще один рівень безпеки, змушуючи обидві сторони підтвердити свою аутентичність перед встановленням з'єднання.

Процес встановлення з'єднання починається із запиту транспортного засобу до сервера і обміну сертифікатами для аутентифікації. Після успішної аутентифікації і встановлення захищеного каналу для комунікації, транспортний засіб може безпечно передавати дані до сервера і навпаки.

Цей метод є особливо корисним для систем, де вимоги до безпеки є високими, і де необхідно забезпечити конфіденційність, цілісність, а також аутентифікацію обох учасників комунікації [11].

VPN з IPsec для захищеного тунелю

VPN (Virtual Private Network) з протоколом IPsec (Internet Protocol Security) створює захищений тунель між транспортним засобом і сервером бази даних. Це забезпечує конфіденційність, аутентифікацію, а також захист від атак типу «людина посередник» (Man-in-the-Middle).

Встановлення VPN з'єднання з використанням IPsec є ефективним методом для забезпечення безпеки даних, які передаються між транспортним засобом та сервером. В даному випадку, VPN служить для створення віртуального захищеного каналу між двома точками, через який може безпечно передаватися інформація.

IPsec, як частина цього рішення, забезпечує кілька ключових аспектів безпеки. По-перше, він шифрує дані, які передаються по VPN-з'єднанню, забезпечуючи конфіденційність інформації. Це означає, що навіть якщо зловмисник вдасться перехопити пакети даних, він не зможе їх розшифрувати без відповідних ключів. По-друге, IPsec забезпечує цілісність даних, гарантуючи, що інформація не була змінена під час передачі. По-третє, IPsec

дозволяє аутентифікувати джерело даних, переконуючись у тому, що інформація дійсно надійшла від легітимного відправника.

Процес встановлення VPN з'єднання з IPsec включає в себе підняття VPN-сервера, конфігурацію параметрів безпеки та розподіл ключів між учасниками з'єднання. Після встановлення з'єднання, транспортний засіб може безпечно передавати дані до сервера, використовуючи цей захищений тунель. Таким чином, VPN з IPsec не лише забезпечує конфіденційність та цілісність даних, але і дозволяє забезпечити високий рівень аутентифікації учасників комунікації [12-13].

JWT (JSON Web Tokens)

Один із сучасних методів захисту передачі даних полягає у використанні JWT (JSON Web Tokens), який активно використовується для обміну аутентифікованою інформацією між різними системами.

JWT складається з трьох основних частин: заголовка, видачі та підпису. Заголовок і видача містять метадані та інформацію, яка необхідна для ідентифікації та аутентифікації користувача або системи. Цифровий підпис забезпечується за допомогою алгоритму HMAC, який гарантує, що токен не був змінений під час передачі та що він було створено легітимною стороною.

В системах моніторингу громадського транспорту, JWT токен може бути згенерований прямо на транспортному засобі після його успішної аутентифікації. Після генерації, токен підписується за допомогою алгоритму HMAC, використовуючи секретний ключ, який зберігається як на транспортному засобі, так і на сервері бази даних. Після підпису, JWT токен відправляється на сервер разом з необхідними даними. На сервері підпис токену перевіряється, і якщо він вірний, дані обробляються відповідно [14].

Отже, використання JWT з HMAC підписом дозволяє не тільки забезпечити конфіденційність та цілісність даних, але і забезпечити високий рівень аутентифікації транспортного засобу, що є важливим для забезпечення безпеки всієї системи моніторингу громадського транспорту.

Кожен із цих методів має свої переваги і недоліки, а вибір конкретного методу залежить від специфічних вимог до безпеки, доступності ресурсів та інших факторів.

При розробці системи моніторингу громадським транспортом міста ключовими критеріями для вибору методу забезпечення безпеки є ефективність, швидкість обробки, гнучкість, а також здатність працювати на різноманітному апаратному забезпеченні. Для реалізації системи було прийняте рішення використовувати JWT, так як він забезпечує:

– ефективність ресурсів. JWT є відносно легким методом, що не вимагає великих обчислювальних потужностей для генерації та валідації токенів. Це особливо важливо для вбудованих систем, таких як контролери на транспортних засобах, які можуть мати обмежені обчислювальні ресурси;

– швидкість обробки. Оскільки JWT може бути підготовлений і перевірений дуже швидко, це забезпечує мінімальні затримки в комунікації між транспортним засобом і сервером. Це критично для систем моніторингу в реальному часі;

– гнучкість. JWT може легко інтегруватися в різні частини системи, включаючи мобільні додатки, веб-інтерфейси та сервери баз даних;

– сумісність з різноманітним апаратним забезпеченням. JWT може бути реалізований на різних платформах, включаючи вбудовані системи з обмеженими ресурсами, такими як Arduino.

Враховуючи всі ці фактори, JWT є оптимальним вибором для системи моніторингу громадським транспортом міста, надаючи ефективний, швидкий та гнучкий метод забезпечення безпеки передачі даних [15].

1.4 Постановка задачі

Громадський транспорт є невід'ємною частиною життя міста. Ефективність його роботи, а також зручність та доступність для громадян, зокрема для осіб з обмеженими можливостями, є критично важливими

аспектами сучасного життя в місті. В контексті цих викликів виникає потреба в розробці комплексної системи моніторингу громадського транспорту.

Метою даної роботи є розробка системи моніторингу громадського транспорту міста засобами мови програмування C# і бази даних MS SQL Server.

Основні завдання:

1. Розробити структуру бази даних:

– таблиця Водії (Driver). Зберігає інформацію про водіїв, включаючи ідентифікаційний номер, прізвище, ім'я, контактну інформацію та досвід роботи;

– таблиця Транспортні засоби (TransportVehicle). Містить дані про типи, моделі, реєстраційні номери та інші технічні характеристики транспортних засобів;

– таблиця Маршрути (Routes). Включає в себе інформацію про назву маршруту, пункти відправлення та прибуття, час та відстань;

– таблиця Історія руху (MovementHistory). Фіксує історію руху кожного транспортного засобу, включаючи час відправлення та прибуття;

– таблиця Ремонтні роботи (Maintenance). Зберігає інформацію про проведені технічні обслуговування та ремонти.

2. Реалізувати функціональність пошуку:

– пошук транспортних засобів. Функціонал, що дозволяє виконувати пошук транспортних засобів за різними параметрами, такими як тип, модель, рік виробництва тощо;

– пошук водіїв. Механізм для знаходження водіїв за іменем, прізвищем або ідентифікаційним номером.

3. Розробити звітність:

– розрахунок середньої швидкості для кожного транспортного засобу на основі даних моніторингу;

– визначення кількості транспортних засобів для кожного типу палива;

– виявлення маршрутів з максимальною та мінімальною відстанню;

– отримання списку маршрутів, які виконуються транспортними засобами на електроенергії;

– вибірка маршрутів, доступних для інвалідів.

4. Забезпечити інтеграцію з системами моніторингу та відстеження

– інтеграція з GPS-системами автотранспорту для отримання даних про місцезнаходження та швидкість транспортних засобів;

– співпраця з системами документування для зберігання інформації про події та дії користувачів.

5. Реалізувати інтерфейс для користувача:

– розробка графічного інтерфейсу, що забезпечує зручний доступ до всіх функцій системи, включаючи пошук, звітність та аналітику;

– впровадження системи аутентифікації та авторизації для різних ролей користувачів.

Об'єкти дослідження:

– Водії;

– Транспортні засоби;

– Маршрути;

– Транспортні зупинки;

– Ремонтні роботи;

– Система моніторингу та відстеження;

– Логи подій;

– Користувачі системи;

– Методика реалізації.

Для реалізації поставлених завдань буде використовуватися мова програмування C# для реалізації системи та MS SQL Server для забезпечення збереження даних. Система буде розроблятися з урахуванням принципів ООП, а також з використанням сучасних методологій розробки програмного забезпечення.

Розроблена система буде спроможна забезпечувати ефективний моніторинг громадського транспорту, включаючи зручні механізми пошуку, видачі звітів, та інтеграцію з системами моніторингу та відстеження. Це покращить якість обслуговування пасажирів та збільшить ефективність управління транспортною інфраструктурою міста.

Висновок до розділу

У рамках даного розділу було проведено загальне дослідження бізнес-процесів у сфері управління громадським транспортом. Визначено основні процеси діяльності, такі як планування маршрутів, управління транспортними засобами, забезпечення безпеки пасажирів та персоналу, а також моніторинг і контроль якості перевезень. Вивчено роль акторів та їхні функціональні обов'язки, а також структуру основних бізнес-процесів.

Огляд схожих систем, таких як GeoLocate Public Transport, TransitTrack та Moovit, дозволив визначити ключові аспекти, які слід враховувати при розробці. Специфічно, було проведено дослідження методів забезпечення безпеки в системах управління громадським транспортом. Після детального аналізу обрано метод JWT (JSON Web Tokens) для забезпечення безпеки передачі даних, враховуючи його ефективність, швидкість обробки, гнучкість та сумісність з різноманітним апаратним забезпеченням, включаючи вбудовані системи.

В результаті проведеного дослідження було сформульовано постановку задачі на розробку системи управління та моніторингу громадського транспорту міста. Сформовано системний підхід до забезпечення безпеки даних і вибрано оптимальні методи шифрування. Все це створює фундамент для подальшої розробки системи, яка матиме високу оперативність, надійність та забезпечить високий рівень безпеки та комфорту для пасажирів.

2 ВИБІР МЕТОДОЛОГІЇ ТА ТЕХНІЧНИХ РІШЕНЬ ДЛЯ ПРОГРАМНОГО КОМПЛЕКСУ МОНІТОРИНГУ

2.1 Оцінка та вибір технічних засобів

Важливим етапом розробки системи моніторингу громадського транспорту є оцінка та вибір технічних засобів, які будуть задіяні у цій системі.

2.1.1 Мова програмування

Від вибору мови програмування залежить не тільки швидкість розробки та впровадження системи, але і її надійність, масштабованість, а також загальні витрати на підтримку та розвиток.

При виборі мови програмування важливо враховувати ряд критеріїв. По-перше, мова повинна мати високий рівень підтримки бібліотек та фреймворків, які зможуть спростити розробку і скоротити терміни впровадження системи. По-друге, необхідно звернути увагу на спільноту розробників, оскільки її активність і розмір впливають на швидкість вирішення технічних проблем та пошуку оптимальних рішень. По-третє, мова програмування повинна бути придатною для реалізації вимог до високої доступності, швидкодії та масштабованості, які є критично важливими для систем моніторингу в реальному часі.

У зв'язку з вищезазначеними критеріями, мови програмування такі як Python, Java або C# можуть бути відмінним вибором. Python пропонує широкий набір бібліотек для обробки даних та веб-розробки, а також відзначається легкістю вивчення та швидкою розробкою. Java є вибором для розробки великих, масштабованих і надійних систем з високою пропускною здатністю. C# відзначається високою продуктивністю та інтеграцією з екосистемою продуктів Microsoft, яка може бути корисною при виборі технологічного стеку [16].

Python відзначається широкою підтримкою бібліотек та фреймворків, що значно спрощує розробку. Велика та активна спільнота розробників дозволяє швидко знаходити рішення для різноманітних технічних проблем.

Щодо швидкодії та масштабованості, Python може забезпечити достатній рівень завдяки асинхронному програмуванню та оптимізованим бібліотекам [17].

Java є ідеальним варіантом для розробки великих, масштабованих систем з високою пропускною здатністю. Ця мова має багатий набір бібліотек та фреймворків, а її спільнота розробників є однією з найбільших і найбільш досвідчених, що гарантує надійну підтримку на всіх етапах розробки [18].

C# є ще одним потужним інструментом для розробки, особливо якщо врахувати тісну інтеграцію з екосистемою продуктів Microsoft. Хоча спільнота розробників C# може бути меншою за розміром порівняно з Python або Java, її активність і фокус на якість забезпечують високий рівень підтримки. Сама мова відзначається високою продуктивністю та можливістю створення масштабованих та надійних систем [19].

У табл. 2.1 наведений порівняльний аналіз функціональних можливостей кожної мови програмування.

Таблиця 2.1 – Порівняльний аналіз мов програмування

Характеристика	Python	Java	C#
Компонентна архітектура	Відсутня	Відсутня	Присутня (WCF, WPF)
Інтеграція з Windows	Помірна	Помірна	Висока
Десктопні фреймворки	Tkinter, PyQt	Swing, JavaFX	WPF, WinForms
Підтримка машинного навчання	Scikit-learn, TensorFlow	Weka, DL4J	ML.NET, Accord.NET
Розширеність	Висока	Висока	Висока

Обґрунтування вибору C# для розробки системи моніторингу громадського транспорту базується на кількох ключових характеристиках цієї мови програмування. Зокрема, C# володіє потужними можливостями для компонентно-орієнтованої архітектури завдяки технологіям WCF та WPF. Це

забезпечує високий рівень гнучкості і дозволяє легко модифікувати систему. Що стосується інтеграції з Windows, C# має вбудовану підтримку та оптимізацію під цю платформу, що робить його особливо корисним для систем, розгорнутих в середовищі Microsoft. Ця мова також характеризується високим рівнем розширеності, що сприяє легкій інтеграції нових модулів та компонентів. На додачу, у контексті десктопних додатків, C# надає великий вибір інструментів для розробки за допомогою WPF та WinForms. За умови всіх цих характеристик, C# є оптимальним варіантом для розробки надійної, гнучкої та високопродуктивної системи моніторингу громадського транспорту.

2.1.2 Середовище розробки

Для розробки системи моніторингу громадського транспорту міста, що базується на мові програмування C#, можна розглянути такі середовища розробки, як Microsoft Visual Studio, Visual Studio Code та JetBrains Rider. Кожне з цих середовищ має свої унікальні переваги та можливості, що можуть бути корисними в залежності від конкретних потреб проекту.

Microsoft Visual Studio відзначається високим рівнем інтеграції з технологіями Microsoft, включаючи платформу .NET, що робить його ідеальним вибором для комплексних та масштабних проектів. Це середовище також надає широкий набір інструментів для відлагодження, тестування та профілювання коду, що є незамінним для забезпечення високої якості програмного забезпечення [20-21].

Visual Studio Code пропонує легковісний та швидкий інтерфейс з можливістю глибокої настройки через систему плагінів. Це робить його хорошим вибором для розробки прототипів або коли швидкість розробки є пріоритетним фактором.

JetBrains Rider, з своєї сторони, пропонує розширені можливості для рефакторингу коду та високу сумісність з іншими продуктами від JetBrains. Це може бути важливим, якщо в проекті планується використовувати інші

інструменти цього виробника, наприклад, для управління базами даних або для неперервної інтеграції [22].

В таблиці 2.2 наведений порівняльний аналіз інтегрованих середовищ розробки (IDE).

Таблиця 2.2 – Порівняльний аналіз середовищ розробки

Характеристика	Microsoft Visual Studio	Visual Studio Code	JetBrains Rider
Інтеграція з технологіями Microsoft	Висока	Середня	Низька
Набір інструментів для відлагодження	Повний	Обмежений	Обмежений
Підтримка автоматичного тестування	Повна	Часткова	Часткова
Профілювання коду	Вбудовані засоби	Через плагіни	Через плагіни
Інтеграція з системами контролю версій	Повна	Часткова	Часткова

З урахуванням наведених характеристик, Microsoft Visual Studio є найбільш оптимальним вибором для розробки системи моніторингу громадського транспорту міста на мові програмування C#. Це середовище розробки надає найбільший набір інструментів для ефективної роботи, включаючи вбудовані засоби для відлагодження, тестування, профілювання коду та інтеграції з системами контролю версій. Також варто відзначити високий рівень інтеграції з технологіями Microsoft, що є критично важливим для успіху проекту.

2.1.3 Система управління базами даних

Для розробки системи моніторингу громадського транспорту міста, що базується на мові програмування C# та платформі .NET, ключовим аспектом є

вибір системи управління базами даних (СУБД). Цей вибір впливає на низку критичних параметрів системи: продуктивність, масштабованість, надійність та безпеку зберігання даних. У цьому контексті рекомендується розглянути такі СУБД, як Microsoft SQL Server, PostgreSQL та MongoDB.

Microsoft SQL Server є високоякісною реляційною системою управління базами даних (СУБД), яка отримала широке визнання на ринку завдяки своїм передовим технологіям, надійності та сумісності з іншими продуктами від Microsoft. Спроектована з урахуванням вимог до великих об'ємів даних і високої доступності, ця СУБД інтегрується з інфраструктурою корпоративних систем на різних етапах — від розробки до впровадження та подальшої підтримки [23].

Однією з ключових характеристик SQL Server є його потужний оптимізатор запитів. Завдяки комплексним алгоритмам аналізу, система здатна ефективно працювати з великими наборами даних, мінімізуючи час відгуку на SQL-запити.

PostgreSQL відзначається високою продуктивністю та масштабованістю, а також підтримкою широкого спектру типів даних і гнучкість у створенні складних запитів. Ця СУБД є відкритим програмним забезпеченням, що забезпечує додаткову гнучкість при адаптації системи [24].

MongoDB є представником NoSQL баз даних і характеризується великою швидкістю запису та читання, а також гнучкістю структури даних. Вона може бути ефективною при зберіганні великих об'ємів неструктурованих або частково структурованих даних [25].

У табл. 2.3 наведено порівняльний аналіз можливостей розглянутих СУБД, який стане в пригоді при вирішенні проблеми зберігання даних.

Таблиця 2.3 – Порівняльний аналіз сховищ БД

Характеристика	Microsoft SQL Server	PostgreSQL	MongoDB

Інтеграція з технологіями Microsoft	Висока	Низька	Низька
Автоматизація резервного копіювання	Вбудовані засоби	Через плагіни	Через плагіни
Висока доступність	Так	Часткова	Часткова
Підтримка транзакцій	Повна	Повна	Обмежена
Безпека даних	Високий рівень	Середній рівень	Середній рівень

З урахуванням цих характеристик, Microsoft SQL Server є найбільш оптимальним вибором для системи моніторингу громадського транспорту міста. Висока інтеграція з технологіями Microsoft, вбудовані засоби для автоматизації резервного копіювання, висока доступність та підтримка транзакцій роблять цю СУБД особливо привабливою для проектів, заснованих на платформі .NET [26].

2.2 Аналіз вимог та моделювання прецедентів через Use-case діаграми

Для проведення детального опису функціональних характеристик системи моніторингу громадського транспорту міста та для точного визначення варіантів її використання з боку користувачів, надзвичайно релевантним є застосування діаграми прецедентів. Цей методологічний інструмент відіграє ключову роль у відображенні комплексних взаємовідносин між учасниками системи та можливими сценаріями її функціонування.

Прецедент у даному контексті представляє собою конкретизований варіант взаємодії користувача з системою з метою досягнення визначеного результату. Отже, кожний прецедент є елементарною функціональною одиницею системи та відображає специфічний аспект її використання.

Використання діаграми прецедентів в системі моніторингу громадського транспорту міста сприяє якісному опису типологічних моделей взаємодії між різними категоріями користувачів та системою. Даний інструмент також дозволяє деталізувати зовнішні критерії та вимоги до функціонування системи, що є критично важливим для її ефективної експлуатації та подальшого розвитку [27].

З урахуванням визначених цілей та завдань даної науково-технічної роботи, можна сформулювати комплекс функціональних вимог до планованого програмного застосунку. Деталізований перелік цих вимог презентовано у табл. 2.4 даного дослідження.

Таблиця 2.4 – Функціональні вимоги до додатку

Вимоги	Опис
REQ-1	Імплементація механізмів реєстрації та авторизації користувачів з різними ролями доступу до систем
REQ-2	Фіксація подій та дій користувачів у системі
REQ-3	Формування та редагування облікових записів користувачів, що містять необхідну інформацію для ідентифікації та контролю доступу
REQ-4	Додавання та модифікація записів в таблицях бази даних (водії, транспортні засоби, маршрути, ремонтні роботи тощо)
REQ-5	Реалізація функціоналу пошуку транспортних засобів та водіїв за різними атрибутами
REQ-6	Створення модулів звітності, що включають розрахунки середньої швидкості, типів палива, аналіз маршрутів та інші аналітичні параметри
REQ-7	Інтеграція системи з існуючими GPS-системами моніторингу для отримання даних про місцезнаходження та швидкість транспортних засобів

REQ-8	Інтеграція з системами документування та управління для зберігання архівних даних та дій користувачів
REQ-9	Розробка користувацького інтерфейсу, що максимально спрощує доступ до функціональних можливостей системи
REQ-10	Впровадження системи автентифікації та авторизації для користувачів з різними ролями і дозволами

В рамках даної системи, нефункціональні вимоги слугують критичним компонентом для аналізу якісних показників системи і стосуються таких атрибутів, як ефективність роботи, стабільність, доступність, захищеність, адаптивність, зручність користування та інші характеристики, не прямо пов'язані з функціональністю. Деталізована специфікація цих нефункціональних вимог приведена в таблиці 2.5 даного наукового дослідження. Цей ансамбль вимог сприяє глибокому осмисленню факторів, що повинні бути взяті до уваги при проектуванні та оцінюванні системи з погляду її якісних характеристик і відповідності очікуванням стейкхолдерів. Таким чином, нефункціональні вимоги відіграють ключову роль у забезпеченні високого рівня якості розробленої системи та її відповідності стандартам та потребам користувачів.

Таблиця 2.5 – Нефункціональні вимоги до додатку

Вимоги	Опис
REQ-11	Система повинна обробляти великі обсяги даних в реальному часі, забезпечуючи низькі затримки відгуку
REQ-12	Система повинна гарантувати стабільну роботу 24/7 та мати відмінні показники відновлюваності після збоїв
REQ-13	Система повинна бути доступною з різних типів пристроїв та мати гнучкі параметри доступності для різних груп користувачів

REQ-14	Забезпечення захисту даних користувачів та системи в цілому, включаючи шифрування, аутентифікацію та авторизацію
REQ-15	Система повинна бути гнучкою у плані розширення функціональності та підтримки великого числа користувачів
REQ-16	Користувацький інтерфейс має бути інтуїтивно зрозумілим та легким у використанні
REQ-17	Система повинна відповідати національним та міжнародним стандартам збереження та обробки даних
REQ-18	Система повинна мінімізувати споживання ресурсів, включаючи електроенергію та обчислювальну потужність

Основними учасниками, що взаємодіють з проектованою системою, є адміністратори та звичайні користувачі. Ці актори представляють собою критичні стейкхолдери, що безпосередньо взаємодіють з системою для здійснення специфічних задач. Кожна з цих груп має власний набір цілей, що відображає їхні потреби та аспекти взаємодії з системою. Деталізована характеристика акторів та їх цілей у взаємодії з системою представлена в таблиці 2.6 даного дослідження.

Таблиця 2.6 – Актори їхні цілі у системі

Актори	Цілі
Адміністратор	Основна ціль адміністратора полягає в ефективному адмініструванні облікових записів, з метою забезпечення їх стабільної функціональності та управлінської діяльності
Користувач	Завдання користувача полягає увзаємодії з системою через використання доступного функціоналу та сервісів
База даних	Основна функція бази даних — забезпечення надійної консервації та ефективного управління даними

Варіанти використання характеризуються як специфічні моделі взаємодії між суб'єктами взаємодії та системою, спрямовані на здійснення визначених завдань. В табл. 2.7 висвітлені ключові сценарії використання в рамках системи. Кожний такий сценарій представляє собою окремий приклад взаємодії, який актори можуть реалізувати з системою, виходячи із своїх конкретних потреб і завдань.

Таблиця 2.7 – Опис варіантів використання додатку

Варіант	Ім’я	Опис
UC1	Створення облікового запису	Дозволяє користувачу ініціювати процес реєстрації та формування нового облікового запису в системі
UC2	Ідентифікація користувача	Забезпечує користувачу можливість верифікації своєї ідентичності для отримання доступу до функціоналу системи
UC3	Перегляд списку користувачів	Дозволяє адміністратору системи ознайомитися з переліком зареєстрованих користувачів
UC4	Додавання користувача	Забезпечує адміністратору можливість інкорпорування нового користувача до системи
UC5	Модифікація даних користувача	Дозволяє адміністратору змінювати та оновлювати деталі облікового запису конкретного користувача
UC7	Перегляд списку водіїв	Забезпечує можливість ознайомлення з переліком водіїв, зареєстрованих в системі
UC8	Додавання водія	Дозволяє започаткувати процес реєстрації нового водія в системі
UC9	Оновлення даних водія	Дозволяє модифікувати існуючу інформацію про вибраного водія

UC10	Перегляд переліку транспортних засобів	Забезпечує доступ до каталогу транспортних засобів, що функціонують в системі
UC11	Додавання транспортного засобу	Дозволяє розпочати процедуру реєстрації нового транспортного засобу
UC12	Модифікація інформації транспортного засобу	Дозволяє змінити атрибути та характеристики вибраного транспортного засобу
UC13	Перегляд переліку зупинок	Забезпечує доступ до списку всіх транспортних зупинок, що входять до системи
UC14	Створення нової зупинки	Дозволяє ініціювати процес додавання нової транспортної зупинки
UC15	Оновлення даних зупинки	Дозволяє редагувати та оновлювати інформацію про конкретну зупинку
UC16	Презентація даних про ТО	Забезпечує доступ до деталей всіх проведених технічних обслуговувань
UC17	Додавання ТО	Дозволяє запускати процес занесення інформації про нове технічне обслуговування
UC18	Модифікація даних ТО	Забезпечує можливість зміни даних обраного технічного обслуговування
UC19	Контроль маршрутів	Дозволяє користувачу налаштовувати та ініціювати нові маршрути
UC20	Емуляція роботи системи	Забезпечує можливість користувачу симулювати діяльність системи в реальному часі

UC21	Пошук транспортних засобів	Забезпечує функціональність для знаходження конкретних транспортних засобів в базі даних
UC22	Пошук даних про водіїв	Дозволяє користувачу здійснювати фільтрацію і пошук інформації про водіїв
UC23	Відображення системних подій	Надає доступ до логу подій, що зареєстровані в системі
UC24	Генерація звітів	Дозволяє користувачу компілювати звіти на основі зібраних системних даних

За допомогою аналізу вхідних даних були створені діаграми варіантів використання, що ілюструють функціональні можливості системи для системного адміністратора (рис. 2.1) та звичайного користувача (рис. 2.2).

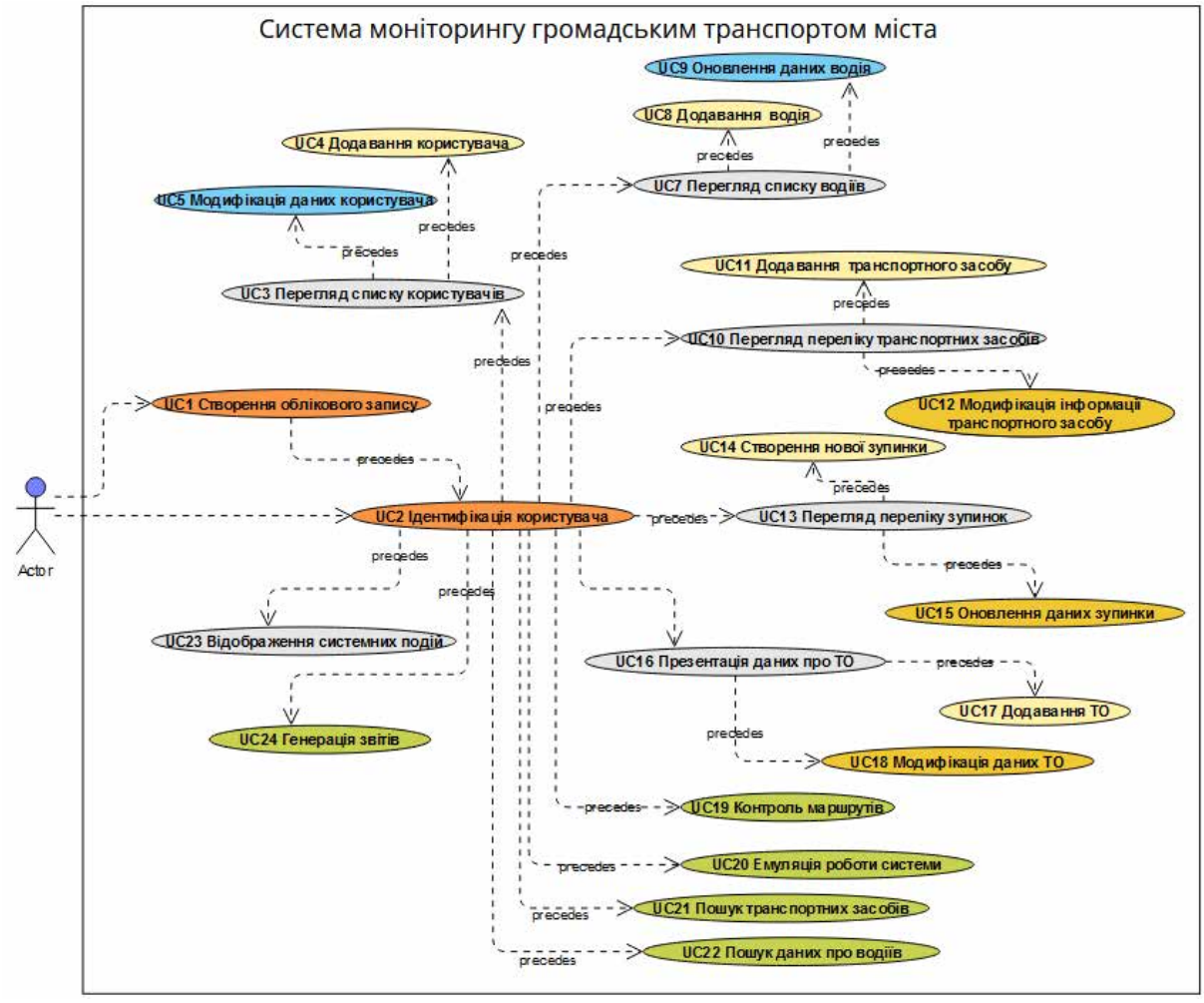

Рисунок 2.1 – Діаграма use-case для ролі «адміністратора»

Рисунок 2.2 – Діаграма use-case для ролі «користувач»

2.3 Проектування структури бази даних з використанням ERD та опис сутностей

В рамках даної досліджувальної роботи було сформовано Entity-Relationship Diagram (ERD), що спрямована на деталізацію сутностей та взаємозв'язків у системі моніторингу громадського транспорту міста. ERD включає в себе ряд ключових сутностей, які далі будуть розглянуті [28].

Для предметної області була створена ER-діаграма, яка дозволяє відобразити сутності та зв'язки між ними у цій системі. ER-діаграма включає наступні сутності та їх атрибути:

– сутність «Maintenance» призначена для зберігання інформації про технічне обслуговування транспортних засобів. Це критично важливий компонент для забезпечення надійності і безпеки експлуатації транспорту;

– сутність «Routes» репрезентує маршрути, по яких курсують транспортні засоби;

– сутність «TransportStop» слугує для зберігання інформації про транспортні зупинки;

– сутність «TransportVehicle» представляє транспортні засоби, що входять до складу громадського транспорту;

– сутність «Driver» слугує для інкапсуляції та зберігання всієї ключової інформації про водіїв громадського транспорту. Ця сутність необхідна для забезпечення оперативного доступу до даних про водіїв, їх особистої інформації, та робочого досвіду;

– сутність «MovementHistory» відповідає за зберігання інформації про історію переміщення транспортних засобів. Ця сутність є необхідною для аналізу ефективності маршрутів, контролю за дотриманням розкладу, а також для ретроспективного аналізу руху транспорту;

– сутність «MonitoringAndTracking» призначена для зберігання даних, отриманих в реальному часі від систем моніторингу і відстеження транспортних засобів;

– сутність «Logs» служить для зберігання інформації про всі події та дії, що відбулися в системі.

Сформовані сутності представляють ключові компоненти даних, які застосовуються в інженерії автоматизованої системи управління теплопостачанням на виробничій площадці. Ці сутності забезпечують ефективне зберігання та структурування релевантних даних для подальшої експлуатації та аналітики. ER-діаграма, розроблена для цієї ініціативи, забезпечує графічне представлення взаємозв'язків між відповідними таблицями та асоційованими з ними даними. Результати проектування цієї ER-діаграми представлено на рис. 3.3.

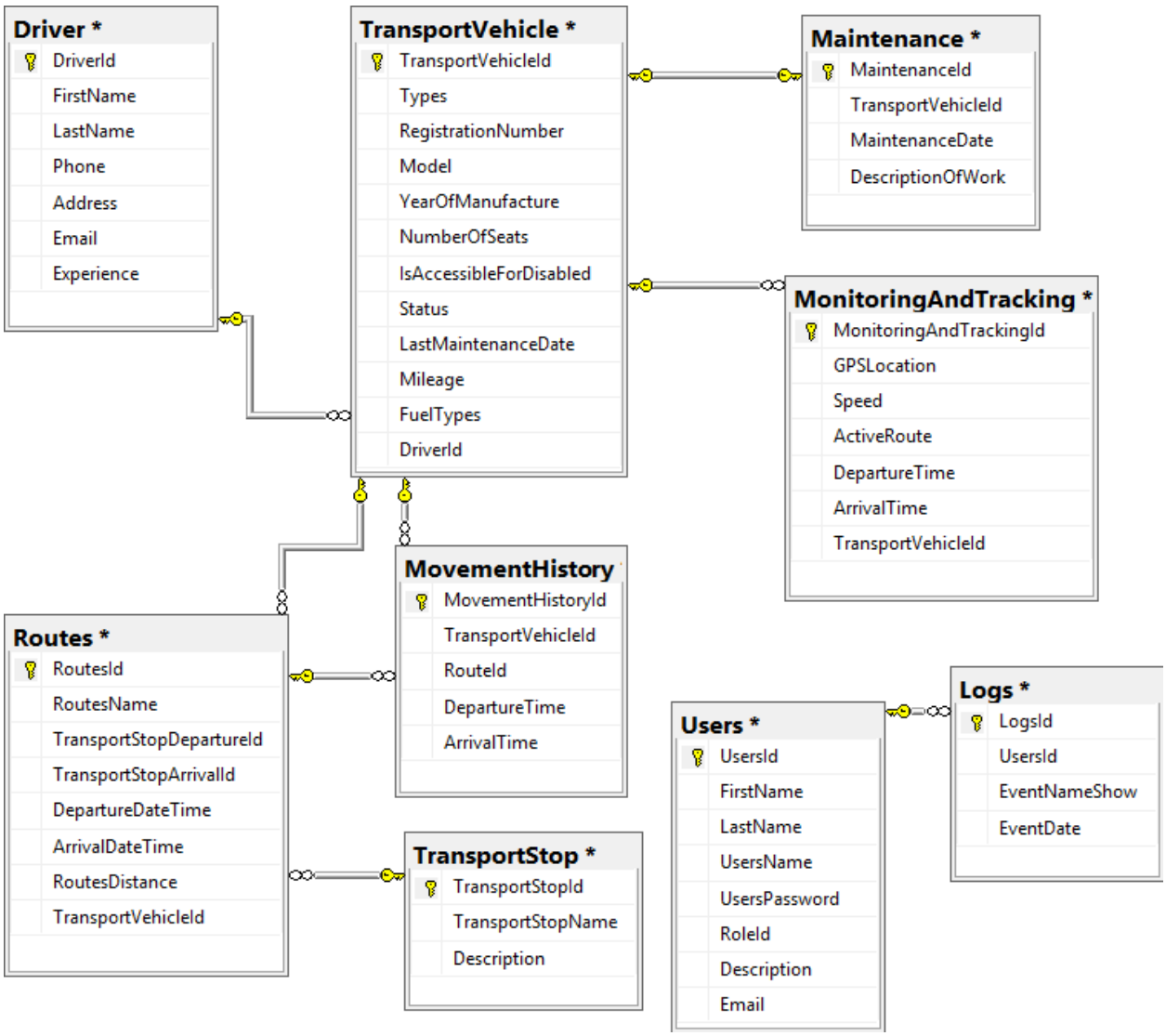

Рисунок 2.3 – ER-діаграма бази даних

2.4 Концепція архітектурних рішень

Архітектура даної системи моніторингу громадського транспорту міста базується на трьохрівневій структурі, яка охоплює такі компоненти:

– рівень користувацького інтерфейсу: Цей верхній рівень відповідає за безпосередню взаємодію між користувачем та системою. Для цієї системи, користувацький інтерфейс розроблено з використанням технологічного стеку, що включає середовище розробки Visual Studio 2022 та мову програмування C#. Цей рівень містить різноманітні користувацькі форми, на яких представлені необхідні дані та інтерактивні елементи для взаємодії з користувачем;

– рівень бізнес-логіки: Це центральний рівень, який зосереджує в собі логіку обробки даних та алгоритми функціонування системи. Він містить класи та методи, які відповідають за аналіз, обробку та маршрутизацію транспортних потоків, а також за взаємодію з базами даних;

– рівень управління даними: Цей нижній рівень займається зберіганням та маніпуляцією даними. Він включає в себе класи та функції, які забезпечують інтеграцію з базою даних, доступ до неї та оптимізацію запитів.

Така трьохрівнева архітектура дозволяє ефективно розділити функціональність системи на декомпозовані модулі, що полегшує її подальше тестування, модифікацію та масштабування.

2.4.1 Розробка рівня доступу до даних

У початковому етапі розробки системи було створено спеціалізований шар для управління доступом до даних, який складається з дев'яти класів. Ці класи мають ідентифікуючий суфікс «Provider» в назві, що є демонстративним відображенням їх функціонального призначення. Кожен з цих класів кореспондує з конкретною таблицею у реляційній базі даних, створюючи таким чином абстракцію над фізичним зберіганням даних.

Специфічність цих класів полягає в наданні розширеного набору методів, які дозволяють взаємодіяти з базою даних на різних рівнях абстракції. Це включає операції зчитування даних, додавання нових записів, модифікації існуючих та їх видалення. Завдяки такому підходу, розробники можуть взаємодіяти з базою даних у відокремленій манері, не зачіпаючи низькорівневу логіку доступу до даних.

Для візуального представлення архітектурної структури шару доступу до даних було створено діаграму класів, яка представлена на рисунку 2.4. Ця діаграма не лише відображає ієрархію та відношення між окремими класами, але і деталізує методи, які кожен клас надає для взаємодії з базою даних. Така діаграма дозволяє зрозуміти, як інкапсульовані в класах методи взаємодіють між собою, та які механізми вони використовують для доступу до бази даних.

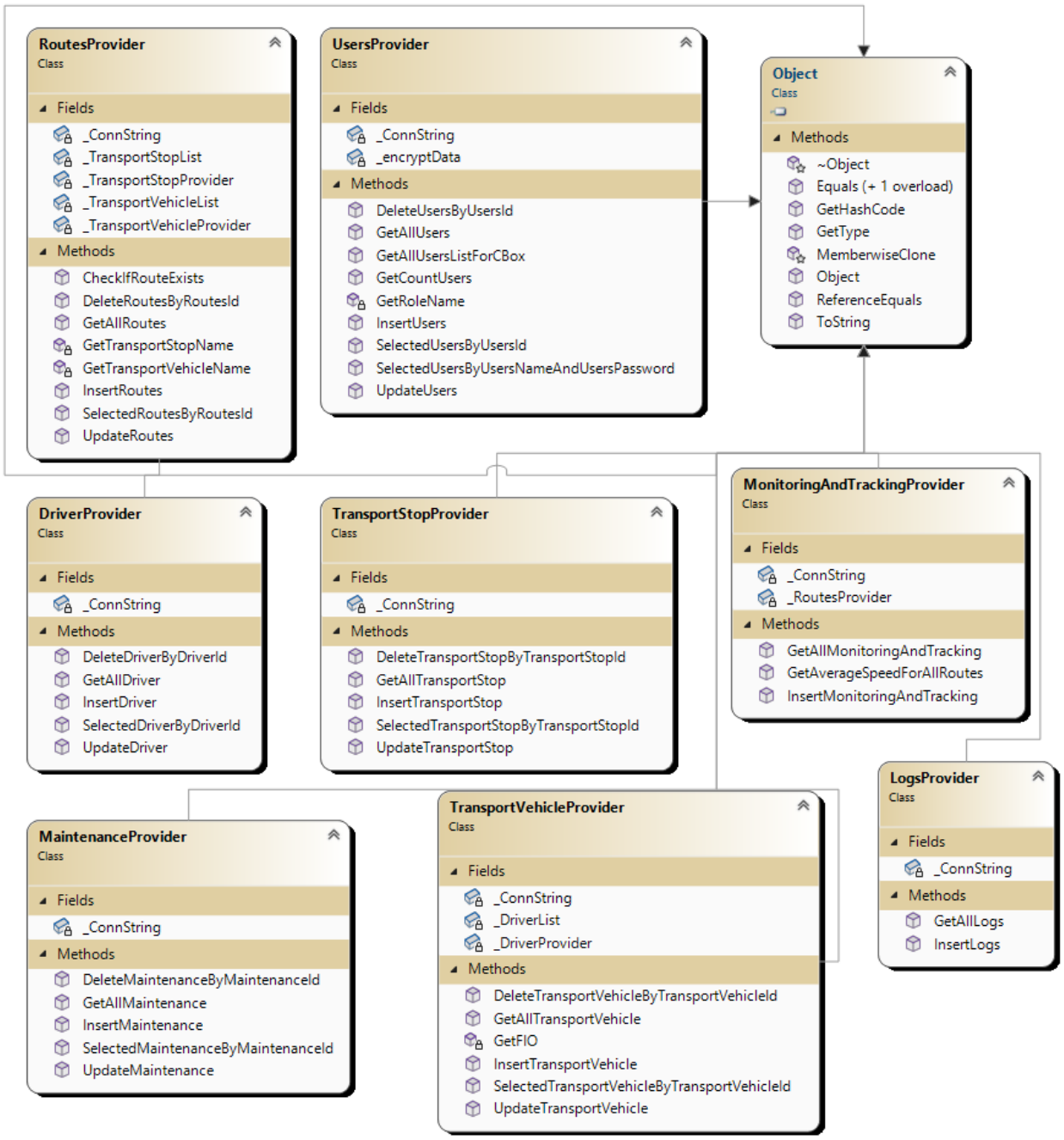

Рисунок 2.4 – Діаграма класів рівня даних

Діаграма класів, що зображена на рис. 2.4 складається із таких основних класів:

– клас DriverProvider служить для управління таблицею Driver в базі даних. Клас надає методи для обробки інформації про водіїв громадського транспорту, включаючи їх ім'я, прізвище, телефон, адресу електронної пошти та інші. Цей клас забезпечує можливість додавання, оновлення та видалення записів про водіїв в базі даних;

– клас LogsProvider відповідає за роботу з таблицею Logs в базі даних. Він має методи для збереження інформації про події, що відбулися в системі, такі як назва події, дата та ідентифікатор користувача, пов'язані з цією подією. Цей клас забезпечує взаємодію з базою даних для запису та отримання логів системи;

– клас MaintenanceProvider займається взаємодією з таблицею Maintenance в базі даних. Клас містить методи для управління інформацією про технічне обслуговування транспортних засобів, такою як дата обслуговування та опис виконаних робіт. Це дозволяє здійснювати запис, оновлення та видалення даних про технічне обслуговування;

– клас MonitoringAndTrackingProvider взаємодіє з таблицею MonitoringAndTracking в базі даних. Цей клас надає методи для обробки даних моніторингу та відстеження руху транспортних засобів. Він зберігає інформацію таку як GPS-локація, швидкість, активний маршрут, час відправлення та прибуття. Клас дозволяє додавати, оновлювати та видаляти записи моніторингу та відстеження;

– клас RoutesProvider фокусується на управлінні таблицею Routes в базі даних. Цей клас надає методи для роботи з маршрутами громадського транспорту, включаючи інформацію про назву маршруту, ідентифікатори зупинок відправлення та прибуття, час відправлення та прибуття, а також відстань маршруту. Клас дозволяє додавати, редагувати та видаляти записи про маршрути в базі даних;

– клас TransportStopProvider здійснює взаємодію з таблицею TransportStop в базі даних. Цей клас має методи для управління даними про транспортні зупинки, такими як назва зупинки та її опис. Це дозволяє здійснювати операції додавання, оновлення та видалення інформації про транспортні зупинки;

– клас UsersProvider служить для управління таблицею Users в базі даних. Цей клас містить методи, що дозволяють обробляти інформацію про користувачів системи. До атрибутів, які можна маніпулювати за допомогою

цього класу, належать ім'я, прізвище, логін, пароль, роль користувача та електронна адреса. Клас забезпечує можливість додавання нових користувачів, редагування існуючих профілів, а також видалення користувачів з системи.

Перелічені класи рівня даних відіграють ключову роль у забезпеченні функціональності системи моніторингу громадського транспорту, слугуючи мостом між базою даних та бізнес-логікою системи.

2.4.2 Розробка бізнес-логіки

Формування бізнес-логіки є критичним компонентом у процесі створення програмного рішення. Ця логіка відіграє ключову роль у виконанні бізнес-операцій та забезпеченні координованої роботи системи. Зазвичай, бізнес-логіка ізольована від взаємодії з базою даних та користувацькою інтерфейсною частиною, що гарантує її незалежність та універсальність.

Структурування бізнес-логіки на логічні модулі сприяє її масштабованості та модифікованості. Це надає можливість адаптації до змінних вимог замовника чи технологічного середовища без перебудови всієї системи.

Окрім того, ізоляція бізнес-логіки від користувацького інтерфейсу та бази даних сприяє тому, що систему можна легко адаптувати до використання різних баз даних або користувацьких інтерфейсів без необхідності змін у основній логіці додатку. Ця архітектурна рішення забезпечує системі високу адаптивність та гнучкість [29].

Візуалізація архітектури бізнес-логіки представлена на рис. 2.5.

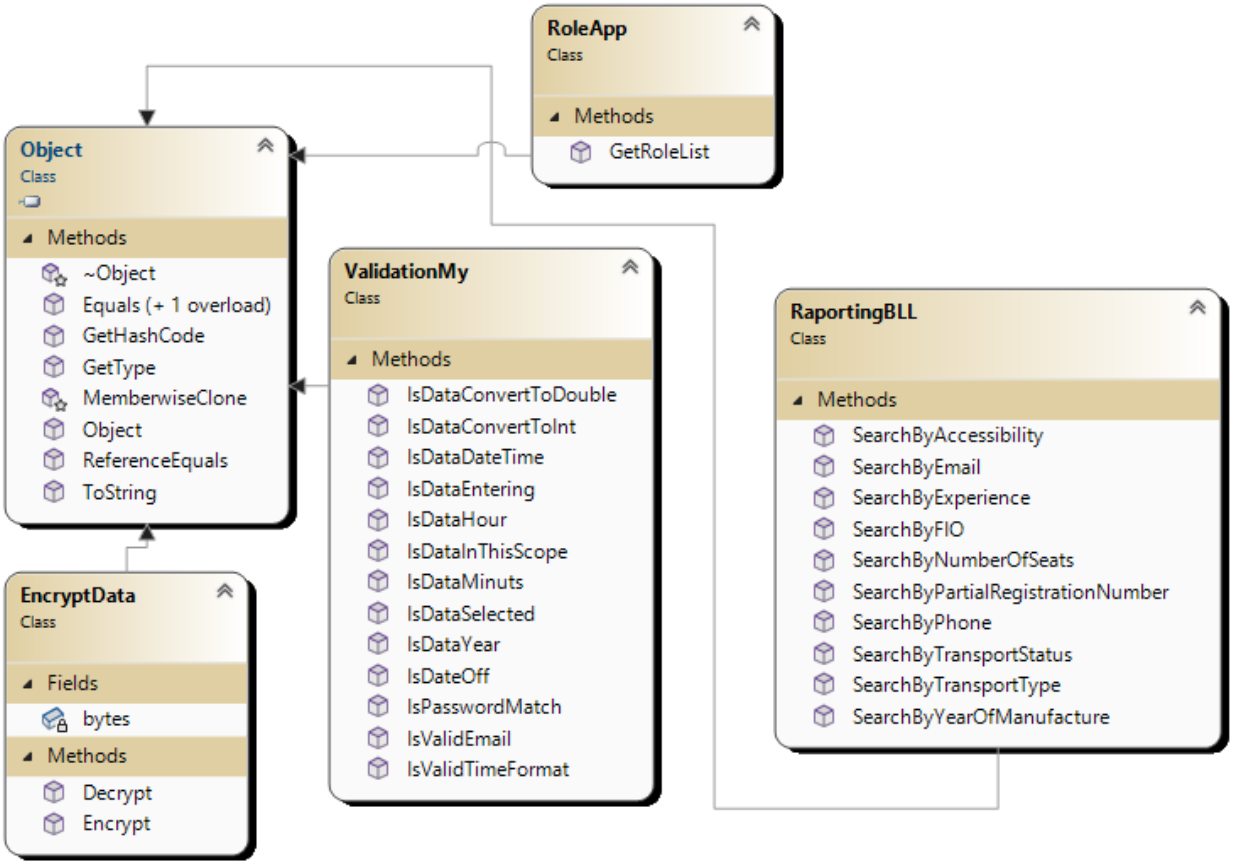

Рисунок 2.5 – Діаграма класів бізнес-логіки

Діаграма класів рівня бізнес-логіки складається з трьох основних класів:

– клас ReportingBLL служить для обробки та формування звітів у системі моніторингу громадського транспорту. Він включає методи для збору даних з різних частин системи, їх аналізу та конвертації в необхідний формат звіту. Цей клас є мостом між збереженими даними та кінцевим представленням інформації у формі звітів;

– клас RoleApp визначає ролі користувачів у системі. Це може бути адміністратор, диспетчер, водій та інші. Він має методи для установки, зміни та перевірки ролей користувачів, що дозволяє налаштувати доступ до різних частин системи відповідно до ролі користувача;

– клас ValidationMy відповідає за перевірку вхідних даних в системі. Він містить набір методів для валідації різних типів даних, таких як електронні адреси, номери телефонів, географічні координати тощо. Використання цього класу сприяє збільшенню надійності системи через відсіювання некоректних або шкідливих даних;

– клас EncryptData відповідає за шифрування чутливої інформації в системі. Він має методи для шифрування та розшифрування даних, таких як паролі, номери кредитних карток та інша конфіденційна інформація. Застосування цього класу забезпечує вищий рівень безпеки інформації.

2.4.3 Розробка користувацького інтерфейсу

Для рівня користувацького інтерфейсу було створено комплекс взаємопов'язаних форм, які є візуальним представленням інтерфейсу системи моніторингу громадського транспорту. Кожна з цих форм складається з багатого набору елементів керування, включаючи, але не обмежуючись, кнопки для виконання команд, текстові поля для введення або відображення даних, а також таблиці для структурованого представлення інформації.

Ці елементи керування не лише спрощує взаємодію між користувачем та системою, але й служать механізмом взаємодії з бізнес-логікою та шаром доступу до даних. Зокрема, за допомогою обробників подій, що ініціюються при взаємодії користувача з елементами керування, забезпечується передача даних та команд до відповідних рівнів системи.

Цей рівень інтерфейсу відіграє критичну роль у взаємодії користувача з системою: він дозволяє вводити необхідну інформацію для обробки, ініціювати різні види пошуку, переглядати актуальну інформацію про стан громадського транспорту та отримувати результати в адаптованому і зручному форматі.

Важливо відзначити, що дизайн та структура користувацького інтерфейсу були розроблені таким чином, щоб максимально спростити та оптимізувати процеси взаємодії користувача з системою. Це важливо для забезпечення високої швидкості реакції та ефективності роботи всієї системи [30].

Для глибшого розуміння структури та взаємозв'язків між елементами користувацького інтерфейсу було створено діаграму, яка наведена на рис. 2.6.

Ця діаграма служить візуальним доповненням, що дозволяє більш наочно представити архітектуру даного рівня системи.

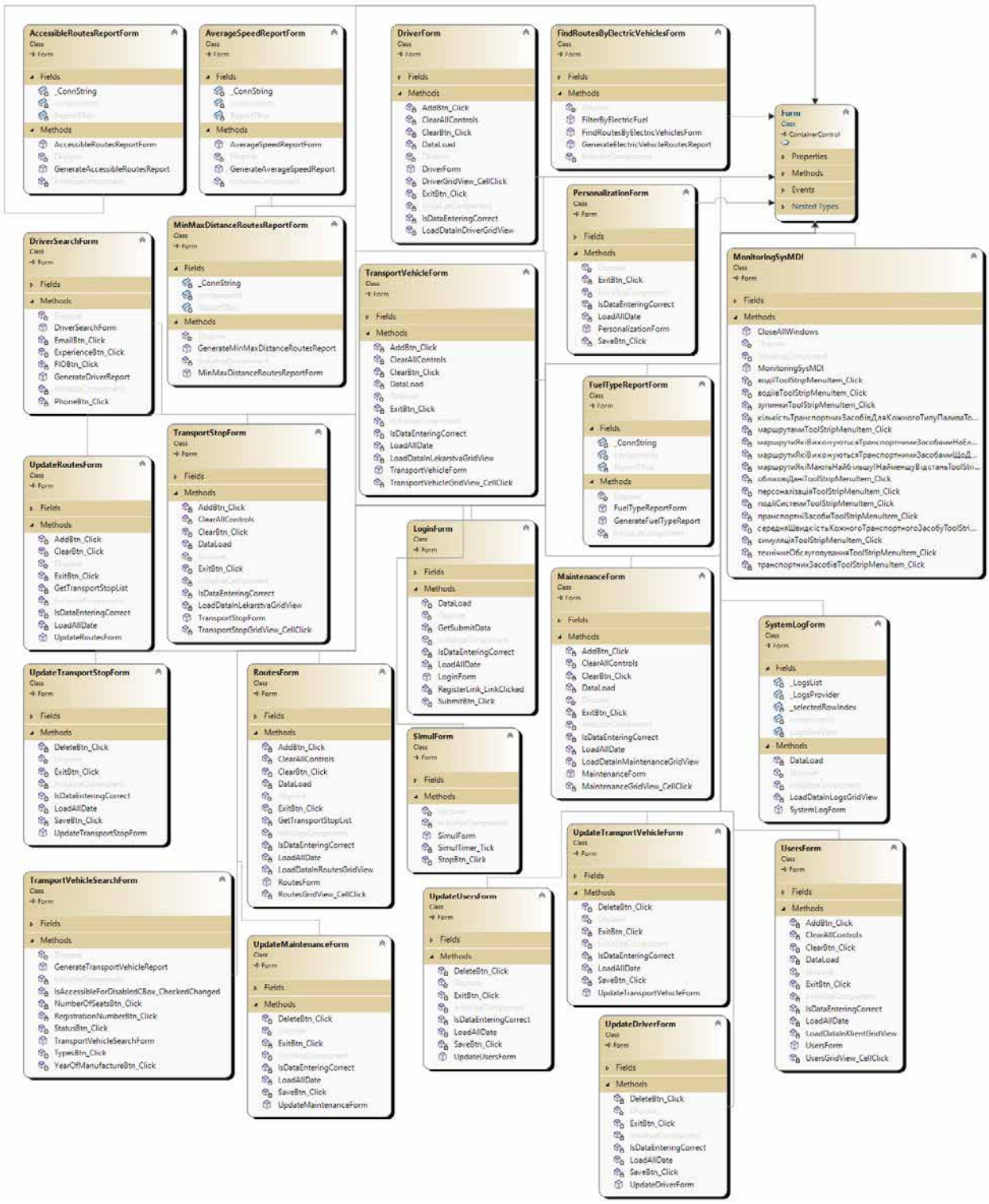

Рисунок 2.6 – Діаграма інтерфейсу системи

Діаграма даного рівня складається з двадцяти чотирьох класів рівня користувацького інтерфейсу і є похідними від класу Form:

– клас UpdateTransportStopForm призначений для редагування інформації про транспортні зупинки в системі. Форма цього класу містить ряд текстових полів, в яких можна змінювати назву зупинки, її опис та інші характеристики. Додатково, форма включає кнопки для збереження внесених змін або їх скасування. Це забезпечує гнучкість управління даними про транспортні зупинки;

– клас UpdateTransportVehicleForm відповідає за редагування даних про транспортні засоби. Форма надає користувачу можливість змінювати такі параметри, як тип транспорту, реєстраційний номер, модель, рік виробництва, кількість місць та інші. Форма також має кнопки для збереження змін або скасування операції;

– клас MonitoringSysMDI є головною формою багатовіконної (MDI) системи моніторингу громадського транспорту. Ця форма служить контейнером для інших форм системи та забезпечує основні функції управління ними. Вона включає меню для навігації, панелі інструментів для швидкого доступу до ключових функцій та зону відображення дочірніх форм;

– клас UpdateRoutesForm використовується для внесення змін до інформації про маршрути в системі. Ця форма містить текстові поля для редагування характеристик маршруту, таких як назва, пункти відправлення та прибуття, а також час відправлення та прибуття. Форма також надає кнопки для збереження змін або скасування операції;

– клас RoutesForm відповідає за відображення загальної інформації про доступні маршрути. Ця форма включає таблицю для відображення деталей кожного маршруту та кнопки для ініціювання дій, таких як редагування або видалення маршрутів;

– клас SimulForm дозволяє користувачу проводити симуляцію руху транспортних засобів по маршрутах. Форма містить елементи керування для вибору маршруту, транспортного засобу та інших параметрів симуляції. Після запуску симуляції, результати можуть бути відображені у вигляді графіків, таблиць або інших візуалізацій;

– клас UpdateDriverForm забезпечує можливість редагування даних про водіїв. Форма містить текстові поля для введення або зміни інформації про водія, такої як ім'я, прізвище, номер телефону та інші. Ця форма також має кнопки для подальшого збереження змін або їх скасування.

– клас UpdateMaintenanceForm служить для управління інформацією про технічне обслуговування транспортних засобів;

– клас DriverForm зосереджений на відображенні інформації про водіїв системи. Форма включає таблицю для відображення даних про всіх водіїв, а також кнопки для виклику форм редагування або видалення вибраного водія. Ця форма служить централізованим інтерфейсом для управління даними про водіїв;

– клас MaintenanceForm дозволяє користувачу переглядати та управляти інформацією про технічні обслуговування транспортних засобів. Ця форма містить таблицю, що відображає деталі кожного технічного обслуговування, включаючи дату, опис робіт та транспортний засіб, до якого вони відносяться. Додатково, форма надає кнопки для ініціювання дій редагування або видалення записів;

– Клас TransportStopForm покликаний забезпечити інтерфейс для управління та перегляду інформації про транспортні зупинки. Форма містить таблицю, що відображає ключові атрибути зупинок, такі як їхні назви та описи. Для функціональності управління, форма має кнопки для додавання нових зупинок, редагування існуючих та їх видалення;

– клас TransportVehicleForm створений для перегляду та управління інформацією про транспортні засоби. Форма включає в себе таблицю, яка відображає детальну інформацію про кожен транспортний засіб, а також набір кнопок для додавання, редагування та видалення записів;

– клас AccessibleRoutesReportForm призначений для генерації та відображення звітів про маршрути, доступні для осіб з обмеженими можливостями. Форма містить ряд фільтрів для вибору періоду часу, типу транспорту та інших параметрів, на основі яких буде сформований звіт;

– клас AverageSpeedReportForm служить для створення звітів про середню швидкість руху транспортних засобів по маршрутах. Форма надає можливість вибору конкретного періоду часу та типу транспорту, за якими буде проведений аналіз;

– клас FindRoutesByElectricVehiclesForm забезпечує користувача інтерфейсом для пошуку маршрутів, які обслуговуються електричними транспортними засобами. Форма містить ряд фільтрів для уточнення критеріїв пошуку, таких як період часу, тип транспорту або конкретна зупинка;

– Клас FuelTypeReportForm дійсно критичний для візуалізації звітів, що стосуються використання пального різними типами транспортних засобів. Цей клас містить компоненти для фільтрації даних за періодом часу, типом пального або типом транспортного засобу. За допомогою цієї форми, системні адміністратори та інші користувачі можуть отримати агреговані дані про використання пального, що є ключовим для стратегічного планування та оптимізації ресурсів.

– клас MinMaxDistanceRoutesReportForm розроблений для виведення звітів про маршрути з мінімальною та максимальною відстанню. Форма має набір фільтрів, що дозволяють користувачам вибирати періоди часу, тип транспорту та інші параметри для генерації звіту, що є інформативним для управлінських рішень;

– клас DriverSearchForm забезпечує інтерфейс для розширеного пошуку водіїв в системі. Ця форма містить ряд текстових полів, випадаючих списків та інших елементів керування, які дозволяють задати критерії пошуку, такі як ім'я, прізвище, стаж водія, та інше;

– клас TransportVehicleSearchForm є центральним місцем для виконання пошукових запитів до бази даних транспортних засобів. Користувач може вводити різні параметри для пошуку, включаючи тип транспортного засобу, реєстраційний номер, рік виробництва та інші атрибути.

Кожен з цих класів створений з метою оптимізації взаємодії користувача з системою, що спрямована на моніторинг громадського транспорту. Вони відіграють важливу роль в архітектурі програмного забезпечення, спрощуючи задачі управління та аналізу даних.

2.5 Обґрунтування вибору компонентів системи

Для реалізації системи моніторингу громадського транспорту міста було вибрано контролер Arduino Uno (рис. 2.7). Цей контролер базується на мікроконтролері ATmega328P та працює на тактовій частоті 16 MHz. Основні характеристики включають 14 цифрових I/O портів, 6 аналогових входів, а також модулі UART, SPI та I2C для комунікації.

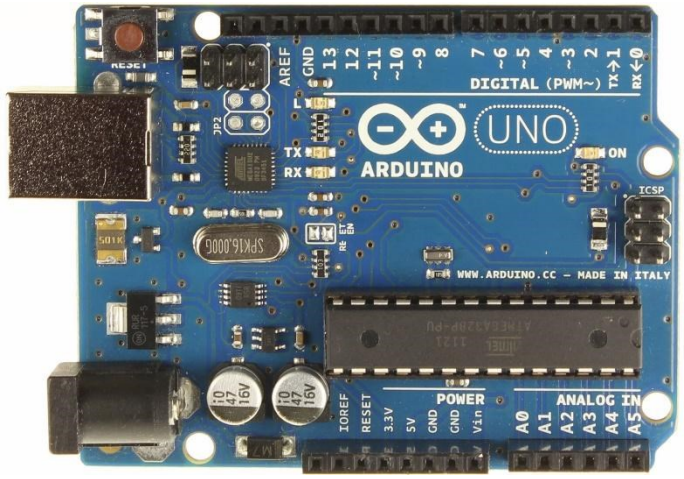

Рисунок 2.7 – Зовнішній вигляд мікроконтролера Arduino Uno

Специфікації контролера:

– мікропроцесор: ATmega328P;
– частота: 16 MHz;
– цифрові I/O порти: 14;
– аналогові входи: 6;
– живлення: 7-12V.

Вибір Arduino Uno можна аргументувати з кількох причин:

– відкритий код. Arduino Uno має відкритий код, що дозволяє інженерам та розробникам легко модифікувати та адаптувати систему до конкретних потреб та вимог;

– гнучкість та масштабування. Завдяки відкритому коду, існує можливість для розробки специфічних модулів та плагінів, які можуть враховувати локальні особливості та потреби;

– сумісність з різноманітними датчиками. Arduino Uno сумісний з широким спектром датчиків, що вимагається для комплексного моніторингу параметрів громадського транспорту;

– доступність та вартість. Це економічно ефективний вибір, що не впливає на функціональність та надійність системи.

Отже, Arduino Uno є оптимальним вибором для даної системи, забезпечуючи необхідний баланс між функціональністю, гнучкістю та вартістю.

Для повноцінної системи необхідним є підбір датчиків, які можуть ефективно відслідковувати різні параметри руху та стану транспортних засобів. Це у свою чергу включає датчики швидкості, температури двигуна, рівня палива, а також сенсори, які можуть визначати геолокаційні дані, швидкість руху транспортного засобу та інші важливі фактори.

Для визначення геолокаційних даних обрано GPS-датчик NEO-6M, який характеризується високою чутливістю та низьким рівнем споживаної потужності, що робить його ефективним для використання в реальних умовах, де економія енергії є критичною. Живлення модуля варіюється від 3 до 5 вольт, що робить його сумісним з більшістю мікроконтролерів, зокрема Arduino Uno.

NEO-6M може працювати з частотою оновлення від 1 до 5 герц, що дозволяє забезпечити досить точну і актуальну інформацію про місцезнаходження транспортного засобу. Інтерфейс модуля базується на UART, який є широко розповсюдженим та добре підтримуваним, тому інтеграція в систему не викличе труднощів.

Однією з ключових особливостей є його здатність забезпечувати геолокаційні дані з точністю до 2,5 метра. Ця характеристика є вкрай важливою для системи моніторингу транспорту, де необхідна висока точність

визначення місцезнаходження для забезпечення ефективної навігації та управління трафіком.

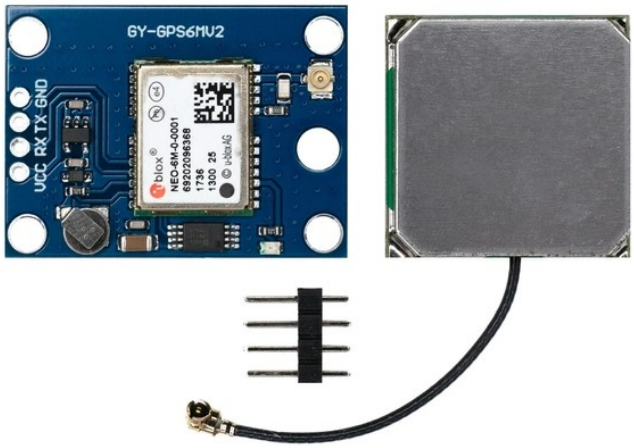

Рисунок 2.8 – Зовнішній вигляд датчика NEO-6M

Отже, GPS-модуль NEO-6M не лише відповідає всім технічним вимогам для системи моніторингу громадського транспорту, але й пропонує ряд переваг, таких як висока точність, низьке споживання енергії та гнучкість в налаштуванні та використанні.

Датчик швидкості на основі ефекту Холла A3144 (Hall Effect Sensor) є важливим компонентом в системі моніторингу громадського транспорту. Цей датчик відзначається високою чутливістю і може визначати швидкість обертання колеса з точністю до 1 мм. Завдяки широкому діапазону живлення, від 4,5 до 24 вольт, датчик легко інтегрується в різні електричні системи транспортних засобів. Вихідний сигнал датчика є цифровим, що спрощує його обробку та інтеграцію з іншими системами.

Рисунок 2.9 – Зовнішній вигляд датчика A3144

Однією з ключових переваг A3144 є його надійність та довговічність, що робить його ідеальним вибором для використання в умовах інтенсивної експлуатації, які є характерними для громадського транспорту. Цей датчик дозволяє точно моніторити швидкісний режим транспортного засобу, що є критично важливим для безпеки пасажирів та ефективності руху транспорту в місті.

Враховуючи всі ці фактори, можна стверджувати, що датчик швидкості A3144 є оптимальним вибором для системи моніторингу громадського транспорту, яка має на меті не лише відстеження місцезнаходження транспортних засобів, але і контроль їхнього дотримання встановлених швидкісних режимів.

Враховуючи обраний для цієї системи контролер на базі Arduino Uno, ключовим аспектом є сумісність між 4G/5G модемом та самим контролером. Arduino Uno підтримує комунікацію через UART, SPI та I2C інтерфейси, тому при виборі модему було звернено увагу на наявність одного з цих інтерфейсів для забезпечення простої і надійної інтеграції.

Один з популярних моделей 4G модемів, який можна використовувати в таких системах, — це SIM7600E-H 4G HAT. Цей модем сумісний з Arduino Uno та має широкий діапазон підтримуваних частот. Він підтримує UART інтерфейс, що дозволяє легко підключити його до Arduino Uno.

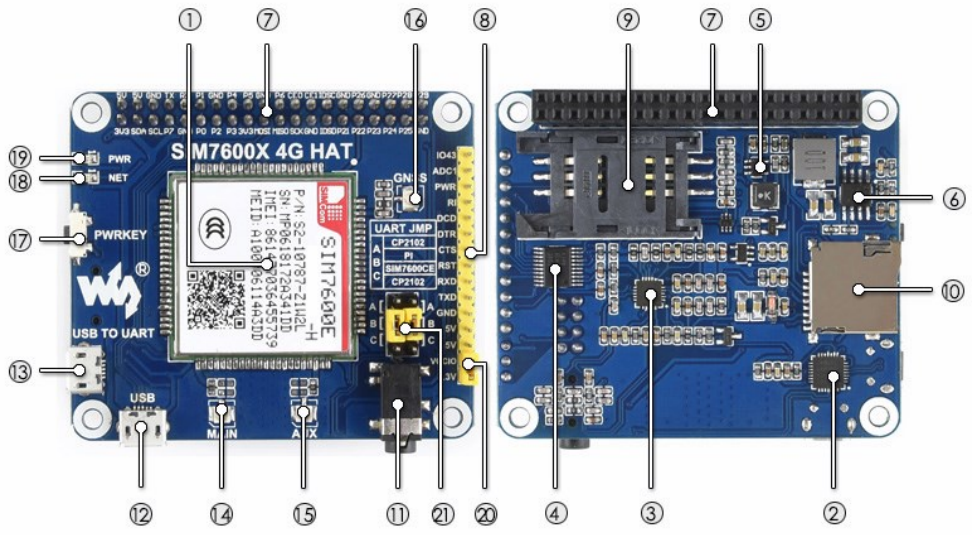

Рисунок 2.10 – Зовнішній вигляд датчика SIM7600E-H 4G HAT

Така конфігурація не тільки забезпечує надійну та швидку передачу даних, але і дозволяє використовувати розширений набір функцій, таких як SSL-шифрування для забезпечення безпеки даних. Крім того, використання такого модему забезпечує можливість майбутнього оновлення та масштабування системи без необхідності заміни основного контролера.

Для забезпечення стабільної роботи всієї системи моніторингу необхідно врахувати питання ефективного та надійного блоку живлення. Вибір здійснюється з огляду на такі параметри як вхідна та вихідна напруга, стабільність живлення, ефективність перетворення енергії, а також сумісність із використовуваним контролером Arduino Uno та підключеними датчиками.

Один із варіантів — це DC-DC конвертер напруги, який може бути підключений безпосередньо до акумулятора транспортного засобу. Такий конвертер може перетворювати вхідну напругу з діапазону, наприклад, 12-24V до необхідних 5V для живлення Arduino Uno і підключених датчиків. Важливо, щоб конвертер мав захист від короткого замикання, перевантаження та перенапруги, щоб забезпечити безперебійну роботу системи.

Рисунок 2.11 – Понижуючий перетворювач DC-DC

Кожен з цих датчиків був вибраний з урахуванням специфіки вимог до системи моніторингу громадського транспорту. Вони забезпечують високу точність вимірювань, надійність в роботі та сумісність з контролерами на базі Arduino.

Висновок до розділу

В рамках даного розділу було здійснено комплексний підбір та обґрунтування ключових технічних рішень для програмного комплексу моніторингу громадського транспорту. По-перше, після аналізу кількох потенційних мов програмування — Python, Java, C# — зроблено вибір на користь C#, що забезпечує високу продуктивність та інтеграцію з платформою Microsoft. Середовище розробки, обране для реалізації проекту, стало Microsoft Visual Studio, яке надає широкий спектр інструментів для ефективної розробки. Щодо систем управління базами даних, було розглянуто декілька варіантів, зокрема Microsoft SQL Server, PostgreSQL, MongoDB, та вибрано Microsoft SQL Server, який найкраще відповідає потребам проекту.

Важливим етапом стало моделювання бізнес-процесів та взаємодій між компонентами системи через Use-case діаграми. Це дало змогу чітко визначити границі системи та її основні функціональні можливості. Далі, було спроектовано структуру бази даних, зокрема побудовано ER-діаграму, яка відображає ключові сутності та зв'язки між ними.

Архітектурно, система базується на трьохрівневій моделі, що включає рівні доступу до даних, бізнес-логіки та користувацького інтерфейсу. Такий підхід забезпечує гнучкість, модульність та можливість легкої адаптації до змін.

Окрему увагу приділено підбору апаратних компонентів. Обрано контролер Arduino Uno, до якого підключено декілька датчиків: GPS-модуль NEO-6M для визначення геолокації, датчик швидкості на основі ефекту Холла A3144 та модуль SIM7600E-H 4G HAT для передачі даних до бази.

Отже, розроблена система є гнучкою, масштабованою та відкритою до подальших модифікацій, що робить її актуальною та перспективною для вирішення задач моніторингу громадського транспорту.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ГРОМАДСЬКИМ ТРАНСПОРТОМ МІСТА

3.1 Створення структури бази даних

Специфіка цього підрозділу полягає в детальному розгляді концептуальної та фізичної моделі бази, включаючи вибір типів даних, створення таблиць, встановлення зв'язків між ними, а також розробка засобів для забезпечення цілісності та безпеки даних.

Концептуальна модель бази даних є фундаментальним етапом в розробці автоматизованої інформаційної системи для моніторингу громадського транспорту. Ця модель слугує як концептуальний каркас, на якому будуть базуватися всі наступні етапи проектування та реалізації системи. Значущість цієї моделі полягає в можливості цілісного та структурованого представлення даних, які будуть зберігатися, оброблятися та аналізуватися в програмному рішенні.

Предметна область включає такі основні сутності:

– водій (Driver). Сутність «Водій» відображає інформацію про окремих водіїв громадського транспорту міста. Вона містить атрибути, такі як ідентифікатор водія, повне ім'я, вік, телефон, адреса, та досвід. Ця сутність ключова для ведення даних про персонал, що обслуговує громадський транспорт;

– технічне обслуговування (Maintenance). Ця сутність представляє записи про технічне обслуговування кожного транспортного засобу. Основні атрибути включають: ідентифікатор обслуговування, ідентифікатор транспортного засобу, дату проведення та опис виконаних робіт. Сутність дозволяє слідкувати за технічним станом транспортних засобів та планувати майбутні технічні втручання;

– історія руху (MovementHistory). Сутність «Історія руху» містить інформацію про рух окремих транспортних засобів по маршрутам. Її атрибути включають: ідентифікатор запису, ідентифікатор транспортного засобу, ідентифікатор маршруту, дату і час відправлення, та дату і час прибуття. Ця

сутність необхідна для моніторингу та аналізу ефективності роботи транспортної системи;

– маршрут (Routes). Сутність містить інформацію про маршрути громадського транспорту. Основні атрибути: ідентифікатор маршруту, назва, початкова зупинка, кінцева зупинка, дата/час прибуття, та відстань, ідентифікатор транспортного засобу. Ця сутність дозволяє систематизувати дані про маршрути та забезпечує взаємозв'язок з іншими сутностями бази даних;

– транспортний засіб (TransportVehicle). Ця сутність зберігає інформацію про транспортні засоби, які входять до складу громадського транспорту міста. Основні атрибути включають ідентифікатор транспортного засобу, тип, реєстраційний номер, модель, рік виробництва, кількість місць, доступність для осіб з інвалідністю, тип палива, кілометраж, статус транспортного засобу та ідентифікатор водія;

– зупинка (TransportStop). Сутність «Зупинка» відображає інформацію про окремі зупинки на маршрутах громадського транспорту. Атрибути сутності: ідентифікатор зупинки, назва, та опис місцезнаходження. Сутність забезпечує важливу інформацію для планування маршрутів та служить важливим зв'язком з іншими сутностями в базі даних;

– користувач (User). Сутність представляє інформацію про користувачів системи моніторингу. Основні атрибути включають: ідентифікатор користувача, ім’я, прізвище, логін, пароль, ідентифікатор ролі, електронна адреса та додатковий опис. Сутність дозволяє забезпечити контроль доступу до системи та її ресурсів;

– моніторинг та відстежування (MonitoringAndTracking). Сутність представляє інформацію про моніторинг та відстеження транспортних засобів в системі. Основні атрибути сутності: ідентифікатор моніторинг та відстежування, дані GPS, швидкість, активний маршрут, дата/час відправлення, дата/час прибуття, ідентифікатор транспортного засобу.

З урахуванням сутностей та їхніх атрибутів, можемо визначити наступні ключові зв'язки у таблицях бази даних:

– зв'язок «один до багатьох» між Технічним обслуговуванням і Транспортним засобом: Один транспортний засіб може проходити через кілька етапів технічного обслуговування, які фіксуються у базі даних;

– зв'язок «один до багатьох» між Транспортним засобом і Історією руху: Один транспортний засіб може бути задіяний у кількох маршрутах, що фіксуються як окремі записи в історії руху;

– зв'язок «багато до багатьох» між Маршрутом і Зупинкою: Одна зупинка може бути частиною кількох маршрутів, і, навпаки, один маршрут може включати кілька зупинок;

– зв'язок «один до багатьох» між Маршрутом і Розкладом: Один маршрут може мати кілька записів в розкладі, що відображають різні дні тижня чи часи відправлення;

– зв'язок «один до одного» між Користувачем і Подіями: Кожен користувач може здійснювати багато дій у системі;

– зв'язок «один до багатьох» між Транспортним засобом і Розкладом: Один транспортний засіб може мати кілька записів в розкладі, що відображають його робочий графік в різні дні;

– зв'язок «один до багатьох» між Транспортним засобом і Технічним обслуговуванням: Один транспортний засіб може ініціювати кілька процесів технічного обслуговування;

– зв'язок «один до багатьох» між Маршрутом і Транспортним засобом: Один маршрут може бути обслуговий кількома транспортними засобами, які змінюються згідно розкладу або на технічних причинах.

На рис. 3.1 зображено концептуальну модель бази даних у нотації Чена.

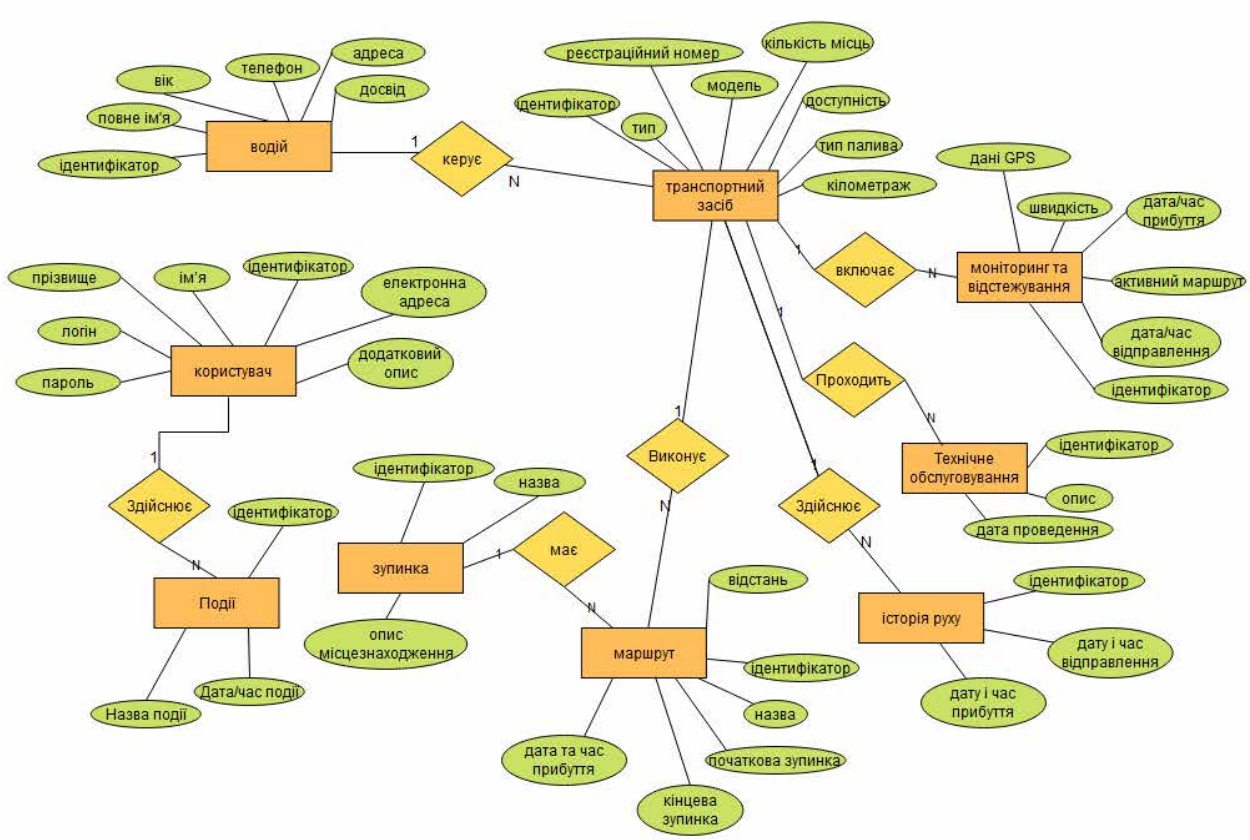

Рисунок 3.1 – База даних в нотації Чена

Наступним кроком є створення реляційної моделі бази даних є ключовим етапом у створенні структури для зберігання даних, яка відповідає вимогам і функціональності системи моніторингу громадського транспорту.

Нижче приведено основні таблиці та атрибути, що включаються до реляційної моделі бази даних:

Таблиця 3.1 – Атрибути сутності «Driver»

№	Найменування	Тип даних	Призначення
1	DriverId	INT	Ідентифікатор водія
2	FirstName	NVARCHAR(50)	Ім'я водія
3	LastName	NVARCHAR(50)	Прізвище водія
4	Phone	NVARCHAR(20)	Номер телефону водія
5	Address	NVARCHAR(MAX)	Адреса водія
6	Email	NVARCHAR(MAX)	Адреса електронної пошти водія
7	Experience	INT	Стаж водія

Таблиця 3.2 – Атрибути сутності «Maintenance»

№	Найменування	Тип даних	Призначення
1	MaintenanceId	INT	Ідентифікатор обслуговування
2	TransportVehicleId	INT	Ідентифікатор транспортного засобу
3	MaintenanceDate	DATETIME	Дата обслуговування
4	DescriptionOfWork	NVARCHAR(MAX)	Опис виконаних робіт

Таблиця 3.3 – Атрибути сутності «Routes»

№	Найменування	Тип даних	Призначення
1	RoutesId	INT	Ідентифікатор маршруту
2	RoutesName	NVARCHAR(150)	Назва маршруту
3	TransportStopDepartureId	INT	Ідентифікатор зупинки відправлення
4	TransportStopArrivalId	INT	Ідентифікатор зупинки прибуття
5	DepartureDateTime	DATETIME	Дата та час відправлення на маршрут
6	ArrivalDateTime	DATETIME	Дата та час прибуття на маршрут
7	RoutesDistance	FLOAT	Відстань маршруту в кілометрах
8	TransportVehicleId	INT	Ідентифікатор транспортного засобу, що обслуговує маршрут

Таблиця 3.4 – Атрибути сутності «MovementHistory»

№	Найменування	Тип даних	Призначення
1	MovementHistoryId	INT	Ідентифікатор історії руху
2	TransportVehicleId	INT	Ідентифікатор транспортного засобу
3	RouteId	INT	Ідентифікатор маршруту
4	DepartureTime	DATETIME	Час відправлення
5	ArrivalTime	DATETIME	Час прибуття

Таблиця 3.5 – Атрибути сутності «TransportVehicle»

№	Найменування	Тип даних	Призначення
1	TransportVehicleId	INT	Ідентифікатор транс. засобу
2	Types	NVARCHAR(50)	Тип транспортного засобу
3	RegistrationNumber	NVARCHAR(50)	Реєстраційний номер
4	Model	NVARCHAR(50)	Модель транспортного засобу
5	YearOfManufacture	INT	Рік виробництва транспортного засобу
6	NumberOfSeats	INT	Кількість сидячих місць у транспортному засобі
7	Status	NVARCHAR(50)	Статус транспортного засобу
8	LastMaintenanceDate	DATETIME	Дата останнього технічного обслуговування
9	Mileage	FLOAT	Пробіг транспортного засобу в кілометрах
10	FuelTypes	NVARCHAR(50)	Типи пального пального
11	DriverId	INT	Ідентифікатор водія, який обслуговує транспортний засіб

Таблиця 3.6 – Атрибути сутності «TransportStop»

№	Найменування	Тип даних	Призначення
1	TransportStopId	INT	Ідентифікатор зупинки
2	TransportStopName	NVARCHAR(150)	Назва зупинки
3	Description	NVARCHAR(MAX)	Опис зупинки

Таблиця 3.7 – Атрибути сутності «Client»

№	Найменування	Тип даних	Призначення
1	LogsId	INT	Ідентифікатор запису журналу подій
2	UsersId	INT	Ідентифікатор користувача, який виконав подію
3	EventNameShow	NVARCHAR(MAX)	Назва та опис події
4	EventDate	DATETIME	Дата та час події

Таблиця 3.8 – Атрибути сутності «Users»

№	Найменування	Тип даних	Призначення
1	UsersId	INT	Ідентифікатор користувача
2	FirstName	NVARCHAR(50)	Ім'я користувача
3	LastName	NVARCHAR(50)	Прізвище користувача
4	UsersName	NVARCHAR(50)	Ім'я користувача для входу
5	UsersPassword	NVARCHAR(50)	Пароль користувача
6	RoleId	INT	Ідентифікатор ролі користувача
7	Description	NVARCHAR(1000)	Опис користувача (максимальна довжина)
8	Email	NVARCHAR(150)	Адреса електронної пошти користувача

Таблиця 3.9 – Атрибути сутності «Client»

№	Найменування	Тип даних	Призначення
1	Monitoring AndTrackingId	INT	Ідентифікатор моніторингу та відстеження
2	GPSLocation	NVARCHAR(100)	Географічне розташування (координати GPS)
3	Speed	FLOAT	Швидкість руху
4	ActiveRoute	NVARCHAR(150)	Активний маршрут
5	DepartureTime	DATETIME	Час відправлення
6	ArrivalTime	DATETIME	Час прибуття
7	TransportVehicleId	INT	Ідентифікатор транспортного засобу

Ці таблиці забезпечують повноту та цілісність інформаційної моделі, дозволяючи системі ефективно управляти всіма аспектами оренди торгових площ: від укладання договорів до контролю платежів, штрафів та ремонтних робіт. Кожна таблиця містить ключові атрибути, необхідні для збереження та обробки відомостей в рамках заданих бізнес-процесів.

Після завершення процесу проектування концептуальної моделі бази даних, наступним етапом є її фізична реалізація.

Для фізичної реалізації бази даних використовується система управління базами даних (СУБД) Microsoft SQL Server.

Для оптимізації запитів до бази даних, було створено первинні та унікальні ключі. Додатково, було використано індекси для полів, які часто використовуються в запитах, зокрема для полів, що містять дати.

Скрипти створення таблиць бази даних приведено у додатку А, а на рис. 3.2 зображено фізичну модель бази даних після відпрацювання цих скриптів.

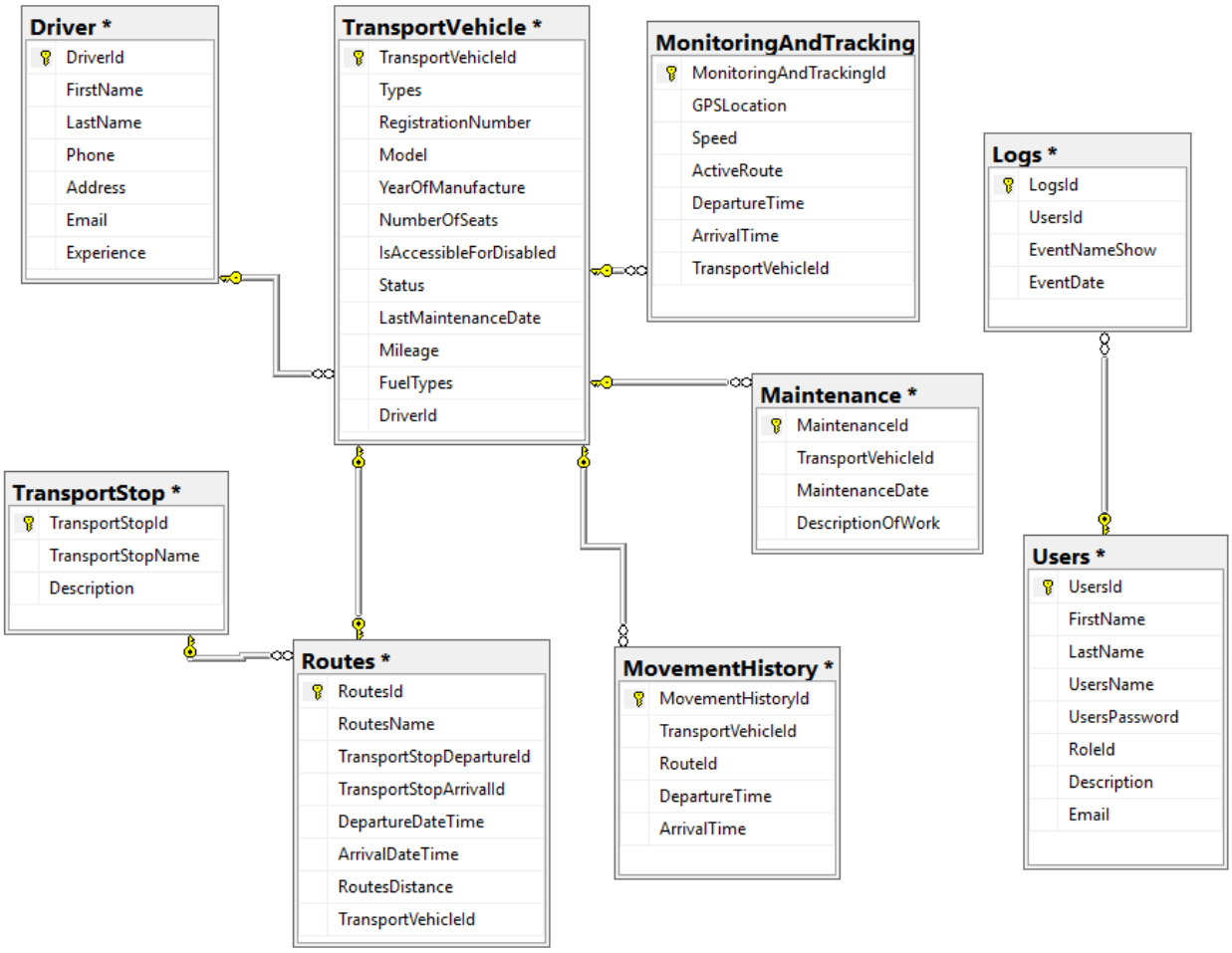

Рисунок 3.2 – Фізична модель бази даних

3.2 Розробка алгоритмів та програмного коду для контролера

Система моніторингу громадського транспорту міста базується на мікроконтролері Arduino Uno, що взаємодіє з датчиком Холла A3144 та GPS модулем NEO-6M. Для передачі даних на віддалений сервер використовується Wi-Fi модуль ESP8266.

Для реалізації задач проекту було використано декілька додаткових бібліотек для Arduino (ліст. 3.1):

– NeoGPS надає інтерфейс для роботи з GPS модулями, включаючи NEO-6M. Вона дозволяє ефективно зчитувати та обробляти NMEA сентенції, отримуючи з них географічні координати, швидкість, висоту та інші параметри. Ця бібліотека є оптимізованою за швидкістю та ресурсами, що є критично для мікроконтролерних систем;

– SoftwareSerial — це бібліотека, яка дозволяє створювати додаткові послідовні порти на пінах Arduino. Це корисно, коли основний апаратний послідовний порт вже використовується, наприклад, для дебагу або інших задач;

– TinyGsmClient — бібліотека для роботи з різними мобільними модемами, включаючи 4G, 3G, та 2G модулі. У контексті цього проекту, вона використовується для забезпечення комунікації через 4G мережу за допомогою модуля SIM7600E-H 4G HAT;

– PubSubClient — це клієнтська бібліотека для MQTT протоколу, яка може використовуватися для публікації та підписки на теми в MQTT брокері. Хоча в даному проекті прямо не вказано використання MQTT, ця бібліотека може бути корисною для реалізації подібних комунікаційних сценаріїв.

Лістинг 3.1 Підключення бібліотек до проекту

```
#include <NeoGPS.h>
#include <SoftwareSerial.h>
#include <TinyGsmClient.h>
#include <PubSubClient.h>
```

В архітектурі системи моніторингу громадського транспорту використовуються наступні піни мікроконтролера Arduino Uno (ліст. 3.2):

– hallSensorPin = 2: Цей пін використовується для зчитування сигналу з датчика Холла A3144. Датчик відповідає за вимірювання швидкості транспортного засобу;

– gpsSerial(3, 4): Піни 3 та 4 використовуються для послідовної комунікації з GPS модулем NEO-6M через SoftwareSerial. Пін 3 слугує для прийому даних (RX), а пін 4 — для передачі даних (TX);

– simSerial(5, 6): Піни 5 та 6 використовуються для послідовної комунікації з 4G модулем SIM7600E-H через SoftwareSerial. Пін 5 слугує для прийому даних (RX), а пін 6 — для передачі даних (TX).

Додаткові змінні:

– previousTime та currentTime: Ці змінні типу unsigned long використовуються для зберігання попереднього та поточного часу для вимірювання швидкості транспортного засобу;

– wheelPerimeter = 0.6283: Змінна, яка зберігає периметр колеса в метрах. Вона використовується для перерахунку обертів колеса в швидкість руху;

– speed = 0.0: Змінна для зберігання обчисленої швидкості в метрах за секунду.

Лістинг 3.2 – Оголошення змінних

```
// Піни
const int hallSensorPin = 2; // Датчик Холла на піні 2
SoftwareSerial gpsSerial(3, 4); // RX, TX
SoftwareSerial simSerial(5, 6); // RX, TX
unsigned long previousTime = 0;  // Зберігання попереднього часу
unsigned long currentTime = 0;   // Зберігання поточного часу
float wheelPerimeter = 0.6283;   // Периметр колеса в метрах
float speed = 0.0;               // Швидкість в м/с
```

Наступним кроком є реалізація коду для проведення початкового налаштування контролера з моменту його включення (ліст. 3.3).

Лістинг 3.3 – Проведення налаштування системи

```
void setup() {
  // ініціалізація Serial і WiFi
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  // Перевірка стану підключення до Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // ініціалізація пінів
```

```
  pinMode(hallSensorPin, INPUT);  // Датчик Холла
  gpsSerial.begin(9600);          // SoftwareSerial для GPS
  simSerial.begin(115200);        // SoftwareSerial для 4G модулю
  pinMode(someOtherPin, OUTPUT);
  pinMode(anotherSensorPin, INPUT);
}
```

Функція setup() в коді Arduino відповідає за ініціалізацію основних параметрів та компонентів системи. Спочатку ініціалізується серійний порт з швидкістю 115200 бод для діагностики та моніторингу. Далі система намагається підключитися до Wi-Fi мережі, інформація про яку зберігається у змінних ssid та password. Після успішного підключення до Wi-Fi ініціалізуються піни для датчика Холла, GPS модулю та 4G модулю.

Після цього реалізовано функцію sendDataToServer, яка відповідає за відправку даних про швидкість та геолокацію на віддалений сервер. Вона приймає два аргументи: швидкість та дані геолокації на вхід (ліст. 3.4).

Лістинг 3.4 – Функція для відправки даних на віддалений сервер

```
void sendDataToServer(float speed, String gpsLocation) {
  if (client.connect(server, port)) {
    String postData = "GPSLocation=" + gpsLocation + "&Speed=" + String(speed);
    client.print(String("POST ") + resource + " HTTP/1.1\r\n");
    client.print(String("Host: ") + server + "\r\n");
    client.print("Content-Type: application/x-www-form-urlencoded\r\n");
    client.print("Content-Length: " + String(postData.length()) + "\r\n\r\n");
    client.print(postData);
    unsigned long timeout = millis();
    while (client.available() == 0) {
      if (millis() - timeout > 5000) {
        Serial.println(">>> Client Timeout !");
        client.stop();
```

```
      return;
    }
  }
```

Початок функції містить спробу підключення до сервера за допомогою методу client.connect(server, port). Якщо підключення успішне, формується рядок postData, який містить параметри для HTTP POST запиту.

Далі виконується формування та відправка HTTP POST запиту на сервер. Спершу відправляється HTTP заголовок, що включає тип запиту, версію протоколу, хост та тип контенту. Після цього відправляється саме тіло запиту — postData.

Після відправки запиту, функція очікує відповіді від сервера. Якщо відповідь не надійде протягом 5 секунд, виводиться повідомлення про тайм-аут (>>> Client Timeout !), і з'єднання з сервером переривається методом client.stop().

Ця функція є ключовою для реалізації функціоналу моніторингу, оскільки саме завдяки ній дані з датчиків можуть бути передані на сервер для подальшого аналізу та обробки.

В кінці було реалізовано функцію loop, що є основною робочою функцією в Arduino коді і виконується циклічно після встановлення налаштувань. Код даної функції приведено у ліст. 3.5.

Лістинг 3.5 – Реалізація циклічної функції

```
void loop() {
  int hallState = digitalRead(hallSensorPin);  // Зчитування стану датчика
  if (hallState == HIGH) {
    previousTime = currentTime;              // Збереження попереднього часу
    currentTime = millis();                  // Збереження поточного часу
    unsigned long timeInterval = currentTime - previousTime;  // Обчислення інтервалу часу в мілісекундах
    if (timeInterval > 0) {
```

```
        speed = (wheelPerimeter / timeInterval) * 1000.0;  // Обчислення швидкості в м/с
      }
      sendDataToServer(speed, gpsLocation);
    }
    delay(100);  // Затримка для стабілізації зчитувань
  }
```

Функція loop() в Arduino коді циклічно зчитує стан датчика Холла, обчислює швидкість руху транспортного засобу та відправляє ці дані на сервер. Якщо датчик Холла виявляє високий рівень сигналу, система оновлює попередній та поточний час, обчислює інтервал часу між зчитуваннями та на цій основі розраховує швидкість. Після цього відправляється HTTP POST запит на сервер з обчисленою швидкістю та геолокацією. Функція також містить коротку затримку в 100 мілісекунд для стабілізації зчитувань.

Код цілого скетчу, що працює на стороні контролера представлений у додатку А.

3.3 Реалізація серверної частини моніторингової системи

Серверна частина буде відповідати за обробку та збереження даних, які надходять від різних датчиків через віддалені клієнтські пристрої. Така архітектура не тільки спрощує управління даними, але і забезпечує масштабованість та гнучкість системи. Для початку було розроблено код, що представляє собою клас контролера в .NET та служить для обробки HTTP POST запитів на серверній частині (ліст. 3.6).

Лістинг 3.6 – Реалізація коду для обробки HTTP POST запитів

```
[ApiController]
[Route("[controller]")]
public class MonitoringController : ControllerBase{
   [HttpPost]
   public IActionResult Post([FromForm] MonitoringModel model){
```

```
            // Вставка даних в MS SQL
        using (SqlConnection connection = new SqlConnection(connectionString)) {
                string sql = "INSERT INTO [dbo].[MonitoringAndTracking]
(GPSLocation, Speed) VALUES (@GPSLocation, @Speed)";
                using (SqlCommand cmd = new SqlCommand(sql, connection)) {
        cmd.Parameters.AddWithValue("@GPSLocation", model.GPSLocation);
                    cmd.Parameters.AddWithValue("@Speed", model.Speed);
                    connection.Open();
                    cmd.ExecuteNonQuery();
                    connection.Close();
                }
            }
            return Ok();
        }
    }
```

Контролер названо MonitoringController і він наслідується від базового класу ControllerBase. Атрибут [ApiController] вказує, що цей клас є API контролером, а [Route("[controller]")] задає шаблон маршруту для доступу до методів контролера.

Метод Post приймає модель MonitoringModel як параметр із форми ([FromForm]). Метод відповідає за вставку даних про геолокацію та швидкість у базу даних MS SQL. Для цього використовується SqlConnection для з'єднання з базою даних і SqlCommand для виконання SQL запиту. Запит SQL вставляє значення параметрів GPSLocation та Speed у таблицю MonitoringAndTracking.

Після виконання SQL запиту, з'єднання з базою даних закривається, і метод повертає статус "OK" як відгук на HTTP POST запит. Цей метод є ключовою частиною серверної логіки, яка забезпечує збереження отриманих даних для подальшого аналізу та обробки.

У подальшому етапі розробки зосереджено на створенні користувацького інтерфейсу та головної панелі навігації. Для цього до головного вікна програми було додано компонент menuStrip. Це надає змогу користувачам легко взаємодіяти з програмою, вибираючи необхідні опції та активуючи різні функції програми. Візуальний вигляд цього інтерфейсу представлено на рис. 3.3.

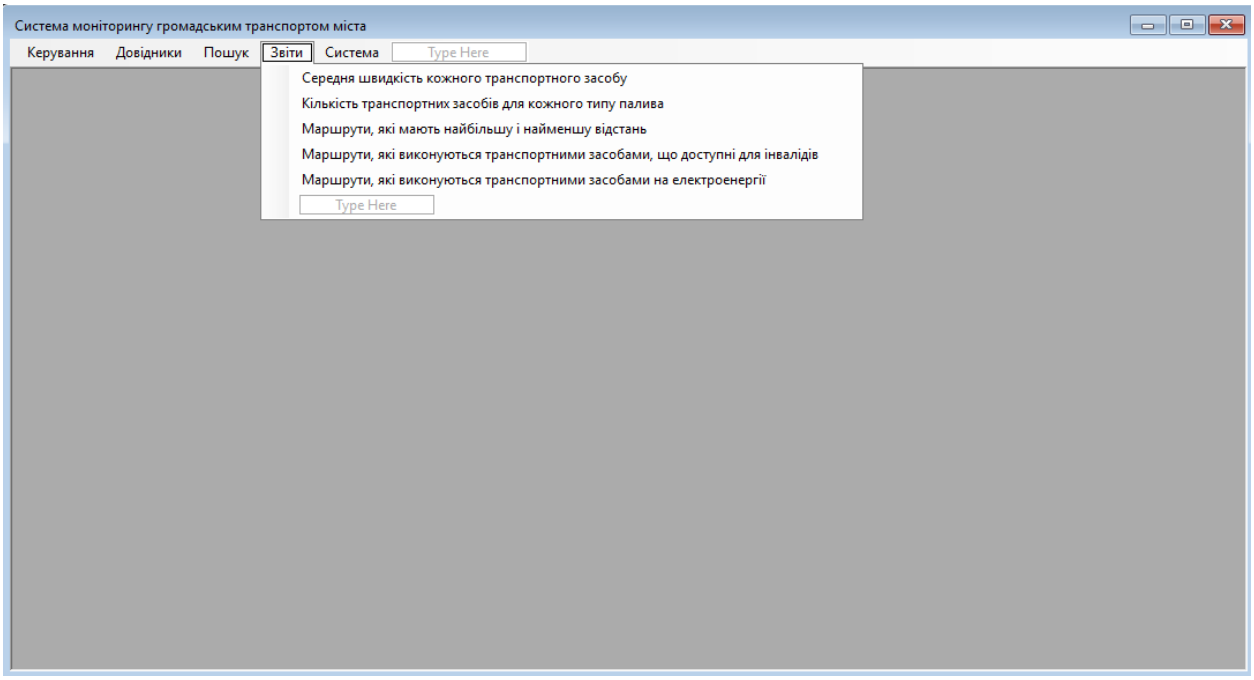

Рисунок 3.3 – Реалізація головного меню

Кожному пункту меню призначено код, що ініціалізує нову форму. При виборі пункту, відповідне вікно активується, при цьому попереднє автоматично закривається, щоб уникнути конфліктів. Такий підхід застосовано для всіх пунктів меню для забезпечення їх консистентної роботи. Приклад коду для такого механізму представлено на рис. 3.4.

```
private void транспортніЗасобиToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  TransportVehicleForm transportVehicleForm = new TransportVehicleForm();
  transportVehicleForm.MdiParent = this;
  transportVehicleForm.WindowState = FormWindowState.Maximized;
  transportVehicleForm.Show();
}
```

Рисунок 3.4 – Код додавання пункту меню

Після цього було розроблено класи для роботи із базою даних. Нижче приведено часткові реалізації методів різних класів із детальним описом.

Наприклад, для виведення даних про маршрути, які виконуються транспортними засобами, що доступні для інвалідів був створений метод із назвою «GenerateAccessibleRoutesReport, код даного методу представлено на рис. 3.5.

```
public void GenerateAccessibleRoutesReport(TextBox textBox) {
  StringBuilder reportBuilder = new StringBuilder();
  // Заголовок звіту
  reportBuilder.AppendLine("Звіт про маршрути, які виконуються транспортними засобами, доступними для інвалідів:");
  reportBuilder.AppendLine(new string('-', 80));

  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();

    string query = @"SELECT R.RoutesName, R.RoutesDistance
                     FROM Routes R
                     INNER JOIN TransportVehicle TV ON R.TransportVehicleId = TV.TransportVehicleId
                     WHERE TV.IsAccessibleForDisabled = 1
                     ORDER BY R.RoutesDistance DESC";
    using (SqlCommand command = new SqlCommand(query, connection)) {
      using (SqlDataReader reader = command.ExecuteReader()) {
        while (reader.Read()) {
          string route_name = reader["RoutesName"].ToString();
          double route_distance = Math.Round(Convert.ToDouble(reader["RoutesDistance"]), 3);
          reportBuilder.AppendLine($"Назва маршруту: {route_name}");
          reportBuilder.AppendLine($"Відстань маршруту: {route_distance} км");
          reportBuilder.AppendLine(new string('-', 80));
        }
      }
    }
  }
```

Рисунок 3.5 – Код методу «GenerateAccessibleRoutesReport»

Метод GenerateAccessibleRoutesReport відкриває з'єднання з базою даних через клас SqlConnection і виконує SQL запит, який об'єднує таблиці Routes і TransportVehicle. Запит фільтрує записи, вибираючи лише ті маршрути, де транспортний засіб доступний для інвалідів.

Після виконання SQL запиту, метод читає результати за допомогою SqlDataReader. Кожен запис перетворюється на рядок у звіті, включаючи назву маршруту та його відстань. Ці рядки додаються до StringBuilder і, в кінці виконання, весь зібраний текст виводиться в текстове поле RaportTBox на формі.

Для виявлення середньої швидкості кожного транспортного засобу був розроблений метод GenerateAverageSpeedReport (рис. 3.6). Він використовує рядок підключення до бази даних і SQL запит для збору інформації. Запит розраховує середню швидкість для кожного транспортного засобу, групуючи дані за ідентифікатором, типом та реєстраційним номером. Результати запиту потім читаються і додаються до текстового звіту. Зібраний текст виводиться в

текстове поле на формі, забезпечуючи користувачу доступ до аналітичної інформації про швидкість транспортних засобів.

```
public void GenerateAverageSpeedReport(TextBox textBox) {
  StringBuilder reportBuilder = new StringBuilder();
  reportBuilder.AppendLine("Звіт про середню швидкість транспортних засобів:");
  reportBuilder.AppendLine(new string('-', 80));

  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();

    string query = @"SELECT tv.TransportVehicleId, tv.Types, tv.RegistrationNumber, AVG(mt.Speed) AS AverageSpeed
                     FROM TransportVehicle tv
                     JOIN MonitoringAndTracking mt ON tv.TransportVehicleId = mt.TransportVehicleId
                     GROUP BY tv.TransportVehicleId, tv.Types, tv.RegistrationNumber";

    using (SqlCommand command = new SqlCommand(query, connection)) {
      using (SqlDataReader reader = command.ExecuteReader()) {
        while (reader.Read()) {
          reportBuilder.AppendLine($"ID транспортного засобу: {reader["TransportVehicleId"]}");
          reportBuilder.AppendLine($"Тип: {reader["Types"]}");
          reportBuilder.AppendLine($"Реєстраційний номер: {reader["RegistrationNumber"]}");
          reportBuilder.AppendLine($"Середня швидкість: {reader["AverageSpeed"]}");
          reportBuilder.AppendLine(new string('-', 80));
        }
      }
    }
  }
```

Рисунок 3.6 – Код методу «GenerateAverageSpeedReport»

По завершенню створення класів для обробки даних, були розроблені користувацькі інтерфейси для взаємодії з системою. Зокрема, була створена форма для організації маршрутів, яка демонструється на рис. 3.7.

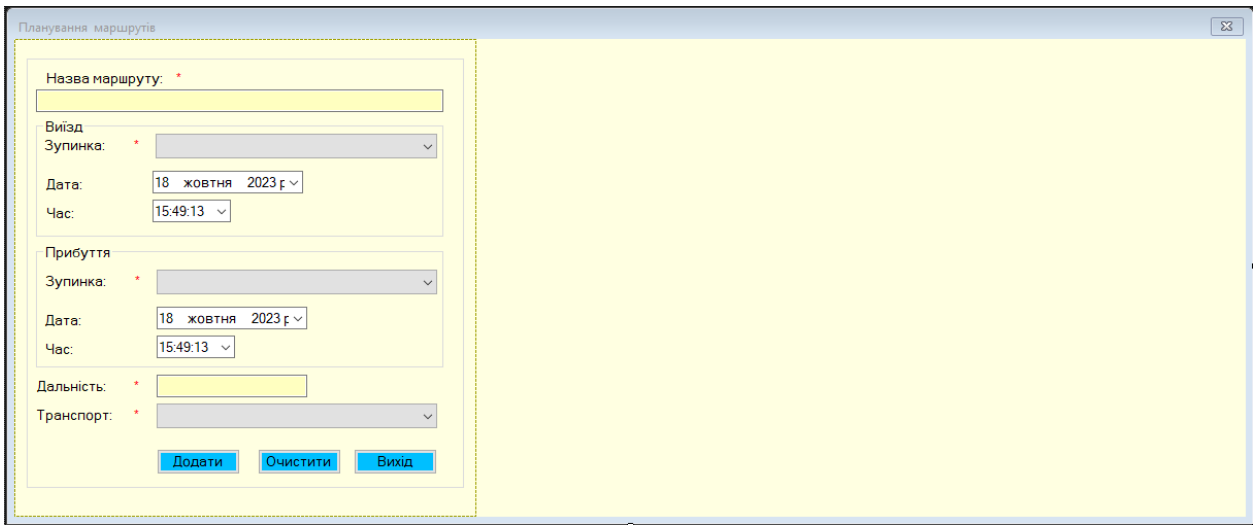

Рисунок 3.7– Форма для планування маршрутів

При ініціалізації форми, в її конструкторі активується метод "LoadAllDate". Цей метод заповнює випадаючі списки даними про транспортні засоби та зупинки, як це зображено на рис. 3.8.

```
private void LoadAllDate() {
    _TransportVehicleList = _TransportVehicleProvider.GetAllTransportVehicle();
    TransportVehicleCBox.DataSource = _TransportVehicleList;
    TransportVehicleCBox.ValueMember = "TransportVehicleId";
    TransportVehicleCBox.DisplayMember = "RegistrationNumber";

    _AirportList = _TransportStopProvider.GetAllTransportStop();
    _AirportDepartureList = GetTransportStopList(_AirportList);
    _AirportArrivalList = GetTransportStopList(_AirportList);

    TransportStopArrivalCBox.DataSource = _AirportArrivalList;
    TransportStopArrivalCBox.ValueMember = "TransportStopId";
    TransportStopArrivalCBox.DisplayMember = "TransportStopName";

    TransportStopDepartureCBox.DataSource = _AirportDepartureList;
    TransportStopDepartureCBox.ValueMember = "TransportStopId";
    TransportStopDepartureCBox.DisplayMember = "TransportStopName";

}
```

Рисунок 3.8– Код методу «LoadAllDate»

В усіх створених формах імплементована перевірка правильності введених даних. Для цього у кожній формі реалізовано спеціалізований метод "IsDataEnteringCorrect", який валідує інформацію в обов'язкових полях. Приклад методу, що здійснює перевірку в формі "RoutesForm", можна побачити на рис. 3.8.

```
private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(RoutesNameTBox.Text)) {
        RoutesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        RoutesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_validation.IsDataConvertToDouble(RoutesDistanceTBox.Text)) {
        RoutesDistanceValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        RoutesDistanceValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (Convert.ToInt32(TransportVehicleCBox.SelectedValue) > 0) {
        TransportVehicleValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        TransportVehicleValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
```

Рисунок 3.8– Фрагмент коду перевірки правильності введених даних

Для візуалізації результатів проведення моніторингу систем розроблено подію SimulTimer_Tick, , що виконується при кожному тику таймера і симулює рух транспортних засобів по маршрутах (рис. 3.9).

```
private void SimulTimer_Tick(object sender, EventArgs e) {
  DateTime currentTime = DateTime.Now; // Отримання поточного часу
  foreach (var route in _RoutesList) {
    // Перевірка, чи поточний час входить в інтервал маршруту
    // Отримання поточних координат
    if (!currentCoordinates.ContainsKey(route.RoutesId)) {
      // Ініціалізація координат, якщо це перший раз для даного маршруту
      double initialLatitude = 50.4501 + rand.NextDouble() * 0.1;
      double initialLongitude = 30.5234 + rand.NextDouble() * 0.1;
      currentCoordinates[route.RoutesId] = (initialLatitude, initialLongitude);
    }
    var (currentLatitude, currentLongitude) = currentCoordinates[route.RoutesId];
    // Оновлення координат
    currentLatitude += (rand.NextDouble() - 0.5) * 0.01;
    currentLongitude += (rand.NextDouble() - 0.5) * 0.01;
    // Зберігання нових координат
    currentCoordinates[route.RoutesId] = (currentLatitude, currentLongitude);
    // Генерація випадкової швидкості
    double speed = 40 + rand.NextDouble() * 10;
    string gpsCoordinates = $"{currentLatitude:F4},{currentLongitude:F4}";
    // Запис в базу даних
    _MonitoringAndTrackingProvider.InsertMonitoringAndTracking(route.RoutesId, gpsCoordinates, speed,
      route.RoutesName, route.DepartureDateTime, route.ArrivalDateTime);
    // Форматований вивід в текстбокс
    string output = $"Route: {route.RoutesName}, GPS: {gpsCoordinates}, Speed: {speed:F2} km/h\r\n";
    RaportTBox.AppendText(output);
  }
}
```

Рисунок 3.9– Реалізований код події SimulTimer_Tick

Спершу метод отримує поточний час системи. Потім для кожного маршруту в списку _RoutesList метод виконує ряд дій. Перевіряє, чи є вже ініціалізовані координати для даного маршруту в словнику currentCoordinates. Якщо ні, то ініціалізує їх випадковими значеннями біля заданих географічних координат.

Далі відбувається оновлення цих координат, додаванням випадкового зсуву до поточних значень широти та довготи. Ці нові координати зберігаються назад у словник. Метод також генерує випадкову швидкість для кожного маршруту.

Після цього метод викликає функцію для запису цих даних в базу даних. В кінці, він формує рядок з усією цією інформацією і додає його до текстового поля RaportTBox на формі.

Отже, в даному підрозділі було проведено частковий опис реалізованої системи.

3.4 Конфігурація та оптимізація системи

Після інсталяції Arduino IDE, Arduino UNO плата може бути з'єднана з комп'ютером через USB-кабель. Одразу після з'єднання, світлодіод "ON" на платі вмикається, а світлодіод "L" починає мигтіти, індикуючи активність мікроконтролера і заводської прошивки Blink.

Для подальшої роботи з Arduino UNO в Arduino IDE, необхідно визначити номер COM-порту, до якого підключена плата. Це можна встановити, перейшовши до Диспетчера пристроїв у Windows і знайшовши відповідний порт в секції "Порти (COM і LPT)". Там буде вказано номер порту поряд з іменем Arduino UNO (рис. 3.10).

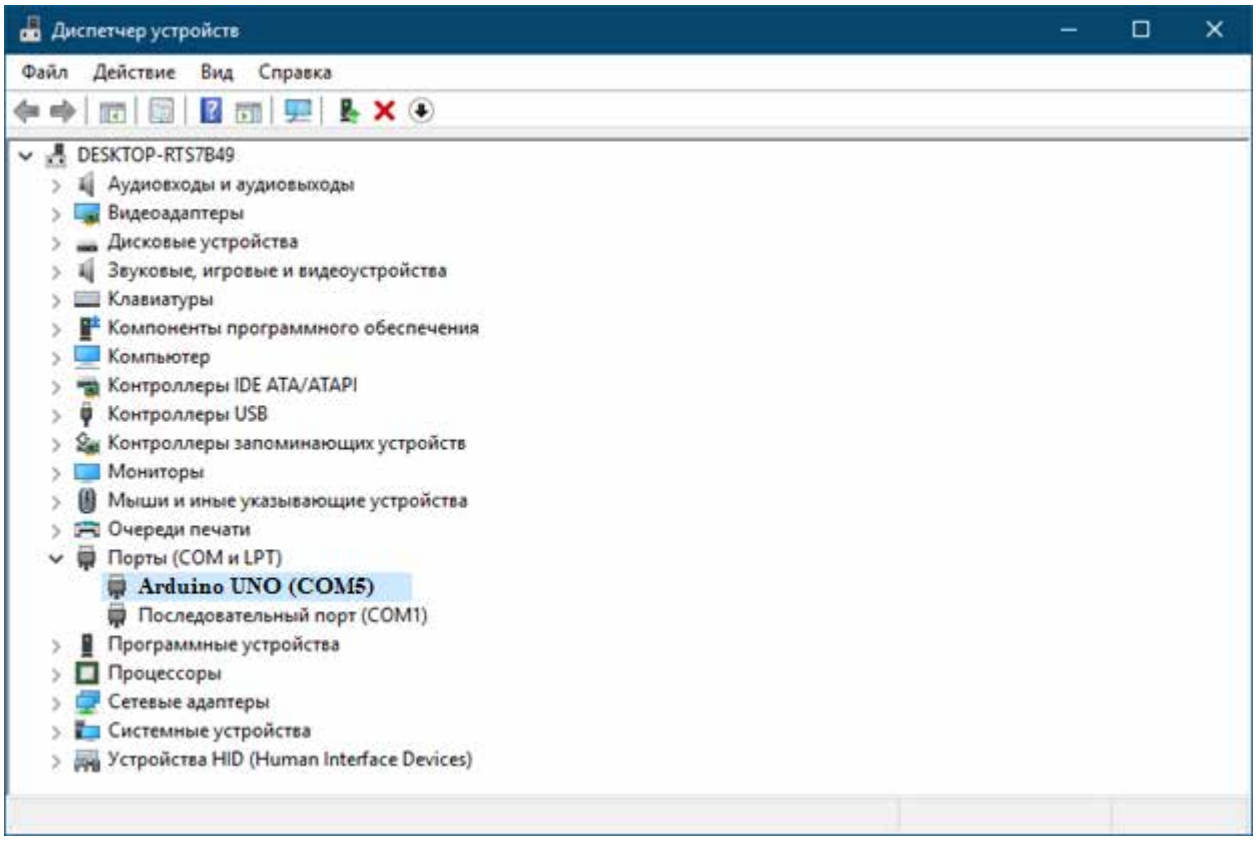

Рисунок 3.10 – Диспетчер пристроїв Windows

Після того, як Arduino UNO з'єднано з комп'ютером за допомогою USB-з'єднання, індикатор "ON" на платі вмикається, вказуючи на її активність, а світлодіод "L" миготить, сигналізуючи про виконання завантаженої на заводі прошивки Blink. Для конфігурації Arduino IDE для співпраці з Arduino UNO необхідно встановити, через який COM-порт плата з'єднана з комп'ютером. Ця інформація доступна в Диспетчері пристроїв на Windows, де показано номер

COM-порту, асоційований з платою. Цей номер може бути різним в залежності від інших підключених пристроїв. Для інформування Arduino IDE про використовуваний COM-порт, необхідно у меню "Сервіс" обрати відповідний "Послідовний порт" (рис. 3.11). Це дозволить правильно налаштувати середовище для завантаження коду на плату і моніторингу даних.

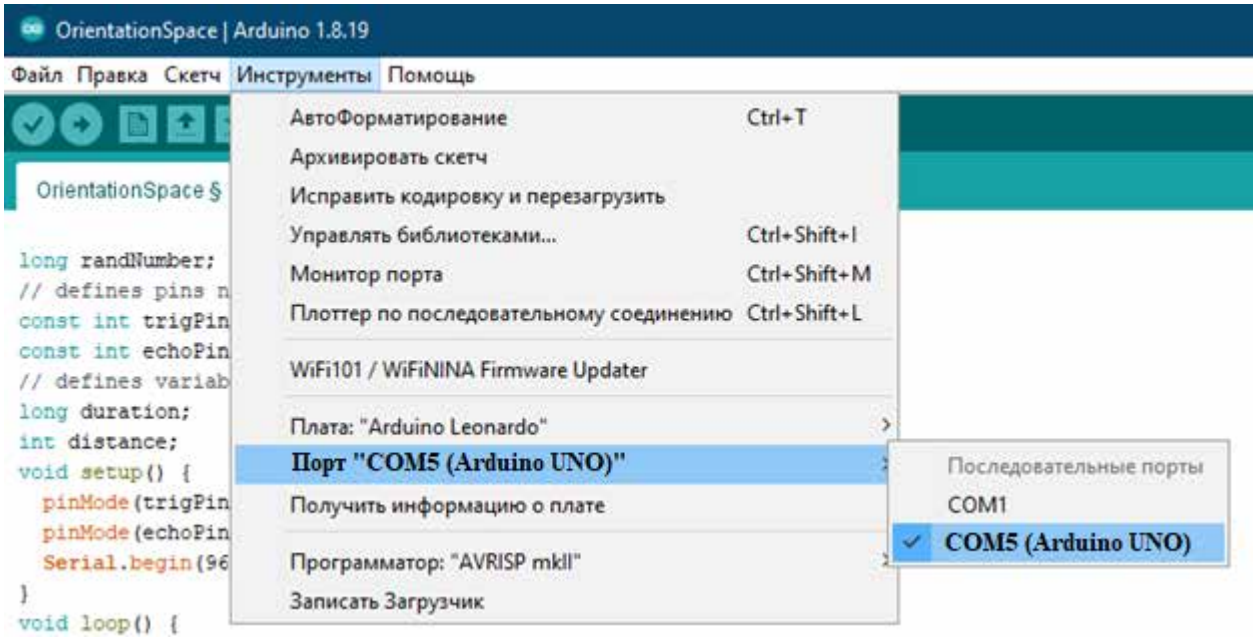

Рисунок 3.11 – Вибір порту для з’єднання контролером Arduino UNO

Після завершення процесу інсталяції Arduino IDE і підключення Arduino UNO до комп'ютера через USB-інтерфейс, важливо зробити необхідні настройки в Arduino IDE для взаємодії з підключеним мікроконтролером. Для цього, користувач повинен перейти в меню "Сервіс", де є пункт "Плата". Тут зі списку доступних плат необхідно вибрати модель "Arduino UNO". Така конфігурація дозволяє Arduino IDE розпізнати, який саме тип мікроконтролера буде використовуватися для завантаження програмного коду.

Після цього наступним кроком є компіляція програмного коду для перевірки на наявність помилок. Це можна зробити, використовуючи клавішну комбінацію "Ctrl+R". У разі успішної компіляції на екрані з'явиться відповідне повідомлення "Компіляція завершена", що підтверджує відсутність помилок у коді і готовність програми до завантаження на мікроконтролер (рис. 3.12).

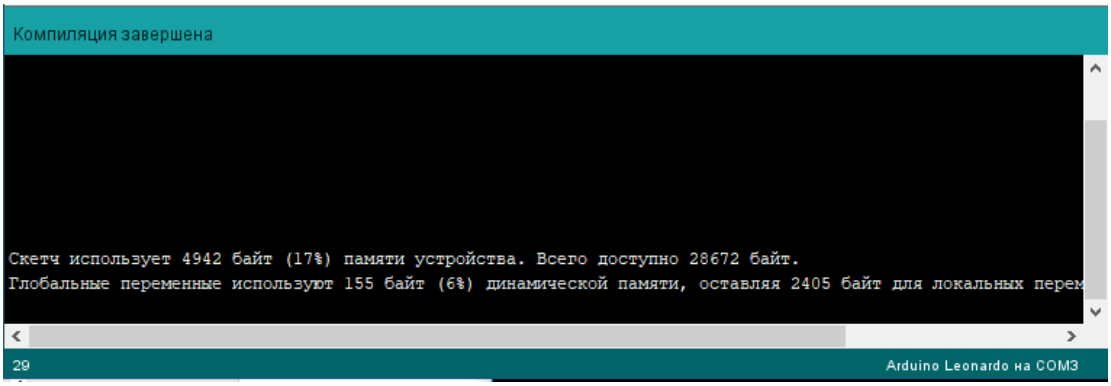

Рисунок 3.12 – Проведення компіляції проекту

Після того, як компіляція коду в Arduino IDE завершилася успішно, наступним етапом є фізичне завантаження цього програмного коду на мікроконтролер Arduino UNO. Для ініціації цього процесу, користувачу потрібно відкрити головне меню Arduino IDE і обрати пункт "Скетч". В цьому розділі меню знаходиться опція "Завантаження", яку і треба вибрати для запуску процедури передачі коду на плату (рис. 3.13). Запуск цієї опції ініціює процес завантаження, і після його завершення програма буде інстальована на мікроконтролері.

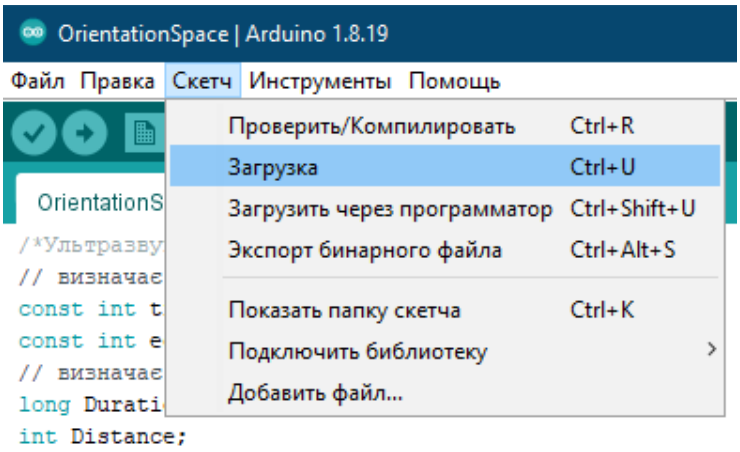

Рисунок 3.13 – Завантаження програми на контролер

По завершенню процесу передачі програмного коду на мікроконтролер Arduino UNO, у вікні Arduino IDE з'явиться спеціальне повідомлення. Це повідомлення підтверджує, що програма була успішно завантажена на плату мікроконтролера. Деталі цього повідомлення можна переглянути на рис. 3.14, де відображено всю інформацію про статус завантаження, включаючи будь-які системні повідомлення або помилки, якщо такі виникли.

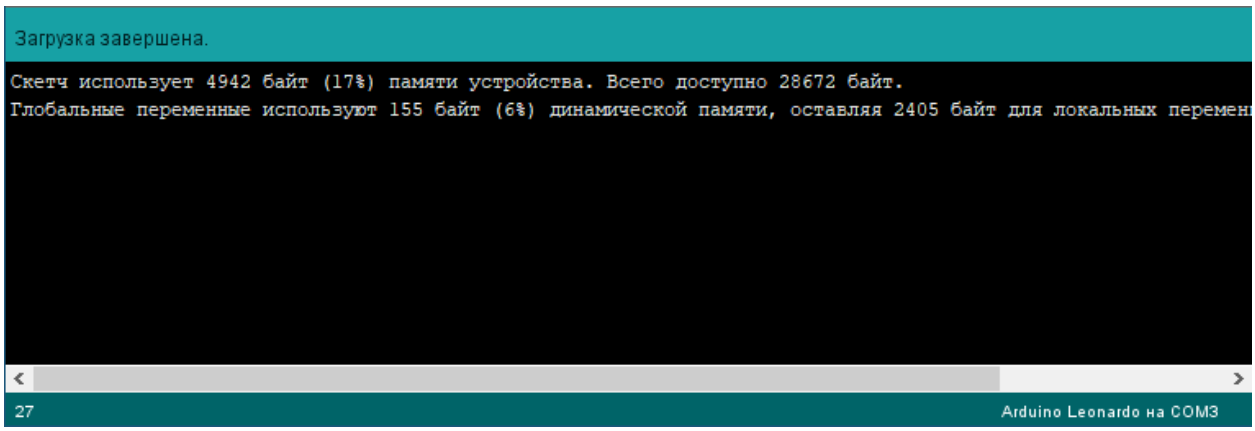

Рисунок 3.14 – Повідомлення про завантаження програми на контролер

3.5 Проведення тестів та аналіз отриманих результатів

Після проведення експериментальної перевірки розробленого програмного забезпечення для систематичного моніторингу маршрутів громадського транспорту було отримано великий об'єм даних. Ці дані включають в себе не лише геолокаційні параметри, але й динамічні характеристики руху транспортних засобів, такі як швидкість, а також час відправлення та прибуття до кінцевих пунктів маршруту. Варто підкреслити, що всі ці дані були ефективно збережені в спеціалізованій базі даних MS SQL. Такий підхід не лише забезпечив інтегритет зібраної інформації, але й дозволив виконувати швидкий та зручний доступ до даних для їх подальшого аналізу та візуалізації.

Аналіз отриманих даних виявив декілька ключових моментів, які заслуговують на увагу. По-перше, висока точність вимірювань геолокаційних та динамічних параметрів свідчить про високу надійність та валідність розробленої системи. Це підтверджує, що система може бути використана в реальних умовах для моніторингу транспортних потоків.

По-друге, швидкість реагування системи на різні зміни в руху транспорту є досить високою. Це означає, що система може надавати актуальну інформацію в реальному часі, що є критично важливим для ефективного управління транспортними мережами.

По-третє, стабільна та надійна синхронізація між серверною частиною і базою даних гарантує безперебійну роботу системи. Це дозволяє уникнути втрати даних та забезпечує високу доступність системи для різних видів аналізу та візуалізації.

Таким чином, комплексна оцінка роботи системи підтверджує її високу ефективність та готовність до впровадження в реальних умовах. На рис. 3.16 можна побачити візуалізовані дані, зібрані системою за певний період часу.

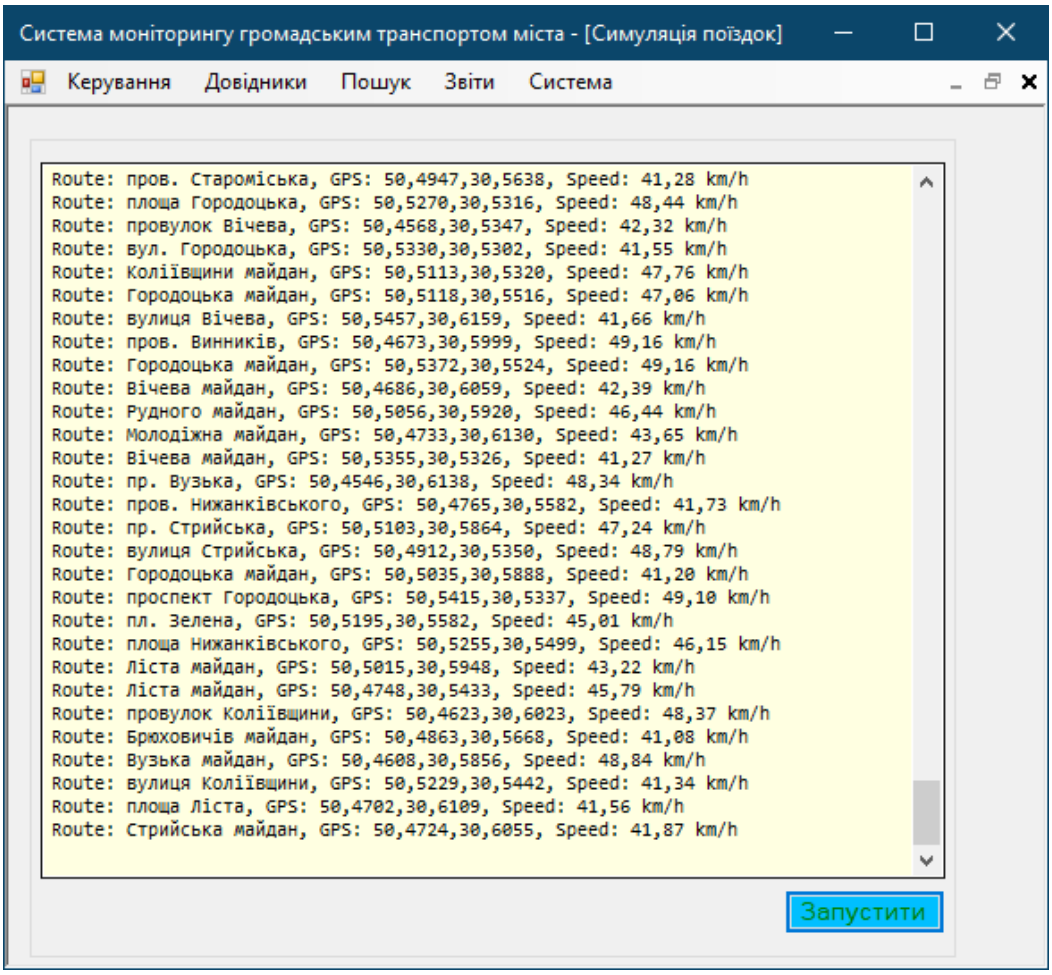

Рисунок 3.15 – Програмна візуалізація отриманих даних

На основі проведеного аналізу та візуалізації результатів можна оцінити наступне:

– система ефективно виконує свої функції, надаючи точні та своєчасні дані для моніторингу громадського транспорту;

– порівняння з аналогічними системами вказує на високу конкурентоспроможність розробленого рішення;

– виявлено потребу в оптимізації алгоритмів для ще більш швидкого реагування на зміни в руху транспорту та можливість адаптації до різних умов експлуатації.

Отже, проведене дослідження та аналіз зібраних даних підтвердили ефективність та працездатність розробленого програмного забезпечення для моніторингу транспортних маршрутів.

Висновок до розділу

Даний розділ був присвячений питанням проектування та розробки комплексної системи моніторингу громадського транспорту міста. Спершу була розроблена структура бази даних, що базувалася на діаграмі нотації Чена, та була створена її фізична модель. Це дало змогу систематично зберігати всю необхідну інформацію та забезпечити її цілісність та доступність.

Далі була здійснена розробка алгоритмів та програмного коду для контролера. Це стало основою для реалізації функціональності системи в реальних умовах. В ході розробки серверної частини були враховані всі особливості роботи з базою даних, що забезпечило надійну та ефективну обробку даних.

В секції про конфігурацію та оптимізацію були розглянуті питання налаштування параметрів системи для досягнення найкращих показників продуктивності. Це включало в себе налаштування зв'язку між сервером та контролерами, оптимізацію коду та інші технічні аспекти.

Заключним етапом стало проведення тестів та аналіз отриманих результатів. Дані, зібрані в ході тестування, підтвердили високу точність, надійність та ефективність системи. Тестування показало, що система спроможна оперативно реагувати на зміни параметрів маршруту та забезпечує надійну синхронізацію даних.

ВИСНОВКИ

Робота була присвячена проектуванню та розробці комплексної системи для моніторингу та управління громадським транспортом міста. Вона складалася з трьох ключових розділів, кожен з яких відігравав важливу роль у формуванні кінцевого продукту.

У першому розділі було здійснено глибоке дослідження бізнес-процесів в області громадського транспорту, включаючи планування маршрутів, управління транспортними засобами, забезпечення безпеки пасажирів та персоналу, а також моніторинг та контроль якості перевезень. Зокрема, було розглянуто основні актори та їх функції, а також структура основних бізнес-процесів була відображена на схемі. Було проведено аналіз схожих рішень, таких як GeoLocate Public Transport, TransitTrack та Moovit, та досліджено методи забезпечення безпеки, включаючи використання протоколу HTTPS, VPN з IPsec та JWT.

У другому розділі основний акцент було зроблено на виборі методології розробки та технічних засобів. Це питання є критично важливим, оскільки від нього залежить не тільки функціональність системи, але і її масштабованість, безпека та довгострокова підтримка. Вибір мови програмування C# було зумовлено потребами високої продуктивності, сильної типізації та вбудованої підтримки об'єктно-орієнтованого програмування.

Вибір середовища розробки — Microsoft Visual Studio — обґрунтовувався його широкими можливостями для розробки, тестування та відлагодження програмного забезпечення на платформі .NET. Щодо бази даних, вибір зупинився на Microsoft SQL Server через його надійність, високу продуктивність та гнучкість в управлінні великими обсягами даних.

Створення ER-діаграми дало змогу глибше зрозуміти структуру даних та взаємозв'язки між сутностями, що є ключовим для ефективного проектування бази даних. Трьохрівнева архітектура, що була прийнята в роботі, забезпечила чітке розділення логіки доступу до даних, бізнес-логіки та презентаційного рівня, що сприяло модульності та масштабованості системи.

Третій розділ присвячений безпосередньо процесу проектування та реалізації системи. Розробка діаграми нотації Чена та фізичної моделі бази даних на MS SQL Server стала основою для подальшої реалізації серверної частини на C#. Цей етап забезпечив стійкість і гнучкість системи, дозволяючи ефективно зберігати, обробляти та аналізувати великі обсяги інформації.

Використання платформи Arduino UNO для конфігурації та оптимізації системи показало високу ефективність в інтеграції з різними датчиками та модулями. Проведені тести та аналіз отриманих результатів не тільки підтвердили надійність системи, але і вказали на можливі вектори для подальшого її вдосконалення.

Зокрема, система продемонструвала високу точність вимірювань геопозиції та швидкості, швидке реагування на зміни умов, а також надійну синхронізацію між серверною частиною та базою даних. Це підтверджує її високий потенціал для широкого впровадження в системах громадського транспорту.

Загалом, робота є цілісним підходом до реалізації комплексної системи моніторингу громадського транспорту, від аналізу бізнес-процесів до практичної реалізації. Вона демонструє, як системний підхід, правильний вибір технологій та детальна реалізація можуть призвести до створення ефективного та надійного рішення, готового до широкого впровадження в громадському транспорті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буткявічус Й. П. Практичні рекомендації щодо удосконалення організації планування та управління місцевими пасажирськими перевезеннями / Й. П. Буткявічус, В. П. Старовойда // Вісник Національного транспортного університету. – 2003. – №8. – С. 135–137.

2. Редзюк А.М. Актуальні питання державного регулювання на автомобільному транспорті / А.М. Редзюк, В.Ф. Штанов // Автошляховик України. – 1998. – № 4. – С. 5–6.

3. Петровська С.І. Необхідність підвищення якості обслуговування пасажирів на міському пасажирському транспорті / С.І. Петровська // Вісник Національного транспортного університету. – 2003. – №8. – С. 230–234.

4. Пінчук С.О. Підвищення якості обслуговування населення на основі оптимізації маршрутної мережі / С. О. Пінчук // Вісник Національного транспортного університету. – 2003. – №8.– С. 234–237.

5. Логачов Є.Г. Мінімізація залучення перевізного ресурсу на маршруті міської пасажирської транспортної системи із урахуванням якості обслуговування пасажирів / Є.Г. Рогачов, К.Ю. Платонова // Вісник Національного транспортного університету. – 2004. – № 9. – С. 169–173.

6. Редзюк А.М. Концепція національної програми розвитку транспортно-дорожнього комплексу України до 2015 року / А.М. Редзюк, А.М. Новикова // Автошляховик України (Вісник НЦ ТАУ). – 2005. – Окремий випуск. – С. 2–7.

7. Лащених, О. А. Методи і моделі оптимізації транспортних процесів і систем. Навчальний посібник / О. А. Лащених, О. Ф. Кузькін. — Запоріжжя: ЗНТУ. — 2006. — 432 с.

8. GeoLocate Public Transport : веб-сайт. URL: https://oa.upm.es/21969/1/TFG_IVAN_GARRIDO_GOMEZ.pdf

9. TransitTrack : веб-сайт. URL:https://www.transitquote.co.uk/transittrack/

10. Moovit : веб-сайт. URL:https://moovitapp.com/index/uk/%D0%93%D1%80%D0%BE%D0%BC%D

0%B0%D0%B4%D1%81%D1%8C%D0%BA%D0%B8%D0%B9_%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82-countries

11. PPP Password Authentication Protocol | Junos OS : веб-сайт. URL:https://www.juniper.net/documentation/us/en/software/junos/network-access-protocols/topics/topic-map/interfaces-configuring-the-ppp-password-authentication-protocol.html

12. VPN (Virtual Private Network) : веб-сайт. URL:https://www.wizcase.com/top-post/what-is-a-vpn-a-beginners-guide/?keyword=what%20is%20vpn%20connection&campaignID=15356342126&matchtype=e&adgroupID=137084173091&adpos=&extension=&kwd=aud-607267681815:kwd-13525356277&location=&geo=1030310&matchtype=e&device=&ad=570860194240&placement=&adposition=&gclid=CjwKCAjw7c2pBhAZEiwA88pOF7nectbRAfifHEtzBM7R0qi52cUXee1mDOZIbqV8r0ji9HdFRZ4f7hoCTgsQAvD_BwE

13. IPsec (Internet Protocol Security) : веб-сайт. URL: https://www.cloudflare.com/learning/network-layer/what-is-ipsec/

14. JWT (JSON Web Tokens) : веб-сайт. URL: https://jwt.io/

15. JWT з HMAC: веб-сайт. URL: https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-hmac

16. Giuri L. A role-based secure database design tool [Текст] / L. Giuri, P. lglio. – Proceedings of the 12th Annual Computer Security Applications Conference, 1996. – 203–212.

17. Python : веб-сайт. URL : https://www.python.org/

18. Java : веб-сайт. URL:https://dou.ua/lenta/articles/how-to-learn-java/

19. C# : веб-сайт. URL: https://beetroot.academy/blog/courses/what-is-C

20. Visual Studio 2019 : веб-сайт. URL: https://codeguida.com/post/1754

21. .NET: веб-сайт. URL:https://training.epam.ua/ua/blog/301

22. JetBrains Rider : веб-сайт. URL : https://ua.softlist.com.ua/catalog/product-jetbrains-rider/

23. Microsoft SQL Server : веб-сайт. URL: https://www.metabase.com/data_sources/microsoft-sql-server

24. PostgreSQL : веб-сайт. URL: https://www.postgresql.org/

25. MongoDB : веб-сайт. URL: https://www.mongodb.com/cloud/atlas/lp/try4?utm_content=controlhterms&utm_source=google&utm_campaign=search_gs_pl_evergreen_atlas_core_prosp-brand_gic-null_emea-ua_ps-all_desktop_eng_lead&utm_term=mongodb&utm_medium=cpc_paid_search&utm_ad=e&utm_ad_campaign_id=12212624575&adgroup=115749710823&cq_cmp=12212624575&gad=1&gclid=CjwKCAjw-KipBhBtEiwAWjgwrB_LCe_tHR8t91IuFBIX8qTgUQJXzioF0ysU0L3MkNgDa4FsfV4yPBoCtJgQAvD_BwE

26. СУБД MS ACCESS IN 2022 : веб-сайт. URL: https://www.theaccessman.co.uk/microsoft-access-being-phased-out-2022/

27. UML Use Case Diagram Tutorial: веб-сайт. URL: https://www.lucidchart.com/pages/uml-use-case-diagram

28. What is an Entity Relationship Diagram (ERD)? : веб-сайт. URL: https://www.lucidchart.com/pages/er-diagrams

29. Creating a Business Logic Layer : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/web-forms/overview/data-access/introduction/creating-a-business-logic-layer-cs (дата звернення 11.07.2022)

30. Етапи розробки користувацького інтерфейсу: веб-сайт. URL: https://studopedia.su/12_23303_etapi-rozrobki-koristuvatskogo-interfeysuIteratsiyna-priroda-rozrobki.html (Дата звернення: 25.05.2020).

ДОДАТКИ

Додаток А. Скрипти створення бази даних

```
USE [master]
GO
/****** Object:  Database [DB]    Script Date: 17.10.2023 19:49:08 ******/
CREATE DATABASE [DB]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'DB', FILENAME = N'C:\Program Files (x86)\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\DB.mdf' , SIZE = 5120KB , MAXSIZE =
UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'DB_log', FILENAME = N'C:\Program Files (x86)\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\DB_log.ldf' , SIZE = 1024KB , MAXSIZE =
2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [DB] SET COMPATIBILITY_LEVEL = 120
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [DB].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
CREATE TABLE [dbo].[Driver](
	[DriverId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](50) NULL,
	[LastName] [nvarchar](50) NULL,
	[Phone] [nvarchar](20) NULL,
	[Address] [nvarchar](max) NULL,
	[Email] [nvarchar](max) NULL,
	[Experience] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[DriverId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Logs]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Logs](
	[LogsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[EventNameShow] [nvarchar](max) NULL,
	[EventDate] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[LogsId] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Maintenance]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Maintenance](
	[MaintenanceId] [int] IDENTITY(1,1) NOT NULL,
	[TransportVehicleId] [int] NULL,
	[MaintenanceDate] [datetime] NULL,
	[DescriptionOfWork] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[MaintenanceId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MonitoringAndTracking]    Script Date: 17.10.2023 19:49:08
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MonitoringAndTracking](
	[MonitoringAndTrackingId] [int] IDENTITY(1,1) NOT NULL,
	[GPSLocation] [nvarchar](100) NULL,
	[Speed] [float] NULL,
	[ActiveRoute] [nvarchar](150) NULL,
	[DepartureTime] [datetime] NULL,
	[ArrivalTime] [datetime] NULL,
	[TransportVehicleId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[MonitoringAndTrackingId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MovementHistory]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MovementHistory](
	[MovementHistoryId] [int] IDENTITY(1,1) NOT NULL,
	[TransportVehicleId] [int] NULL,
	[RouteId] [int] NULL,
	[DepartureTime] [datetime] NULL,
```

```
	[ArrivalTime] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[MovementHistoryId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Routes]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Routes](
	[RoutesId] [int] IDENTITY(1,1) NOT NULL,
	[RoutesName] [nvarchar](150) NULL,
	[TransportStopDepartureId] [int] NULL,
	[TransportStopArrivalId] [int] NULL,
	[DepartureDateTime] [datetime] NULL,
	[ArrivalDateTime] [datetime] NULL,
	[RoutesDistance] [float] NULL,
	[TransportVehicleId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[RoutesId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TransportStop]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TransportStop](
	[TransportStopId] [int] IDENTITY(1,1) NOT NULL,
	[TransportStopName] [nvarchar](150) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[TransportStopId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TransportVehicle]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TransportVehicle](
	[TransportVehicleId] [int] IDENTITY(1,1) NOT NULL,
```

```
	[Types] [nvarchar](50) NULL,
	[RegistrationNumber] [nvarchar](50) NULL,
	[Model] [nvarchar](50) NULL,
	[YearOfManufacture] [int] NULL,
	[NumberOfSeats] [int] NULL,
	[IsAccessibleForDisabled] [bit] NULL,
	[Status] [nvarchar](50) NULL,
	[LastMaintenanceDate] [datetime] NULL,
	[Mileage] [float] NULL,
	[FuelTypes] [nvarchar](50) NULL,
	[DriverId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[TransportVehicleId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]    Script Date: 17.10.2023 19:49:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
	[UsersId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](50) NULL,
	[LastName] [nvarchar](50) NULL,
	[UsersName] [nvarchar](50) NULL,
	[UsersPassword] [nvarchar](50) NULL,
	[RoleId] [int] NULL,
	[Description] [nvarchar](1000) NULL,
	[Email] [nvarchar](150) NULL,
PRIMARY KEY CLUSTERED
(
	[UsersId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
USE [master]
GO
ALTER DATABASE [DB] SET  READ_WRITE
GO
```

Додаток Б. Прошивка мікроконтролера

```
#include <NeoGPS.h>
#include <SoftwareSerial.h>
#include <TinyGsmClient.h>
#include <PubSubClient.h>

// Піни
const int hallSensorPin = 2; // Датчик Холла на піні 2
SoftwareSerial gpsSerial(3, 4); // RX, TX
SoftwareSerial simSerial(5, 6); // RX, TX
unsigned long previousTime = 0;  // Зберігання попереднього часу
unsigned long currentTime = 0;   // Зберігання поточного часу
float wheelPerimeter = 0.6283;   // Периметр колеса в метрах
float speed = 0.0;               // Швидкість в м/с

// Об'єкти
NMEAGPS gps;
TinyGsm modem(SerialAT);
TinyGsmClient client(modem);
PubSubClient mqtt(client);

// Змінні
float speed = 0.0;
String gpsLocation = "";
int transportVehicleId = 1; // Ідентифікатор транспорту

void setup() {
  // ініціалізація Serial і WiFi
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  // Перевірка стану підключення до Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // ініціалізація пінів
  pinMode(hallSensorPin, INPUT);  // Датчик Холла
  gpsSerial.begin(9600);          // SoftwareSerial для GPS
  simSerial.begin(115200);        // SoftwareSerial для 4G модулю
  pinMode(someOtherPin, OUTPUT);
  pinMode(anotherSensorPin, INPUT);
}

void loop() {
  int hallState = digitalRead(hallSensorPin);  // Зчитування стану датчика
  if (hallState == HIGH) {
    previousTime = currentTime;               // Збереження попереднього часу
    currentTime = millis();                   // Збереження поточного часу
    unsigned long timeInterval = currentTime - previousTime;  // Обчислення інтервалу часу в
мілісекундах
    if (timeInterval > 0) {
      speed = (wheelPerimeter / timeInterval) * 1000.0;  // Обчислення швидкості в м/с
```

```
    }
      sendDataToServer(speed, gpsLocation);
  }
  delay(100);  // Затримка для стабілізації зчитувань
}

void sendDataToServer(float speed, String gpsLocation) {
  if (client.connect(server, port)) {
    String postData = "GPSLocation=" + gpsLocation + "&Speed=" + String(speed);
    client.print(String("POST ") + resource + " HTTP/1.1\r\n");
    client.print(String("Host: ") + server + "\r\n");
    client.print("Content-Type: application/x-www-form-urlencoded\r\n");
    client.print("Content-Length: " + String(postData.length()) + "\r\n\r\n");
    client.print(postData);

    unsigned long timeout = millis();
    while (client.available() == 0) {
      if (millis() - timeout > 5000) {
        Serial.println(">>> Client Timeout !");
        client.stop();
        return;
      }
    }

    // Зчитування відповіді від сервера
    while(client.available()){
      String line = client.readStringUntil('\r');
      Serial.print(line);
    }
  }
}
```

Додаток В. Лістинги програми

Лістинг 1. Код класу «RaportingBLL»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MonitoringSystemApp.BLL {
  internal class RaportingBLL {
    public List<TransportVehicle> SearchByTransportType(List<TransportVehicle> vehicles,
string type) {
      return vehicles.Where(v => v.Types == type).ToList();
    }

    public List<TransportVehicle> SearchByPartialRegistrationNumber(List<TransportVehicle>
vehicles, string partialNumber) {
      return vehicles.Where(v => v.RegistrationNumber.Contains(partialNumber)).ToList();
    }

    public List<TransportVehicle> SearchByYearOfManufacture(List<TransportVehicle>
vehicles, int selYear) {
      return vehicles.Where(v => v.YearOfManufacture <= selYear).ToList();
    }

    public List<TransportVehicle> SearchByNumberOfSeats(List<TransportVehicle> vehicles,
int selSeats) {
      return vehicles.Where(v => v.NumberOfSeats >= selSeats).ToList();
    }

    public List<TransportVehicle> SearchByAccessibility(List<TransportVehicle> vehicles, bool
isAccessible) {
      return vehicles.Where(v => v.IsAccessibleForDisabled == isAccessible).ToList();
    }

    public List<TransportVehicle> SearchByTransportStatus(List<TransportVehicle> vehicles,
string status) {
      return vehicles.Where(v => v.Status == status).ToList();
    }



    public List<Driver> SearchByFIO(List<Driver> drivers, string FIO) {
      return drivers.Where(d => d.FIO.Contains(FIO)).ToList();
    }

    public List<Driver> SearchByPhone(List<Driver> drivers, string phone) {
      return drivers.Where(d => d.Phone.Contains(phone)).ToList();
    }

    public List<Driver> SearchByEmail(List<Driver> drivers, string email) {
      return drivers.Where(d => d.Email.Contains(email)).ToList();
    }
```

```
    public List<Driver> SearchByExperience(List<Driver> drivers, int selExperience) {
      return drivers.Where(d => d.Experience >= selExperience).ToList();
    }


  }
}
```

Лістинг 2. Код класу «SimulForm»

```
using MonitoringSystemApp.AppCode;
using MonitoringSystemApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;

namespace MonitoringSystemApp.Forms.Controls {
  public partial class SimulForm : Form {
    Random rand = new Random();
    private RoutesProvider _RoutesProvider = new RoutesProvider();
    private List<Routes> _RoutesList = new List<Routes>();
    private MonitoringAndTrackingProvider _MonitoringAndTrackingProvider = new
MonitoringAndTrackingProvider();
    // Словник для зберігання поточних координат для кожного маршруту.
    Dictionary<int, (double latitude, double longitude)> currentCoordinates = new Dictionary<int,
(double, double)>();


    public SimulForm() {
      InitializeComponent();
      _RoutesList = _RoutesProvider.GetAllRoutes();
      if (_RoutesList[0].Message != NamesMy.NoDataNames.NoDataInRoutes) {
        // Ініціалізація координат для кожного маршруту.
        foreach (var route in _RoutesList) {
          double initialLatitude = 50.4501 + rand.NextDouble() * 0.1;
          double initialLongitude = 30.5234 + rand.NextDouble() * 0.1;
          currentCoordinates[route.RoutesId] = (initialLatitude, initialLongitude);
        }
        SimulTimer.Enabled = true;
      } else {
        RaportTBox.Text = NamesMy.NoDataNames.NoDataInRoutes;
      }
    }

    private void StopBtn_Click(object sender, EventArgs e) {
```

```
      if (SimulTimer.Enabled) {
        SimulTimer.Enabled = false;
        StopBtn.Text = "Запустити";
      } else {
        SimulTimer.Enabled = true;
        StopBtn.Text = "Зупинити";
      }
    }

    private void SimulTimer_Tick(object sender, EventArgs e) {
      DateTime currentTime = DateTime.Now; // Отримання поточного часу
      foreach (var route in _RoutesList) {
        // Перевірка, чи поточний час входить в інтервал маршруту
        // Отримання поточних координат
        if (!currentCoordinates.ContainsKey(route.RoutesId)) {
          // Ініціалізація координат, якщо це перший раз для даного маршруту
          double initialLatitude = 50.4501 + rand.NextDouble() * 0.1;
          double initialLongitude = 30.5234 + rand.NextDouble() * 0.1;
          currentCoordinates[route.RoutesId] = (initialLatitude, initialLongitude);
        }
        var (currentLatitude, currentLongitude) = currentCoordinates[route.RoutesId];
        // Оновлення координат
        currentLatitude += (rand.NextDouble() - 0.5) * 0.01;
        currentLongitude += (rand.NextDouble() - 0.5) * 0.01;
        // Зберігання нових координат
        currentCoordinates[route.RoutesId] = (currentLatitude, currentLongitude);
        // Генерація випадкової швидкості
        double speed = 40 + rand.NextDouble() * 10;
        string gpsCoordinates = $"{currentLatitude:F4},{currentLongitude:F4}";
        // Запис в базу даних
        _MonitoringAndTrackingProvider.InsertMonitoringAndTracking(route.RoutesId,
gpsCoordinates, speed,
          route.RoutesName, route.DepartureDateTime, route.ArrivalDateTime);
        // Форматований вивід в текстбокс
        string output = $"Route: {route.RoutesName}, GPS: {gpsCoordinates}, Speed: {speed:F2}
km/h\r\n";
        RaportTBox.AppendText(output);
      }
    }


  }
}
```

Лістинг 3. Код класу «AccessibleRoutesReportForm»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MonitoringSystemApp.Forms.Raport {
  public partial class AccessibleRoutesReportForm : Form {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public AccessibleRoutesReportForm() {
      InitializeComponent();
      GenerateAccessibleRoutesReport(RaportTBox);
    }

    public void GenerateAccessibleRoutesReport(TextBox textBox) {
      StringBuilder reportBuilder = new StringBuilder();
      // Заголовок звіту
      reportBuilder.AppendLine("Звіт про маршрути, які виконуються транспортними
засобами, доступними для інвалідів:");
      reportBuilder.AppendLine(new string('-', 80));

      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();

        string query = @"SELECT R.RoutesName, R.RoutesDistance
                    FROM Routes R
                    INNER JOIN TransportVehicle TV ON R.TransportVehicleId =
TV.TransportVehicleId
                    WHERE TV.IsAccessibleForDisabled = 1
                    ORDER BY R.RoutesDistance DESC";
        using (SqlCommand command = new SqlCommand(query, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            while (reader.Read()) {
              string route_name = reader["RoutesName"].ToString();
              double route_distance = Math.Round(Convert.ToDouble(reader["RoutesDistance"]), 3);
              reportBuilder.AppendLine($"Назва маршруту: {route_name}");
              reportBuilder.AppendLine($"Відстань маршруту: {route_distance} км");
              reportBuilder.AppendLine(new string('-', 80));
            }
          }
        }
      }

      // Вивід згенерованого звіту у текстове поле
      textBox.Text = reportBuilder.ToString();
    }

  }
}
```

Лістинг 4. Код класу «AverageSpeedReportForm»

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MonitoringSystemApp.Forms.Raport {
  public partial class AverageSpeedReportForm : Form {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public AverageSpeedReportForm() {
      InitializeComponent();
      GenerateAverageSpeedReport(RaportTBox);
    }

    public void GenerateAverageSpeedReport(TextBox textBox) {
      StringBuilder reportBuilder = new StringBuilder();

      // Заголовок звіту
      reportBuilder.AppendLine("Звіт про середню швидкість транспортних засобів:");
      reportBuilder.AppendLine(new string('-', 80));

      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();

        string query = @"SELECT tv.TransportVehicleId, tv.Types, tv.RegistrationNumber,
AVG(mt.Speed) AS AverageSpeed
                    FROM TransportVehicle tv
                    JOIN MonitoringAndTracking mt ON tv.TransportVehicleId =
mt.TransportVehicleId
                    GROUP BY tv.TransportVehicleId, tv.Types, tv.RegistrationNumber";

        using (SqlCommand command = new SqlCommand(query, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            while (reader.Read()) {
              reportBuilder.AppendLine($"ID транспортного засобу:
{reader["TransportVehicleId"]}");
              reportBuilder.AppendLine($"Тип: {reader["Types"]}");
              reportBuilder.AppendLine($"Реєстраційний номер:
{reader["RegistrationNumber"]}");
              reportBuilder.AppendLine($"Середня швидкість: {reader["AverageSpeed"]}");
              reportBuilder.AppendLine(new string('-', 80));
            }
          }
        }
      }
```

```
      // Вивід згенерованого звіту у текстове поле
      textBox.Text = reportBuilder.ToString();
    }


  }
}
```

Лістинг 5. Код класу «FindRoutesByElectricVehiclesForm»

```
using MonitoringSystemApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MonitoringSystemApp.Forms.Raport {
  public partial class FindRoutesByElectricVehiclesForm : Form {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    private List<TransportVehicle> _TransportVehicleL = new List<TransportVehicle>();
    private TransportVehicleProvider _TransportVehicleProvider = new
TransportVehicleProvider();

    private List<TransportVehicle> _TransportVehicleSearchL = new List<TransportVehicle>();

    public FindRoutesByElectricVehiclesForm() {
      InitializeComponent();
      _TransportVehicleL = _TransportVehicleProvider.GetAllTransportVehicle();
      _TransportVehicleSearchL = FilterByElectricFuel(_TransportVehicleL);
      GenerateElectricVehicleRoutesReport(RaportTBox, _TransportVehicleSearchL);
    }

    public void GenerateElectricVehicleRoutesReport(TextBox textBox, List<TransportVehicle>
electricVehicles) {
      StringBuilder reportBuilder = new StringBuilder();

      // Заголовок звіту
      reportBuilder.AppendLine("Звіт про маршрути, які виконуються транспортними
засобами на електроенергії:");
      reportBuilder.AppendLine(new string('-', 80));

      // Список ID електричних транспортних засобів
      var electricVehicleIds = electricVehicles.Select(ev => ev.TransportVehicleId).ToList();

      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
```

```
        string query = @"SELECT RoutesName, RoutesDistance, TransportVehicleId
                         FROM Routes
                         ORDER BY RoutesDistance DESC";

        using (SqlCommand command = new SqlCommand(query, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            while (reader.Read()) {
              int vehicleId = (int)reader["TransportVehicleId"];
              if (electricVehicleIds.Contains(vehicleId)) {
                reportBuilder.AppendLine($"Назва маршруту: {reader["RoutesName"]}");
                reportBuilder.AppendLine($"Відстань маршруту: {reader["RoutesDistance"]} км");
                reportBuilder.AppendLine(new string('-', 80));
              }
            }
          }
        }
      }

      // Вивід згенерованого звіту у текстове поле
      textBox.Text = reportBuilder.ToString();
    }

    public List<TransportVehicle> FilterByElectricFuel(List<TransportVehicle> vehicles) {
      return vehicles.Where(v => v.FuelTypes.Contains("Електрика")).ToList();
    }

  }
}
```

Лістинг 6. Код класу «FuelTypeReportForm»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MonitoringSystemApp.Forms.Raport {
  public partial class FuelTypeReportForm : Form {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public FuelTypeReportForm() {
      InitializeComponent();
      GenerateFuelTypeReport(RaportTBox);
    }
```

```
    public void GenerateFuelTypeReport(TextBox textBox) {
      StringBuilder reportBuilder = new StringBuilder();

      // Заголовок звіту
      reportBuilder.AppendLine("Звіт про кількість транспортних засобів за типами палива:");
      reportBuilder.AppendLine(new string('-', 80));

      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();

        string query = @"SELECT FuelTypes, COUNT(*) AS VehicleCount
                         FROM TransportVehicle
                         GROUP BY FuelTypes
                         ORDER BY FuelTypes";

        using (SqlCommand command = new SqlCommand(query, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            while (reader.Read()) {
              reportBuilder.AppendLine($"Тип палива: {reader["FuelTypes"]}");
              reportBuilder.AppendLine($"Кількість транспортних засобів:
{reader["VehicleCount"]}");
              reportBuilder.AppendLine(new string('-', 80));
            }
          }
        }
      }

      // Вивід згенерованого звіту у текстове поле
      textBox.Text = reportBuilder.ToString();
    }

  }
}
```

Лістинг 7. Код класу «MinMaxDistanceRoutesReportForm»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MonitoringSystemApp.Forms.Raport {
  public partial class MinMaxDistanceRoutesReportForm : Form {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public MinMaxDistanceRoutesReportForm() {
```

```
      InitializeComponent();
      GenerateMinMaxDistanceRoutesReport(RaportTBox);
    }

    public void GenerateMinMaxDistanceRoutesReport(TextBox textBox) {
      StringBuilder reportBuilder = new StringBuilder();

      // Заголовок звіту
      reportBuilder.AppendLine("Звіт про маршрути з найбільшою та найменшою
відстанню:");
      reportBuilder.AppendLine(new string('-', 80));

      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();

        // Запит для знаходження маршруту з найбільшою відстанню
        string maxDistanceQuery = @"SELECT TOP 1 RoutesName, RoutesDistance
                                    FROM Routes
                                    ORDER BY RoutesDistance DESC";

        // Запит для знаходження маршруту з найменшою відстанню
        string minDistanceQuery = @"SELECT TOP 1 RoutesName, RoutesDistance
                                    FROM Routes
                                    ORDER BY RoutesDistance ASC";

        // Виконання запитів і формування звіту
        using (SqlCommand command = new SqlCommand(maxDistanceQuery, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            if (reader.Read()) {
              double maxDistance = Convert.ToDouble(reader["RoutesDistance"]);
              reportBuilder.AppendLine($"Маршрут з найбільшою відстанню:
{reader["RoutesName"]}");
              reportBuilder.AppendLine($"Відстань: {maxDistance.ToString("F3")} км");
            }
          }
        }

        reportBuilder.AppendLine(new string('-', 80));

        using (SqlCommand command = new SqlCommand(minDistanceQuery, connection)) {
          using (SqlDataReader reader = command.ExecuteReader()) {
            if (reader.Read()) {
              double minDistance = Convert.ToDouble(reader["RoutesDistance"]);
              reportBuilder.AppendLine($"Маршрут з найменшою відстанню:
{reader["RoutesName"]}");
              reportBuilder.AppendLine($"Відстань: {minDistance.ToString("F3")} км");
            }
          }
        }

        reportBuilder.AppendLine(new string('-', 80));
      }
```

```
      // Вивід згенерованого звіту у текстове поле
      textBox.Text = reportBuilder.ToString();
    }



  }
}
```

Лістинг 8. Код класу «GenerateData»

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Bogus;

namespace MonitoringSystemApp.Providers {
  internal class GenerateData {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void PopulateDrivers(int numberOfRecords) {
      // Встановлення локалі Bogus для України
      var faker = new Faker("uk");

      for (int i = 0; i < numberOfRecords; i++) {
        // Генерація випадкових даних для водіїв
        string lastName = faker.Name.LastName();
        string firstName = faker.Name.FirstName();
        string phone = faker.Phone.PhoneNumber();
        string address = faker.Address.FullAddress();
        string email = faker.Internet.Email();
        int experience = faker.Random.Number(1, 30);

        // Вставка даних в таблицю бази даних
        InsertDriver(lastName, firstName, phone, address, email, experience);
      }
    }

    public void InsertDriver(string LastName, string FirstName, string Phone, string Address,
string Email, int Experience) {
      string SqlString = "INSERT INTO Driver (LastName, FirstName, Phone, Address, Email,
Experience) " +
        "Values(@LastName, @FirstName, @Phone, @Address, @Email, @Experience)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
```

```
            cmd.Parameters.AddWithValue("@LastName", LastName);
            cmd.Parameters.AddWithValue("@FirstName", FirstName);
            cmd.Parameters.AddWithValue("@Phone", Phone);
            cmd.Parameters.AddWithValue("@Address", Address);
            cmd.Parameters.AddWithValue("@Email", Email);
            cmd.Parameters.AddWithValue("@Experience", Experience);
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
          }
        }
      }

      public void GenerateRandomTransportVehicles(int numRecords, List<Driver> drivers) {
        var faker = new Faker("uk");

        // Типи транспорту
        string[] transportTypes = { "Автобус", "Тролейбус", "Трамвай", "Спецтранспорт",
"Електрокар" };
        string[] fuelTypes = { "Бензин", "Дизель", "Електрика", "Газ" };
        string[] statusTypes = { "В експлуатації", "На ТО", "В резерві" };

        for (int i = 0; i < numRecords; i++) {
          string types = faker.PickRandom(transportTypes);
          string registrationNumber = "AA" + faker.Random.Number(1000, 9999);
          string model = faker.Vehicle.Model();
          int yearOfManufacture = faker.Random.Int(2000, 2023);
          int numberOfSeats = faker.Random.Int(10, 60);
          bool isAccessibleForDisabled = faker.Random.Bool();
          string status = faker.PickRandom(statusTypes);
          DateTime lastMaintenanceDate = faker.Date.Past();
          double mileage = faker.Random.Double(1000.0, 200000.0);
          string flTypes = faker.PickRandom(fuelTypes);

          // Вибір водія з переданого списку
          int randomDriverIndex = faker.Random.Int(0, drivers.Count - 1);
          int driverId = drivers[randomDriverIndex].DriverId;

          // Вставка даних в базу даних
          InsertTransportVehicle(types, registrationNumber, model, yearOfManufacture,
numberOfSeats,
                                 isAccessibleForDisabled, status, lastMaintenanceDate, mileage, flTypes,
driverId);
        }
      }

      public void InsertTransportVehicle(string Types, string RegistrationNumber, string Model,
    int YearOfManufacture, int NumberOfSeats, bool IsAccessibleForDisabled,
    string Status, DateTime LastMaintenanceDate, double Mileage, string FuelTypes, int DriverId)
{
        string SqlString = "INSERT INTO TransportVehicle (Types, RegistrationNumber, Model,
YearOfManufacture, NumberOfSeats," +
```

```
      " IsAccessibleForDisabled, Status, LastMaintenanceDate, Mileage, FuelTypes, DriverId) "
+
      "Values(@Types, @RegistrationNumber, @Model, @YearOfManufacture,
@NumberOfSeats, @IsAccessibleForDisabled, @Status, " +
      "@LastMaintenanceDate, @Mileage, @FuelTypes, @DriverId)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@Types", Types);
        cmd.Parameters.AddWithValue("@RegistrationNumber", RegistrationNumber);
        cmd.Parameters.AddWithValue("@Model", Model);
        cmd.Parameters.AddWithValue("@YearOfManufacture", YearOfManufacture);
        cmd.Parameters.AddWithValue("@NumberOfSeats", NumberOfSeats);
        cmd.Parameters.AddWithValue("@IsAccessibleForDisabled", IsAccessibleForDisabled);
        cmd.Parameters.AddWithValue("@Status", Status);
        cmd.Parameters.AddWithValue("@LastMaintenanceDate", LastMaintenanceDate);
        cmd.Parameters.AddWithValue("@Mileage", Mileage);
        cmd.Parameters.AddWithValue("@FuelTypes", FuelTypes);
        cmd.Parameters.AddWithValue("@DriverId", DriverId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }


  public void InsertRandomTransportStops(int numberOfStops) {
    var faker = new Faker("uk");

    for (int i = 0; i < numberOfStops; i++) {
      string randomStopName = faker.Address.StreetName();
      string randomDescription = faker.Lorem.Sentence();

      InsertTransportStop(randomStopName, randomDescription);
    }
  }

  public void InsertTransportStop(string TransportStopName, string Description) {
    string SqlString = "INSERT INTO TransportStop (TransportStopName, Description)
Values(@TransportStopName, @Description)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@TransportStopName", TransportStopName);
        cmd.Parameters.AddWithValue("@Description", Description);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
```

```
    }
  }

  public void InsertRandomRoutes(int numberOfRoutes, List<TransportVehicle>
transportVehicles, List<TransportStop> transportStops) {
    var faker = new Faker("uk");

    for (int i = 0; i < numberOfRoutes; i++) {
      string randomRoutesName = faker.Address.StreetName();
      DateTime randomDepartureDateTime = faker.Date.Recent();
      DateTime randomArrivalDateTime =
randomDepartureDateTime.AddHours(faker.Random.Int(1, 5));
      double randomRoutesDistance = faker.Random.Double(1, 100);

      int randomTransportVehicleIndex = faker.Random.Int(0, transportVehicles.Count - 1);
      int randomDepartureStopIndex, randomArrivalStopIndex;
      do {
        randomDepartureStopIndex = faker.Random.Int(0, transportStops.Count - 1);
        randomArrivalStopIndex = faker.Random.Int(0, transportStops.Count - 1);
      } while (randomDepartureStopIndex == randomArrivalStopIndex);

      int randomTransportVehicleId =
transportVehicles[randomTransportVehicleIndex].TransportVehicleId;
      int randomDepartureStopId = transportStops[randomDepartureStopIndex].TransportStopId;
      int randomArrivalStopId = transportStops[randomArrivalStopIndex].TransportStopId;

      InsertRoutes(randomRoutesName, randomDepartureStopId, randomArrivalStopId,
randomDepartureDateTime, randomArrivalDateTime, randomRoutesDistance,
randomTransportVehicleId);
    }
  }

  public void InsertRoutes(string RoutesName, int TransportStopDepartureId, int
TransportStopArrivalId,
 DateTime DepartureDateTime, DateTime ArrivalDateTime, double RoutesDistance, int
TransportVehicleId) {
    string SqlString = "INSERT INTO Routes (RoutesName, TransportStopDepartureId,
TransportStopArrivalId, " +
      "DepartureDateTime, ArrivalDateTime, RoutesDistance, TransportVehicleId) " +
      "Values(@RoutesName, @TransportStopDepartureId, @TransportStopArrivalId,
@DepartureDateTime," +
      " @ArrivalDateTime, @RoutesDistance, @TransportVehicleId)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@RoutesName", RoutesName);
        cmd.Parameters.AddWithValue("@TransportStopDepartureId",
TransportStopDepartureId);
        cmd.Parameters.AddWithValue("@TransportStopArrivalId", TransportStopArrivalId);
        cmd.Parameters.AddWithValue("@DepartureDateTime", DepartureDateTime);
        cmd.Parameters.AddWithValue("@ArrivalDateTime", ArrivalDateTime);
```

```
            cmd.Parameters.AddWithValue("@RoutesDistance", RoutesDistance);
            cmd.Parameters.AddWithValue("@TransportVehicleId", TransportVehicleId);
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
        }
      }
    }

    public void InsertRandomMaintenance(int numberOfRecords, List<TransportVehicle>
transportVehicles) {
      var faker = new Faker("uk");

      for (int i = 0; i < numberOfRecords; i++) {
        DateTime randomMaintenanceDate = faker.Date.Past();
        string randomDescription = faker.Lorem.Sentence();

        int randomTransportVehicleIndex = faker.Random.Int(0, transportVehicles.Count - 1);
        int randomTransportVehicleId =
transportVehicles[randomTransportVehicleIndex].TransportVehicleId;

        InsertMaintenance(randomTransportVehicleId, randomMaintenanceDate,
randomDescription);
      }
    }

    public void InsertMaintenance(int TransportVehicleId, DateTime MaintenanceDate, string
DescriptionOfWork) {
      string SqlString = "INSERT INTO Maintenance (TransportVehicleId, MaintenanceDate,
DescriptionOfWork) " +
        "Values(@TransportVehicleId, @MaintenanceDate, @DescriptionOfWork)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@TransportVehicleId", TransportVehicleId);
          cmd.Parameters.AddWithValue("@MaintenanceDate", MaintenanceDate);
          cmd.Parameters.AddWithValue("@DescriptionOfWork", DescriptionOfWork);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}
```