Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

Бакалаврська дипломна робота
на тему:
«Автоматизована система контролю стану автомобіля через OBD2 інтерфейс»

Виконав:студент 4 курсу, групи АКІТ-22мс
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології

_____Вадим КРУЧКО

Керівник: к.т.н., доц. каф. КСУ

_____Олена НИКИТЕНКО

«__12__» _червня__2024 р.

Рецензент: к.т.н., доцент каф. АІІТ

_____Юрій ІВАНОВ

«__12__» _червня__2024 р.

Допущено до захисту
Зав. кафедри КСУ

_____В'ячеслав Ковтун
«12 »_червня___2024р.

Вінниця ВНТУ – 2024

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо - професійна програма – Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
д.т.н., проф.
_____В'ячеслав КОВТУН
"24" _травня___2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ

студенту____Кручку Вадиму Володимировичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи Автоматизована система контролю стану автомобіля через OBD2 інтерфейс
керівник роботи доц., Никитенко Олена Дмитрівна к.т.н доцент кафедри КСУ
затверджені наказом ВНТУ № 80 від 11 березня 2024 р.
2. Термін подання студентом роботи 12 червня 2024 року
3. Вихідні дані до роботи: автоматизована система контролю стану автомобілю через OBD2 інтерфейс; алгоритми та технології для реалізації системи; кросплатформність;
4. Зміст текстової частини: огляд існуючих систем збору та аналізу інформації; огляд основних компонентів системи; характеристики передавача даних; розробка програмного продукту; особливості роботи програмного забезпечення передавача; тестування.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) взаємодія компонентів системи; базова структурна схема системи; спрощенадіаграмаархітектури серверного додатку; модельбазиданих; діагностичний сканер ELM 327.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Виконання прийняв
1	Ковтун В.В д.т.н проф.,		
2	Ковтун В.В д.т.н проф.,		
3	Ковтун В.В д.т.н проф.,		
4	Ковтун В.В д.т.н проф.,		

1. Дата видачі завдання 27.05.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Огляд існуючих систем збору	13.05.2024р.	14.05.2024р.	
2	Визначення технічних характеристик	15.05.2024р.	17.05.2024р.	
3	Розробка програмного забезпечення системи	27.05.2024р	03.06.2024р.	
4	Тестування програмного забезпечення	04.06.2024р	05.06.2024р.	
5	Оформлення пояснювальної записки, графічного матеріалу	05.06.2024р.	10.06.2024р.	
6.	Захист БДР	19.06.2024р.		

Студент _____ Вадим КРУЧКО
(підпис) (ім'я і ПРІЗВИЩЕ)

Керівник роботи _____ Олена НИКИТЕНКО
(підпис) (ім'я і ПРІЗВИЩЕ)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 88 сторінок формату А4, на яких є 23 рисунка, 4 таблиці, список використаних джерел містить 18 найменувань.

Метою роботи є створення ефективної автоматизованої системи контролю стану автомобіля через OBD2 інтерфейс.
У теоретично-методичні частині роботи розглянуто систему для збору та аналізу інформації отриманої із OBD2 роз'єму автомобіля для подальшого сповіщення власників про проблеми із технічним станом автомобіля та перевищені обмеження, встановлені власником до цього (до таких обмежень можуть відноситись: максимальні швидкість, максимальні оберти двигуна, максимальні температури різних компонентів автомобіля).

До основних функцій розробленої системи відноситься: моніторинг різних значень показників (швидкість, оберти, температура компонентів авто), виявлення перевищень встановлених обмежень, відображення статистичних даних по поїздках (включаючи витрати пального та пройдену дистанцію), перегляд помилок діагностики автомобіля.

Ключові слова: OBD2, моніторинг автомобіля, Node.js, Fastify, React.js, PostgreSQL, діагностика, збір даних, аналіз даних.

ABSTRACT

Bachelor's thesis consists of 88 pages of A4 format, which has 23 figures, 4 tables, the list of sources used contains 18 items.

The purpose of the work is to create an effective automated system for monitoring the condition of the car through the OBD2 interface.

In the theoretical and methodological part of the work, a system is considered for collecting and analyzing information received from the OBD2 connector of the car to further notify the owners of problems with the technical condition of the car and the exceeded restrictions set by the owner before that (such restrictions may include: maximum speed, maximum engine speed, maximum temperatures of various components of the car).

The main functions of the developed system include: monitoring of various values of indicators (speed, speed, temperature of car components), detection of exceeding the established restrictions, displaying statistical data on trips (including fuel consumption and distance traveled), viewing car diagnostics errors.

Keywords: OBD2, car monitoring, Node.js, Fastify, React.js, PostgreSQL, diagnostics, data collection, data analysis.

ЗМІСТ

ВСТУП

Сучасний світ постійно розвивається, особливо в сфері автомобільного транспорту. Сьогодні автомобілі використовуються у всіх сферах нашого життя: особисті потреби, оренда, службове використання та бізнес. У зв'язку з цим необхідні системи моніторингу та діагностики автомобіля, які забезпечують контроль технічного стану автомобіля, дотримання правил експлуатації автомобіля в оренду та виявлення можливих неспр авностей.

Застосування таких систем досить широке й охоплює різноманітні галузі, такі як оренда автомобілів, компанії з обслуговування автомобілів, страхові компанії та інші організації, які відповідають за стан транспортних засобів або надають транспортні засоби для використання іншим водіям.

Розробка та впровадження систем збору й аналізу автомобільних даних допоможе підвищити якість автомобільних послуг, підвищити ефективність роботи автосервісних і страхових компаній, а також допоможе власникам автомобілів краще розуміти та контролювати стан автомобіля. Впровадження цієї системи дозволить швидше виявляти та усувати несправності, скорочувати час простою автомобіля в сервісних центрах та підвищувати надійність транспорту.

Важливо також відзначити, що використання цієї системи може сприяти зниженню аварійності на дорогах шляхом контролю за дотриманням встановлених обмежень і превентивного виявлення можливих несправностей, які можуть вплинути на безпеку руху. Враховуючи велику кількість автомобілів на дорогах і високу аварійність, такі системи відіграють важливу роль у підвищенні безпеки дорожнього руху та зменшенні кількості ДТП.

Рішення, які розглядаються для цієї дипломної роботи, включатимуть розробку алгоритмів і програмного забезпечення для збору, аналізу та відображення даних, отриманих від автомобільних роз’ємів OBD2. При

розробці проекту будуть враховані сучасні тенденції у сфері автомобільного транспорту та потреби та потреби різних категорій користувачів.

Завдяки широкому застосуванню та актуальності теми, розроблена система допоможе підвищити якість надання автомобільних послуг, скоротити час простою в автосервісах та підвищити надійність перевезень.

Під час виконання цієї дипломної роботи будуть вивчені існуючі рішення, проаналізовані їх переваги та недоліки, а також розроблені власні методи для реалізації систем збору та аналізу даних, отриманих від автомобільних роз'ємів OBD2.

Тому в завданні даної дипломної роботи входить розробка відповідного програмного забезпечення, створення алгоритмів аналізу системних даних, зручних інтерфейсів для забезпечення простоти та доступності системи для різних категорій користувачів. Впровадження цієї системи збору та аналізу даних, отриманих від автомобільних роз'ємів OBD2, дозволить власникам автомобілів, орендарям, автосервісам і страховим компаніям Ефективно контролюйте та діагностуйте технічний стан транспортного засобу для підвищення безпеки та зниження ризику ДТП.

Об'єкт дослідження – процес контролю стану автомобіля.

Предмет дослідження – методи й засоби аналізу забезпечення комп'ютерної діагностики автомобіля.

Мета роботи полягає в створенні ефективної автоматизованої системи контролю стану автомобіля через OBD2 інтерфейс.

Задачі дослідження:

- зробити огляд аналогів об'єкта дослідження,
- проаналізувати аналоги й визначити параметри та характеристики створюваної системи,
- спроектувати систему, орієнтовану на досягнення поставленої мети,
- протестувати створену систему та довести її ефективність в порівняння зі знайденими аналогами.

Розроблена система для збору та аналізу інформації з OBD2 роз'єму автомобіля є інноваційною завдяки інтеграції декількох функцій у єдину платформу. Це включає реальний час моніторинг, аналітику, сповіщення та історичний перегляд даних про технічний стан автомобіля.

Інноваційність полягає в наступному:

Комплексний підхід: Система об'єднує моніторинг, діагностику, аналіз та сповіщення в одну платформу, що забезпечує зручність для користувача.

Персоналізація: Власник автомобіля може встановлювати індивідуальні обмеження для різних показників (швидкість, оберти, температура), а система буде автоматично сповіщати про їх перевищення.

Аналіз даних: Система не тільки збирає дані, але й проводить аналіз, надаючи статистичну інформацію про поїздки, витрати пального та дистанцію, що дозволяє власнику приймати більш обґрунтовані рішення щодо експлуатації автомобіля.

Діагностика в реальному часі: Система дозволяє відслідковувати та аналізувати діагностичні помилки автомобіля в режимі реального часу, що допомагає вчасно виявляти та вирішувати проблеми.

Практична цінність:

Розроблена система має значну практичну цінність для власників автомобілів завдяки наступним аспектам:

Підвищення безпеки: Швидке виявлення технічних проблем та сповіщення про перевищення встановлених обмежень допомагає уникнути аварійних ситуацій та знижує ризики, пов'язані з несправностями автомобіля.

Ефективне управління витратами: Аналіз витрат пального та дистанції допомагає власнику оптимізувати використання автомобіля, знижуючи витрати на пальне та технічне обслуговування.

Простота діагностики: Зручний перегляд діагностичних помилок та стану компонентів автомобіля дозволяє швидко виявляти та усувати несправності, що економить час та ресурси.

Персоналізований контроль: Можливість встановлення індивідуальних обмежень для різних показників дозволяє власнику адаптувати систему під свої потреби та підвищити контроль над використанням автомобіля.

Покращення експлуатації автомобіля: Система надає детальні статистичні дані про поїздки, що дозволяє власнику аналізувати та покращувати свій стиль водіння, знижуючи знос компонентів автомобіля.

Таким чином, розроблена система є не лише інноваційним рішенням, але й має велику практичну цінність, сприяючи підвищенню безпеки, ефективності та комфорту експлуатації автомобіля.

1 ВИВЧЕННЯ СИСТЕМ ЗБИРАННЯ Й АНАЛІЗУ ДАНИХ ПРО СТАН АВТОМОБІЛЯ З OBD2 РОЗ'ЄМУ

1.1 Теоретичні аспекти протоколу OBD2

Система OBD2 (бортова діагностика другого покоління) є одним з ключових інструментів для отримання інформації про стан вашого автомобіля. Це стандартна діагностична система, яка використовується в більшості сучасних автомобілів для контролю роботи різних вузлів і систем автомобіля. Цей протокол спрощує процес діагностики та виявлення проблем з електронними системами вашого автомобіля, дозволяючи отримати доступ до даних про стан вашого автомобіля за допомогою спеціального роз'єму OBD2.

Система OBD з'явилася в 1990-х роках і була призначена для контролю викидів шкідливих речовин і забезпечення екологічної безпеки. З часом OBD2 став більш досконалим і корисним і тепер надає доступ до різноманітних даних про продуктивність автомобіля.

Система використовує стандартизовані коди помилок (DTC - діагностичні коди несправностей), щоб швидко визначити причину проблеми. Ці коди допомагають визначити проблемну систему, тип проблеми та конкретний компонент або налаштування, які спричинили збій. [1]

Системи OBD2 постійно контролюють роботу критичних автомобільних систем, таких як двигун, трансмісія, система запалювання тощо. Це дозволяє автомобілю самостійно виявляти несправність і повідомляти водія, увімкнувши індикатор Check Engine або інші відповідні індикатори на панелі приладів. Крім того, система зберігає в пам'яті інформацію про роботу і несправності автомобіля. Ці дані можна отримати за допомогою сканера OBD2 для аналізу, який допомагає краще зрозуміти причину проблеми та допомагає ефективно вирішити проблему.

Роз'єм OBD2 розташований всередині автомобіля, зазвичай під або біля керма. Цей роз’єм дозволяє отримати доступ до даних системи OBD2 за допомогою спеціального пристрою або адаптера. Доступні параметри включають, але не обмежуються: швидкість автомобіля, швидкість двигуна, рівень палива, температура рідини, навантаження двигуна, стан датчика кисню тощо. Ви також можете отримати інформацію про коди помилок, щоб ви могли діагностувати проблему.

Загалом, різні системи збору й аналізу даних про транспортні засоби розроблені на основі алгоритмів і методів аналізу даних OBD2 і представляють результати аналізу в зручній для користувача графічній формі.

1.2 Огляд існуючих систем збору та аналізу інформації

1.2.1 Система моніторингу автомобіля та управління автопарком «Зубі».

«Zubie» — компанія, яка надає автоматизовані рішення для моніторингу транспортних засобів та управління автопарком. Продукція Zubie надає пристрій (рис. 1.1), який підключається безпосередньо до роз’єму OBD2 автомобіля та автоматично збирає дані автомобіля, такі як швидкість, місцезнаходження, використання палива, стан двигуна та інші параметри.

Рисунок 1.1 – Пристрій Zubie

Переглядайте зібрані дані в Інтернеті чи мобільному додатку. Користувачі можуть переглядати інформацію про свої транспортні засоби, відстежувати рухомі об'єкти, отримувати сповіщення про проблеми та планувати технічне обслуговування.

"Zubie" дозволяє виявляти проблеми з вашим автомобілем, аналізуючи зібрані дані. Користувачі можуть отримувати сповіщення про можливі проблеми з двигуном, трансмісією, електронними системами тощо. Це допомагає власникам автомобілів і автопарків уникнути дорогого ремонту та забезпечує більшу безпеку на дорозі.

Незважаючи на те, що «Zubie» пропонує багато функцій моніторингу автомобіля та управління автопарком, у порівнянні з розробленою системою є деякі недоліки. Зокрема, Zubie орієнтований насамперед на великі автопарки та корпоративних користувачів, тому він може бути менш привабливим для окремих власників автомобілів, які шукають простіше рішення для моніторингу автомобіля.

Вартість «Зубі» досить висока: 27 доларів на місяць за тарифний план, який надасть доступ до функцій системи розробки. Система, розроблена в цій роботі, могла б забезпечити український ринок нижчими витратами або більш гнучкими тарифами, що робить користувачів більш прибутковими.

Крім того, розроблена система дозволяє користувачам встановлювати власні обмеження на певні параметри, такі як максимальна швидкість, швидкість двигуна або температура компонентів автомобіля, тоді як Zubie може не запропонувати таку глибину персоналізації, зосереджуючись на стандартному моніторингу та парку автомобілів. функції управління. [2]

1.2.2 Система моніторингу автомобіля «Геотаб».

Geotab Vehicle Monitoring Systems є лідером у сфері моніторингу та управління автопарком, надаючи повний спектр продуктів і послуг для автопарків будь-якого розміру та галузей. Головним продуктом компанії є

«Geotab GO», пристрій, який підключається до порту OBD-II автомобіля для збору та передачі даних про транспортний засіб, маршрут і стан палива.

«Geotab» надає можливість відстежувати рух транспортних засобів у режимі реального часу за допомогою відстеження GPS, дозволяючи власникам автопарків контролювати маршрути та планувати оптимальні маршрути. Системи Geotab також можуть допомогти менеджерам автопарків відстежувати стиль водіння водія, виявляти агресивні маневри та оцінювати продуктивність водія на основі даних про обладнання (Рисунок 1.2).

Рисунок 1.2 – Пристрій «Геотаб».

Крім того, Geotab пропонує функцію відстеження палива, яка надає докладні звіти про споживання палива та ефективність автомобіля. Це допомагає менеджерам автопарків знаходити шляхи оптимізації використання палива та зниження витрат.

Geotab надає діагностичні звіти про стан автомобіля, включаючи несправності двигуна, температуру рідини та стан акумулятора, допомагаючи власникам і персоналу автопарку визначити потенційні проблеми та спланувати ремонт до того, як вони призведуть до простою автомобіля. Це підвищує ефективність і надійність автопарку, що дозволяє уникнути перерв у роботі.

Інтеграція цієї системи в автомобіль може бути складним процесом, який вимагає спеціально навченого персоналу. Це може включати налаштування параметрів пристрою, адаптацію до специфікацій автомобіля та вирішення будь-яких проблем із сумісністю з електронними системами автомобіля. Такий рівень складності може збільшити вартість впровадження системи Geotab, особливо для невеликих автопарків або окремих водіїв, які можуть не мати ресурсів або навичок для належної інтеграції. Крім того, для повної інтеграції системи може знадобитися встановити в транспортному засобі спеціальні датчики (Рисунок 1.3) для виконання певних функцій. Цей процес може призвести до скасування гарантії на автомобіль, особливо якщо виробник вимагає, щоб усі зміни в електронних системах автомобіля проводилися сертифікованими фахівцями або авторизованими сервісними центрами.

Рисунок 1.3 – Датчик GRRS системи «Геотаб».

На відміну від «Геотаб», розроблена в даній роботі система не потребує встановлення спеціальних датчиків, оскільки працює з наявними датчиками автомобіля через порт OBD-II. Це зменшує ризик втрати гарантії на автомобіль і забезпечує зниження витрат на впровадження системи. Крім того, розроблена система може бути більш придатною для окремих водіїв і невеликих автопарків, які шукають спосіб відстежувати та аналізувати дані

транспортного засобу без встановлення додаткового обладнання чи ризику гарантії. [3]

1.2.3 Інструмент моніторингу автомобіля «IOSiX OBD-II Data Logger»

"IOSiX OBD-II Data Logger" - це пристрій, який використовується для збору даних автомобіля, він підключається до порту OBD-II і збирає інформацію з різних датчиків і систем автомобіля. Він сумісний з різними протоколами OBD-II, що дозволяє працювати з великою кількістю марок і моделей автомобілів.

Використовуючи «IOSiX OBD-II Data Logger», ви можете збирати такі дані, як швидкість, швидкість двигуна, рівень палива та температура рідини, а потім записувати цю інформацію на SD-карту або інший зовнішній носій для подальшого аналізу за допомогою комп’ютера.

Рисунок 1.4 - Пристрій "IOSiX OBD-II Data Logger".

«IOSiX OBD-II Data Logger» має деякі обмеження порівняно з розробленою системою: розроблена система дозволяє автоматично збирати та аналізувати дані, а також встановлювати обмеження на різні параметри автомобіля.

Крім того, розроблена система використовує веб-інтерфейс і серверну частину для передачі даних і взаємодії з користувачами, що робить систему більш зручною і гнучкою. У той час як "IOSiX OBD-II Data Logger" фокусується на реєстрації даних на зовнішніх носіях, розроблена система надає користувачам можливість переглядати та аналізувати дані в режимі реального часу через веб-інтерфейс. Кожен пристрій також коштує 900 доларів, що дорого для автопарків. [4]

Проаналізувавши найпопулярніші системи збору та аналізу інформації, отриманої від автомобільних роз'ємів OBD2, можна зробити наступні висновки: Zubie надає автоматизовані рішення для моніторингу автомобіля та управління автопарком, забезпечуючи збір даних про автомобіль і дозволяючи користувачеві відстежувати рухомі об'єкти. , отримувати повідомлення про проблеми та планувати обслуговування. Однак «Zubie» також має деякі недоліки: він в основному орієнтований на великі автопарки та корпоративних користувачів, вартість тарифних планів висока, і він може не мати можливості надавати користувачам глибоко персоналізовані послуги. Порівняно з уже розробленими системами Zubie може бути менш привабливим для індивідуальних власників автомобілів, які шукають простіше рішення для моніторингу автомобіля. Розроблена система може запропонувати українському ринку нижчі витрати або більш гнучкі тарифи, а також дозволяє користувачам встановлювати власні ліміти для певних параметрів.

Про Geotab Vehicle Monitoring Systems. Незважаючи на те, що Geotab Systems є провідною компанією з управління автопарком і пропонує повний спектр продуктів і послуг, вона має деякі значні недоліки: процес інтеграції системи може бути складним і дорогим, особливо для невеликих автопарків і окремих водіїв, які вони може не мати ресурсів або навичок для належної інтеграції. Крім того, встановлення спеціальних датчиків може призвести до порушення гарантії на автомобіль.

"IOSiX OBD-II Data Logger", незважаючи на те, що він є корисним пристроєм для збору даних з автомобілів, має кілька недоліків, які обмежують його функціональність порівняно з розробленою системою: оскільки пристрій зосереджений на записі даних на зовнішній носій, користувачі не можуть контролювати та аналізувати дані. в режимі реального часу та централізовано, що робить систему незручною при встановленні на кількох автомобілях. Ціна в 900 доларів за пристрій також досить висока, що може зробити його недоступним для автопарків. Таким чином, система, розроблена в цій роботі може виявитися більш практичним і ефективним варіантом для відстеження та аналізу даних транспортного засобу.

Підсумовуючи все, можна зробити висновок, що система буде конкурентоспроможною, якщо її легко інтегрувати в автомобіль, Він відносно дешевий у встановленні та продовженні експлуатації, а також дозволяє переглядати статистику всіх поїздок (наприклад, загальну та середню витрату палива, пройдену відстань), сповіщення про перевищення значень різних параметрів автомобіля та діагностичні помилки (несправності, коди) систем автомобіля.

2 ПІДХОДИ, АЛГОРИТМИ Й ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Огляд основних компонентів системи

Система збору та аналізу даних складається з наступних частин:

- Серверна частина (backend server)
- Інтерфейс користувача (веб)
- Передача даних з автомобіля на сервер
- Діагностичний сканер

Внутрішній сервер відповідає за обробку, зберігання та аналіз даних, отриманих від автомобіля, і надання даних за запитом користувача. Це основна частина системи, і від її функціонування залежить стан усієї системи. Тому він повинен бути спроектований таким чином, щоб максимально ефективно витримувати великі навантаження.

Інтерфейс користувача (веб-сторінка) забезпечує зручний інтерфейс для перегляду та аналізу даних, отриманих від автомобіля. У його реалізації найголовніше – зручність і зрозумілість.

Передача даних між автомобілем і сервером — це зв’язок між автомобілем і внутрішнім сервером. Цей компонент системи відповідає за забезпечення стабільного з’єднання для передачі даних на сервер і з’єднання Bluetooth між діагностичними сканерами для отримання даних.

Діагностичний сканер підключається до автомобіля через роз’єм OBD2, щоб збирати дані та надсилати їх на передавач. Сканер може зчитувати різні параметри автомобіля, такі як швидкість автомобіля, частота обертання двигуна, рівень палива та інші діагностичні дані. Діагностичні сканери допомагають забезпечити точність зібраних даних і надійність їх подальшого аналізу.

Кожен аспект системи, показаний на діаграмі нижче (рис. 2.1), буде детально розглянуто нижче:

Рисунок 2.1 – Взаємодія компонентів системи

Як видно на рисунку 2.1, розроблена система буде складатися з серверної частини, передавача даних і діагностичного сканера.

2.2 Технологія та алгоритми внутрішнього сервера

2.2.1 Платформа Node.js

За основу для розробки системи збору та аналізу даних було обрано Node.js. Платформа дозволяє розробляти серверні програми за допомогою JavaScript. вона

Заснований у 2009 році Раяном Далем і заснований на механізмі виконання Google V8 JavaScript, він забезпечує високопродуктивне виконання коду JavaScript.

вузол. Node.js має багато застосувань, наприклад:

1. Створення API: Node.js часто використовується для створення RESTful API, які взаємодіють з базами даних та іншими ресурсами;

2. Розробка програм на стороні сервера: Node.js дозволяє вам створювати програми на стороні сервера, такі як веб-сайти, керовані JavaScript, де ви можете обробляти запити клієнтів і відповідати на них;

3. Реалізація мікросервісів: Node.js можна використовувати для створення мікросервісів, які підтримують модульність і гнучкість архітектури програмного забезпечення.

Загалом, Node.js — це потужний інструмент для розробки програм JavaScript на стороні сервера, який забезпечує швидкість, масштабованість і гнучкість. Він має велику спільноту, багато доступних пакетів і модулів і є кросплатформним, що робить його привабливим варіантом для розробників.

2.2.2 Переваги Node.js

Крім того, Node.js є чудовим вибором для високонавантажених API-сервісів завдяки численним перевагам і особливостям архітектури платформи:

> Однопотокова асинхронна архітектура (рис. 2.2): Node.js використовує однопотокову архітектуру, яка дозволяє обробляти кілька запитів одночасно, не блокуючи основний потік введення-виведення (I/O).

Завдяки такому підходу Node.js може ефективно обробляти велику кількість одночасних з’єднань з мінімальним навантаженням на ЦП і пам’ять, що є важливим фактором у високонавантажених службах API.

> Цикл подій: основою архітектури Node.js є цикл подій, який дозволяє обробляти асинхронні операції, такі як взаємодія з базою даних, файловою системою або стороннім API, не блокуючи основний потік. Це забезпечує високу продуктивність і масштабованість у середовищах із високим навантаженням.

> Легко інтегрувати з різними базами даних: Node.js підтримує кілька баз даних, таких як MongoDB, MySQL, PostgreSQL тощо. Це дозволяє створювати високопродуктивні служби API, які можуть взаємодіяти з різними базами даних для забезпечення швидкого доступу до інформації.

> Високий рівень масштабованості: завдяки асинхронній моделі програмування, орієнтованій на події, Node.js може масштабуватися горизонтально, додаючи додаткові вузли, або вертикально, збільшуючи ресурси кожного вузла. Це допомагає підтримувати високі навантаження та підтримує майбутнє зростання користувачів і запитів.

> Багата екосистема пакетів: екосистема Node.js включає велику кількість пакетів і модулів, доступних через NPM (Node Package Manager). Це дозволяє розробникам швидко інтегрувати готові рішення та інструменти для створення високонавантажених API-служб із мінімальними зусиллями. Відомі фреймворки, такі як Express.js, Koa та Hapi, спрощують розробку API і дозволяють створювати надійні та масштабовані програми.

> Легко розробляти та налагоджувати: Node.js надає чудові інструменти розробки коду та налагодження, які допомагають знаходити та вирішувати проблеми. Це важливо для служб API із високим навантаженням, де стабільність і продуктивність є критичними.

> Простота використання в архітектурі мікросервісів: Node.js підходить для створення мікросервісів, що дозволяє розбивати великі програми на менші, легкі та незалежні компоненти. Це допомагає зберегти модульну та

гнучку архітектуру програмного забезпечення, покращуючи розподіл навантаження та роботу високонавантажених служб API.

^ Швидкість розробки: оскільки Node.js використовує JavaScript, розробники можуть використовувати одну мову програмування як на стороні клієнта, так і на стороні сервера, що прискорює процес розробки та полегшує обслуговування коду. [5]

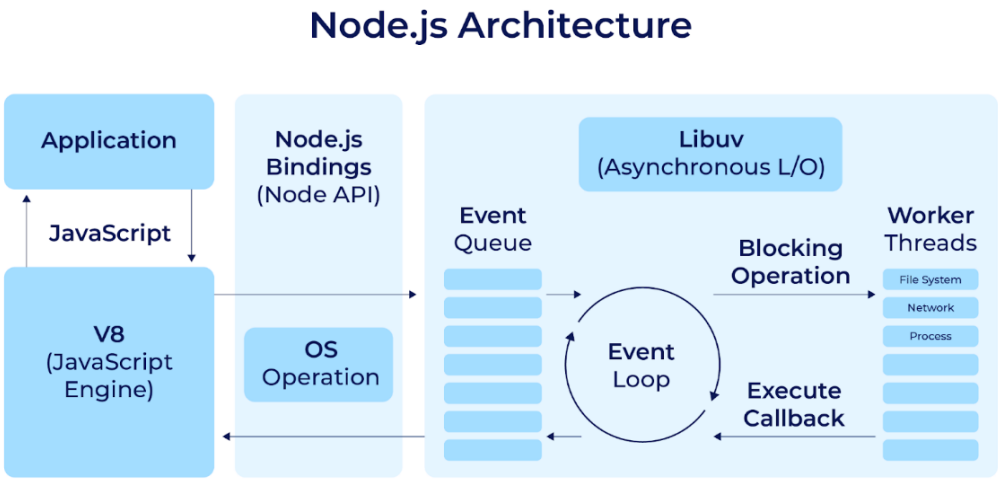


Рисунок 2.2 – Схема архітектури платформи Node.js [6]

Тому, завдяки перевагам і особливостям архітектури Node.js, платформа є відмінним вибором для створення високонавантажених API-сервісів. Створено на вузлі. Сервіс js API забезпечує швидкість,

Масштабованість і гнучкість дозволяють розробникам легко адаптуватися до зростаючих потреб і навантажень користувачів і додавати нові функції з мінімальними зусиллями.

Крім того, Node.js надає розробникам можливість використовувати відкриті стандарти для полегшення співпраці, інтеграції та взаємодії з іншими системами та службами. Це допомагає підтримувати інновації та постійне вдосконалення великих обсягів послуг API, тим самим сприяючи розвитку всієї екосистеми програмного забезпечення.

Усі ці фактори, а також широкі можливості для налаштування, розширення та інтеграції з іншими системами та службами роблять Node.js привабливим рішенням для розробки API-сервісів із високим навантаженням, які забезпечують відмінну продуктивність, стабільність та адаптивність до різних потреб і середовищ.

Загалом, можна зробити висновок, що Node.js є найкращим рішенням для розробки серверної системи для збору та аналізу даних, отриманих від автомобільних роз'ємів OBD2. Node.js відомий своєю високою продуктивністю завдяки асинхронному вводу-виводу та циклу подій, що дозволяє йому швидко обробляти дані з різних джерел, включаючи роз'єми OBD2, і надавати користувачам актуальну інформацію про стан автомобіля . Крім того, Node.js забезпечує стабільність і надійність при обробці великих обсягів даних, що має вирішальне значення для системи збору та аналізу даних роз'єму OBD2, яка повинна працювати без перебоїв. Крім того, Node.js можна легко інтегрувати з технологіями IoT.

Нарешті, завдяки модульній структурі та можливості розширення Node.js систему можна легко розширити для підтримки нових функцій, додаткових пристроїв або інших джерел даних, пов'язаних з автомобілем.

2.2.3 Фреймворк Fastify та його переваги

2.2.3.1 Огляд системи

Fastify — це веб-фреймворк Node.js, розроблений з упором на швидкість, низьке навантаження на сервер і простоту використання. Фреймворк призначений для створення ефективних і швидких веб-додатків і API-служб з мінімальним навантаженням на сервер.

Ключові переваги Fastify:

1. Висока швидкість: Fastify прагне досягти максимальної швидкості обробки запитів і надсилання відповідей, що дуже важливо для систем збору й аналізу даних, яким необхідно швидко обробляти великі обсяги інформації;

2. Низьке навантаження на сервер: Fastify має оптимізовану архітектуру, яка дозволяє зменшити навантаження на сервер і підтримувати високу продуктивність навіть при великій кількості запитів;

3. Простота використання: Fastify надає зрозумілий і простий у використанні API, що дозволяє розробникам швидко розробляти програми. Це полегшує процес розробки та підтримки коду;

4. Модульність: Fastify підтримує плагіни, що дозволяє додавати нові функції та розширювати функціональність вашої веб-програми. Це сприяє гнучкості та адаптивності систем збору та аналізу даних;

5. Вбудована підтримка схеми: Fastify має вбудовану підтримку схеми JSON, що дозволяє автоматично генерувати перевірку даних і серіалізацію. Це полегшує обробку даних і покращує безпеку служб API;

6. Спільнота та підтримка: Fastify має активну та зростаючу спільноту розробників, які створюють плагіни та діляться своїм досвідом. Це допомагає підвищити якість фреймворку, розвинути нові можливості та вдосконалити існуючі рішення;

7. Сумісність з іншими фреймворками: Fastify сумісний з багатьма іншими фреймворками, які використовуються в спільноті Node.js. Це дозволяє легко інтегрувати Fastify з існуючими рішеннями та інфраструктурою;

8. Безпека: Fastify активно розробляє та враховує найкращі методи безпеки, надаючи способи впровадження захищених від несанкціонованого доступу API.

9. Продуктивність: Fastify збільшує швидкість розробки, особливо завдяки вбудованим утилітам тестування та налагодження, що дозволяє розробникам пришвидшити процес розробки та забезпечити високу якість продукту. [7]

2.2.3.2 Порівняння між фреймворком Fastify та іншими фреймворками

Найпопулярнішими фреймворками для розробки веб-додатків на платформі Node.js є Fastify, Koa, Express, Restify і Hapi. Кожен із них має свої

особливості, плюси та мінуси, але давайте порівняємо їх у кількох ключових аспектах.

Таблиця 2.1 – Порівняння фреймворків веб-додатків Node.js

Фреймворк	**Короткий опис**	**Кількість запитів за секунду**
Fastify	Має найвищу продуктивність та швидкість завдяки оптимізації обробки запитів та використанню JSON Schema для валідації та серіалізації даних, що значно полегшує написання REST API.	21 768
Koa	Забезпечує достатню продуктивність та мінімальний набір функцій, що дозволяє додавати тільки необхідні модулі, але трохи менш швидкий в порівнянні із Fastify.	19 470
Express	Найбільш популярний фреймворк із непоганою швидкістю, але він менш продуктивний, ніж Fastify та Koa.	5 753
Restify	Спеціалізується на створенні RESTful API, має непогану швидкість та продуктивність, але менші, ніж у Fastify.	19 286
Hapi	Широко використовується для корпоративних застосунків, проте має меншу продуктивність та швидкість у порівнянні з	16 561

Таким чином, система, яка використовує Fastify для збору та аналізу даних, отриманих від автомобільних роз’ємів OBD2, надасть можливість створити швидкий, ефективний і надійний бекенд-сервер з мінімальним навантаженням на ресурси [8].

2.2.4 Архітектура ARY-SERVICE

Правильна архітектура в службах API Node.js є важливою частиною успішної розробки та підтримки програмного забезпечення. Це підвищує ефективність розробки, забезпечує стабільність і масштабованість програм, а також полегшує впровадження нових функцій і виправлення помилок.

По-перше, правильне структурування допомагає переконатися, що ваш код є чистим і легким для розуміння. Це полегшує зміну, тестування та налагодження коду.

По-друге, кваліфікована архітектура дозволяє розподілити відповідальність між різними компонентами системи, сприяючи таким чином модульності та гнучкості. Модульність дає змогу замінювати або оновлювати частини програми, не впливаючи на інші компоненти, зменшуючи ризик помилок і забезпечуючи легшу підтримку.

По-третє, правильна архітектура допомагає забезпечити масштабованість і стабільність сервера API. Масштабованість важлива для забезпечення оптимальної продуктивності програми в міру зростання вашої бази користувачів і збільшення навантаження. Стабільність забезпечує надійність сервера та забезпечує користувачам гарний досвід.

Нарешті, правильна архітектура гарантує, що програму можна підтримувати та розвивати з часом. Оскільки структура коду є чіткою та легкою для розуміння, розробники можуть легко вносити зміни, виправляти помилки та покращувати програмне забезпечення, забезпечуючи його стабільність і підтримуючи високу якість обслуговування для користувачів.

Дотримання правильної архітектури з самого початку проекту допомагає уникнути проблем, які можуть виникнути пізніше, таких як складність підтримки коду, проблеми з продуктивністю та безпекою. Тому йому приділяється особлива увага. На рисунку. 2.3 Архітектура бек-енд-сервера розробленої системи така:

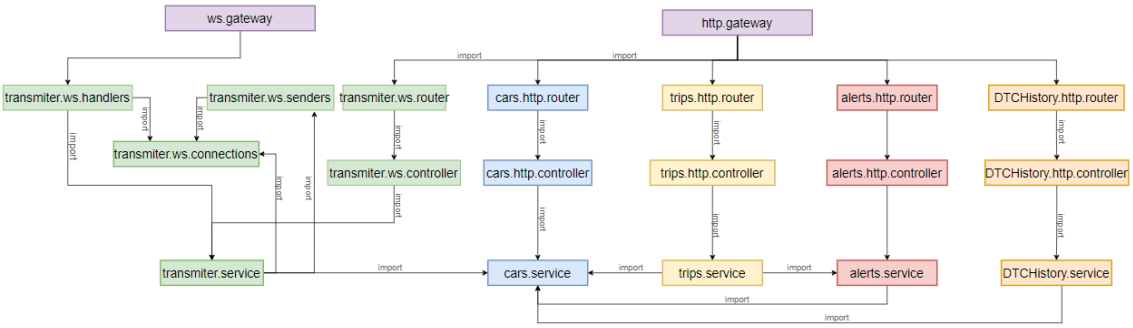


Рисунок 2.3 – Спрощена схема архітектури серверної програми

Як видно з діаграми архітектури розробленого серверного додатку (див. рис. 2.3), усі залежності сформовані у вигляді одностороннього графа і відсутні циклічні залежності. Це допомагає отримати такі переваги:

1. Чітка структура: односторонній графік забезпечує прозору та просту для розуміння структуру залежностей між модулями. Це допомагає краще зрозуміти програмний код і оптимізувати процес розробки;

2. Легко підтримувати: немає циклічних залежностей, що полегшує зміну та розширення коду. Коли залежності мають чітку структуру, легше вносити зміни в окремі модулі, не впливаючи на інші частини системи;

3. Мінімізуйте ризик: циклічні залежності можуть призвести до помилок, непередбачуваної поведінки та збоїв системи. Відсутність циклічних залежностей знижує ризик таких проблем і підвищує стабільність програми;

4. Тестування та налагодження: односторонні залежності

Полегшує написання тестів і налагодження коду. Коли модулі мають чіткі межі та відомі залежності, їх можна тестувати незалежно один від одного, покращуючи якість коду та забезпечуючи надійність програми;

5. Розподіл обов'язків: одностороння залежність

Сприяє розподілу відповідальності між модулями. Це дозволяє створити гнучку модульну архітектуру, яка спрощує розробку коду та обслуговування.

2.2.5 База даних PostgreSQL

PostgreSQL — це потужна та гнучка система керування об’єктно-реляційною базою даних, яка є чудовим вибором для обробки великих обсягів даних.

Поєднання широкої функціональності, високої продуктивності та надійності робить PostgreSQL популярною відкритою альтернативою комерційним реляційним базам даних, таким як Oracle або Microsoft SQL Server.

База даних PostgreSQL має такі переваги:

1. PostgreSQL має хорошу масштабованість і підтримує одночасне виконання запитів і паралельну обробку даних. Завдяки гнучкій системі індексування, масштабованості та можливостям реплікації він може легко обробляти великі обсяги даних і високі робочі навантаження;

2. Він має високу продуктивність завдяки різним оптимізаціям, таким як використання кешу, інтелектуальне планування запитів і ефективне використання системних ресурсів. Це дозволяє швидко обробляти великі обсяги даних і відповідати на запити в реальному часі;

3. PostgreSQL відомий своєю надійністю та стабільністю. Прийняти транзакції для підтримки відновлення після помилок, ведення журналів і багаторівневого блокування для забезпечення безпеки зберігання даних і оперативної узгодженості;

4. СУБД має відкриту архітектуру, що дозволяє розширювати її функціональність. Це включає підтримку розширень для додавання нових типів даних, функцій, операторів, індексів та інших функцій;

5. PostgreSQL має вбудовані механізми безпеки, які допомагають захистити доступ до даних. Це включає підтримку SSL/TLS для шифрування з’єднань, аутентифікацію на основі пароля або Сертифікаційні та об'єктні системи контролю доступу;

6. Завдяки підтримці різних мов програмування, PostgreSQL можна використовувати з будь-якою популярною мовою програмування, такою як Python, Ruby, Java або Node.js;

7. PostgreSQL має інтуїтивно зрозумілий синтаксис SQL і наявність різноманітних засобів керування базами даних, що допомагає в обробці великих обсягів даних;

Враховуючи ці переваги, PostgreSQL є чудовим вибором для обробки великих обсягів даних, коли вам потрібна масштабованість, продуктивність, надійність і гнучкість. Це дозволяє підтримувати складні та динамічні додатки, які вимагають відповіді на запити в режимі реального часу та аналізу великих наборів даних.

2.3 Технологія інтерфейсу користувача

2.3.1 Фреймворк React.js

Інтерфейс користувача — це веб-сторінка, написана на основі React.js. React.js — бібліотека JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача, розроблена та підтримувана Facebook. Він став дуже популярним серед розробників завдяки своїм перевагам і функціям, які полегшують розробку веб-додатків. Основні переваги React.js:

1. Компонентний метод: React. js використовує компоненти

Архітектура, яка дозволяє розбивати ваш інтерфейс на незалежні, багаторазово використовувані та модульні компоненти. Це допомагає підвищити читабельність коду та полегшує розробку та обслуговування додатків;

2. Віртуальний DOM: React.js використовує віртуальний DOM (модель об’єктів документа) для оптимізації візуалізації та оновлення веб-сторінок. Це дозволяє вам ефективно оновлювати лише ті частини сторінки, які змінилися, таким чином покращуючи продуктивність вашої програми;

3. Односторонні дані: React.js використовує односторонній потік даних, що полегшує розуміння та контроль стану програми. Це забезпечує більш структурований і передбачуваний код;

4. Розгалужена екосистема: завдяки популярності React.js він має величезну екосистему інструментів, бібліотек і розширень, які полегшують розробку та розширення функціональності вашої програми;

5. Підтримка серверного рендеринга: React.js підтримує серверний рендеринг (SSR), що може покращити швидкість завантаження сторінки та покращити пошукову оптимізацію;

6. Міжплатформна сумісність: React.js можна використовувати не лише для веб-додатків, але й для крос-платформних додатків, таких як мобільні та настільні додатки. За допомогою таких проектів, як React Native і React Desktop, розробники можуть використовувати React.js і JavaScript для створення нативних мобільних програм для iOS, Android і настільних програм для Windows, macOS і Linux;

7. Спільнота: React.js має велику й активну спільноту розробників, яка створює нові інструменти, ділиться знаннями та надає підтримку. Це робить роботу з React.js комфортнішою та забезпечує доступ до великої кількості ресурсів для полегшення навчання та вирішення проблем;

8. Легко вивчити: JavaScript є однією з найпопулярніших мов програмування, і React.js дуже легко вивчити для тих, хто вже знайомий з JavaScript. Це робить його доступним для широкого кола розробників і забезпечує швидке включення в проекти. [10]

У свою чергу, переваги та особливості React.js роблять його зручним та ефективним інструментом для розробки веб-додатків. Завдяки компонентному підходу, віртуальній DOM, розгалуженій екосистемі та активній спільноті розробників React.js дозволяє створювати ефективні, гнучкі та легко масштабовані програми з мінімальними затратами зусиль і часу.

2.3.2 Бібліотека компонентів UI матеріалу

З метою економії часу, необхідного для розробки та розміщення елементів інтерфейсу на сторінці, використовується бібліотека Material UI.

Material UI — це одна з найпопулярніших бібліотек компонентів фреймворку React.js, яка реалізує принципи Material Design — системи дизайну, розробленої Google. Матеріальний дизайн прагне створити послідовний, привабливий і функціональний дизайн на всіх платформах і пристроях.

Material UI надає готові до використання компоненти, стилі та значки, які дозволяють розробникам швидко створювати візуально привабливі веб-додатки. Бібліотека містить такі компоненти, як кнопки, віджети введення, списки, меню, панелі, картки тощо, усі вони оформлені відповідно до вказівок Material Design. Основні переваги Material UI включають:

1. Готові компоненти: Material UI надає велику кількість готових компонентів, які можна використовувати в програмах, значно прискорюючи процес розробки та забезпечуючи узгодженість дизайну;

2. Налаштування: Material UI дозволяє легко налаштовувати компоненти та стилі за допомогою тем, щоб адаптувати вашу програму до вашого бренду та інших конкретних вимог;

3. Легко інтегрувати: оскільки Material UI спеціально розроблений для React.js, інтеграція компонентів у програму є простою та природною;

4. Спільнота та підтримка: Material UI має активну спільноту та документацію, яка допомагає розробникам знаходити відповіді на запитання та вирішувати проблеми. [11]

Загалом Material UI є потужним інструментом для розробки веб-додатків React.js, що дозволяє створювати привабливі, зручні та узгоджені інтерфейси.

2.4 Платформа надсилання даних

Передавач даних у розробленій системі є одноплатним мікрокомп’ютером або мікрокомп’ютером, який отримує дані від діагностичного сканера, перетворює їх у формат, необхідний серверу, і передає на сервер, який потім передає дані на сервер. Обробляється сервером.

Для цього компонента важливо, щоб його платформа була не дорогою, але при цьому продуктивною. Крім того, при виборі пристрою слід враховувати, що код і алгоритми передавача будуть написані на мові Python. На основі цього ви можете сформувати список можливих одноплатних комп’ютерів, які відповідають вашим вимогам: Orange Pi PC і Raspberry Pi 3 Model A+.

ПК Orange Pi виглядає так, як показано на рисунку 1. 2.4:

Рисунок 2.4 – Зовнішній вигляд ПК Orange Pi [12]

Міні-комп’ютер Raspberry Pi 3 Model A+ виглядає схожим на ПК Orange Pi, як показано нижче:

Рисунок 2.5 – Зовнішній вигляд Raspberry Pi 3 Model A+ [13]

Нижче буде проведено порівняння на основі технічних характеристик цих двох одноплатних комп'ютерів.

Таблиця 2.2 – Порівняння Raspberry Pi 3 Mod. A+ та Orange Pi PC

Характеристика	**Orange Pi PC**	**Raspberry Pi 3 Model A+**	**Висновок**
Процесор	Allwinner H3 Quad-core Cortex-A7, 1.2 ГГц	Broadcom BCM2837B0, Quadcore Cortex-A53, 1.4 ГГц	Raspberry Pi 3 Model A+ має трохи більш потужний процесор порівняно з Orange Pi PC
Графіка	ARM Mali-400 MP2	Broadcom VideoCore IV	Обидва пристрої мають схожі можливості щодо графічної підтримки.
Оперативна пам'ять	1 ГБ DDR3	512 МБ LPDDR2	Orange Pi PC має більше оперативної пам'яті, що може забезпечити кращу багатозадачність та продуктивність.

Підтримка носіїв	microSD карта	microSD карта	Обидва пристрої використовують microSD карти для зберігання даних.
Підтримка носіїв	microSD карта	microSD карта	Обидва пристрої використовують microSD карти для зберігання
Мережеві можливості	10/100 Ethernet	2.4 GHz 802.11 b/g/n Wi-Fi, Bluetooth 4.2, 10/100 Ethernet	Raspberry Pi 3 Model A+ має Wi-Fi та Bluetooth, тоді як Orange Pi PC має
Інтерфейси	3x USB 2.0, 1x HDMI, 1x AV, 1x CSI, 40-пін роз'єм	USB 2.0, 1x HDMI, 1x CSI, 1x DSI, 40-пін роз'єм GPIO	Orange Pi PC має більше USB портів, але Raspberry Pi 3 Model A+
Ціна	1800 грн	2700 грн	Orange Pi PC може бути більш вигідним варіантом з точки зору
Спільнота та підтримка	Має меншу спільноту користувачів,	Має велику спільноту, яка підтримує та	

Для прискорення розробки системи Raspberry Pi 3 Model A+ було обрано як платформу передачі даних автомобіля. На це впливають дві основні переваги: вбудовані модулі Wi-Fi і Bluetooth, які будуть використовуватися для встановлення зв'язку з діагностичним сканером.

Враховуючи, що однією з цілей цієї роботи є зробити продукт значно дешевшим, ніж у конкурентів, необхідно враховувати можливість майбутніх змін платформи, тому програму передавача слід розробляти таким чином, щоб він міг працювати на різних платформах.

2.5 Діагностичний сканер

Вибрати діагностичний сканер набагато простіше, ніж вибрати платформу передачі даних, оскільки на ринку вже є лідер продажів – ELM 327 (рис. 2.6).

Рисунок 2.6 – Діагностичний сканер ELM 327 [14]

ELM-327 - універсальний адаптер для автомобільної діагностики. Цей пристрій популярний і широко використовується з наступних причин:

1. Сумісність протоколів: цей сканер підтримує різні протоколи, які використовуються різними автомобілями, наприклад ISO 9141-2, ISO 14230-4 (KWP2000), SAE J1850 (VPW і PWM) і ISO 15765-4 (CAN). Це робить пристрій універсальним і сумісним з більшістю автомобілів, випущених після 1996 року;

2. Простота у використанні: ELM-327 можна легко підключити до роз'єму OBD-II (бортова діагностика) автомобіля, що дозволяє отримати діагностичну інформацію, не розбираючи автомобіль. Після підключення пристрою до роз'єму OBD-II і смартфона або комп'ютера через Bluetooth, Wi-Fi або USB користувачі можуть використовувати спеціальний додаток для отримання діагностичної інформації;

3. Широкий спектр функцій: Сканер дозволяє користувачам отримувати різноманітну інформацію про автомобіль, включаючи коди помилок, реальну продуктивність двигуна та інші діагностичні дані;

4. Бездротове з'єднання: ELM-327 може забезпечувати різні типи бездротових з'єднань, такі як Bluetooth і Wi-Fi, тобто передавач даних не буде підключатися до сканера за допомогою кабелів, що значно підвищує зручність і простоту встановлення системи. в автомобілі. . Крім того, це дозволяє встановити передавач у будь-якому місці автомобіля. Це корисно при розширенні передавача системою GPS у майбутньому, оскільки дозволяє приховати передавач, що корисно для безпеки автомобіля: у разі крадіжки зловмисник може навіть не знати, що такий система існує. [15]

Тому цей діагностичний сканер є хорошим вибором, особливо враховуючи його універсальність і ціну.

У цьому розділі описані основні компоненти системи та їх взаємодія, платформа Node.js (основа для бекенд-сервера), фреймворк Fastify.js (основа для побудови API), архітектура компонентів бекенд-сервера, фреймворк React (Основи інтерфейсів користувача), бази даних і фізичні компоненти: платформи передачі даних і діагностичні сканери.

Варто відзначити, що при виборі технічних і фізичних компонентів особливий акцент був зроблений на швидшій розробці системи та її архітектури, щоб забезпечити подальшу легку заміну компонентів. Це найбільше стосується платформ даних, оскільки на них припадає найбільша частка загальних витрат на інтеграцію систем у транспортні засоби. Цього разу був обраний одноплатний комп’ютер Raspberry Pi 3 Model A+. Його головна перевага полягає в тому, що він має вбудовані засоби бездротового зв’язку, такі як Wi-Fi і Bluetooth, які відіграють велику роль в інтеграції системного обладнання. автомобіль.

З іншого боку, можна вибрати комбінацію ПК Orange Pi із зовнішнім адаптером Bluetooth, але при цьому можуть виникнути проблеми, пов’язані з цією платформою: драйвери та сумісність із зовнішніми пристроями. Насправді додаткову вартість обладнання можна компенсувати простоєм і варіантами встановлення, але це також відкриває багато потенційних можливостей, таких як приховування Автомобільний передавач даних або

використовуйте зовнішній порт USB для підключення різних пристроїв, наприклад модуля GPS.

Загалом у цьому розділі формується стек технологій і вибираються фізичні компоненти для подальшого розвитку системи.

3 РЕЗУЛЬТАТИ ПРОЕКТУВАНННЯ СИСТЕМИ

3.1 Характеристики передавача даних

3.1.1 Зв'язок між передавачем і сервером

3.1.1.1 Інструменти для встановлення мобільного з'єднання

Встановлення підключення до сервера для передачі даних з Raspberry є одним із найскладніших завдань цієї системи. Це пов’язано з тим, що автомобіль не стоїть на місці, а постійно перебуває в русі: він часто їздить по дорогах без зв’язку або може перетинати кордони з іншими країнами та потребує Інтернету для передачі даних.

Є кілька способів забезпечити мобільний інтернет для авто:

1. Мобільний маршрутизатор Wi-Fi: це портативний пристрій,

Для створення використовуйте мобільний Інтернет (3G, 4G, 5G).

Точка бездротового доступу Wi-Fi. Ви можете придбати такий роутер, встановити сім-карту мобільного оператора і розмістити її в машині і підключити передавач даних через Wi-Fi;

2. Смартфони з функціями точки доступу: більшість смартфонів можуть створити точку доступу Wi-Fi через мобільний Інтернет, і ви можете підключити передавач даних, як мобільний маршрутизатор Wi-Fi;

3. Плата розширення Raspberry Pi 3G: модуль, який дозволяє підключати Raspberry Pi до мобільної мережі 3G для передачі даних;

4. Портативний USB-модем: ця опція доступна лише для пристроїв із портами USB і доступна в Raspberry.

З усіх цих варіантів, враховуючи, що однією з початкових цілей було здешевлення таких систем, найбільш підходящим є портативний USB-модем, оскільки цей варіант найдешевший: плата розширення для SIM-карти коштує близько 2800 грн., а USB-модем 3G коштує близько 500 гривень, і оскільки Raspberry Pi 3 Model A+ був обраний з самого початку, він має вбудовані

модулі бездротового зв'язку Bluetooth і Wi-Fi, тому його єдиний USB-вхід можна використовувати Connect USB modem.

Вибираючи цю опцію, слід враховувати, що в подальшому вам доведеться сплачувати додаткові щомісячні суми моніторингу та тарифного плану.

3.1.1.2 Протокол передачі даних

Дані в цій системі будуть передаватися за допомогою протоколів HTTP і WebSocket.

HTTP — це протокол без збереження стану, що означає, що кожен запит обробляється окремо, і сервер не зберігає жодної інформації про попередні запити. Протокол HTTP використовується як основний протокол для передачі туристичних пакетів, з якого генеруються попередження та статистика про порушення лімітів. Він добре підходить для цих цілей, тому що передача таких даних не вимагає постійного з'єднання з сервером і легко перевірити, чи був запит успішним. Це особливо важливо в цій реалізації, оскільки мобільні з’єднання не очікуються стабільними.

WebSocket — це протокол, який дозволяє встановити двостороннє з'єднання між клієнтом і сервером для передачі даних у реальному часі. У цій системі на початковому етапі він використовується для надсилання команд на передавач даних для підключення діагностичного сканера. Ви також можете використовувати його пізніше для надсилання команд для очищення діагностичних помилок.

3.1.2 Зв'язок між передавачем даних і діагностичним сканером і його налаштування

3.1.2.1 Характеристики встановлення з'єднання

У першій версії впровадження системи планується встановити зв'язок з діагностичним сканером через Bluetooth-з'єднання (це може бути розширено

в майбутньому для встановлення з'єднання через Bluetooth і Wi-Fi). Це пояснюється тим, що такі діагностичні сканери, як ELM-327, у більшості випадків повністю підтримують ці протоколи передачі даних. В основному діагностичні сканери, що використовують протокол передачі даних Bluetooth, дешевші, ніж ті, що використовують модулі Wi-Fi. Швидкість передачі даних через Wi-Fi вище, ніж через Bluetooth, але це значення не стосується цієї системи.

Таким чином, підключення передавача даних до діагностичного сканера буде встановлено за допомогою протоколу Bluetooth. У протоколу є свої особливості: для початку передачі даних пристрої повинні «спаритися» один з одним. Для цього потрібно:

1. Скануйте навколишні пристрої
2. Виберіть пристрій
3. Надіслати запит на підключення

Слід зазначити, що такі сканери зазвичай вимагають пароль для підключення, тому при розробці системи все було враховано.

3.1.2.2 Реалізація

Для з’єднання двох пристроїв за допомогою протоколу Bluetooth об’єднуються два протоколи: HTTP і WebSocket. Підключення відбувається в 3 етапи:

1. Скануючий пристрій навколо передавача
2. Виберіть потрібний пристрій
3. Спробуйте встановити з'єднання

При спробі встановити з'єднання пристрій може запитати пароль. Це також регулюється.

Загалом, для реалізації цієї функціональності потрібно написати спеціальний модуль, основним завданням якого є надсилання команд до передавача та очікування відповіді.

Послідовність дій, які виконуються під час спроби підключити відправник даних до діагностичного сканера в модулі запису, така:

1. Надішліть HTTP-запит з інтерфейсу користувача для пошуку всіх пристроїв Bluetooth, що знаходяться поруч.

2. Коли сервер отримує запит, він перевіряє, чи транспорт, який надсилає запит, зареєстрований у системі, і якщо відправника знайдено, сервер надсилає повідомлення відправнику через протокол WebSocket і чекає заданий час для відповідь. Коли пристрій сповіщає сервер через WebSocket про те, що сканування триває, зворотний відлік перезапускається протягом указаного часу та чекає на відповідь пристрою. Якщо відповіді немає, повертається відповідь на запит HTTP, час очікування якого минув.

3. Передавач ініціалізує сканування пристроїв Bluetooth навколо нього та повідомляє сервер про те, що сканування триває кожну 1 секунду. Після завершення сканування передавач повертає результати сканування у вигляді повідомлення через протокол WebSocket.

4. Після отримання результатів сканування сервер повертає відповідь на HTTP-запит, що містить список пристроїв та їх MAC-адреси.

5. Інтерфейс відображає отриманий список пристроїв (або помилки).

6. Користувач вибирає потрібний пристрій.

7. Надішліть запит HTTP з інтерфейсу користувача, щоб встановити з’єднання з вибраним пристроєм. У тілі запиту надішліть адресу пристрою та порожній пароль.

8. Коли сервер отримує запит, він надсилає повідомлення відправнику через протокол WebSocket і чекає відповіді протягом зазначеного часу (якщо відповіді немає, він відповідає на HTTP-запит, що час очікування минув).

9. Коли пристрій отримує повідомлення з адресою та паролем, передавач намагається встановити з’єднання та повертає результат операції.

10. Після отримання результатів сканування сервер повертає відповідь на HTTP-запит для виконання операції

11. Інтерфейс відображає повідомлення про помилку, повідомлення про успішне підключення або форму для введення пароля. У випадку третього варіанту повторіть операцію ще раз, починаючи з операції номер 7.

3.1.3 Зчитування даних з бортового комп'ютера

При спробі встановити з'єднання пристрій може запитати пароль. Це також регулюється.

Як було сказано вище, зчитування даних буде відбуватися через роз'єм OBD2 в автомобілі. OBD2 (On-Board Diagnostics II) — стандартна діагностична система, яка встановлюється на більшість автомобілів, випущених після 1996 року. Система допомагає діагностувати та контролювати різні параметри автомобіля, такі як продуктивність двигуна, вихлоп, трансмісія та інші компоненти.

Якщо в автомобілі є роз'єм, процес підключення діагностичного сканера до системи OBD2 автомобіля виглядає наступним чином:

1. Визначення протоколу з’єднання: система OBD2 підтримує кілька протоколів зв’язку, таких як: ISO 9141-2, ISO 14230 (KWP2000), ISO 15765 (CAN), J1850 PWM і J1850 VPW.

Діагностичний сканер повинен визначити, який протокол використовує автомобіль. Це можна зробити автоматично (у більшості сучасних сканерів) або вручну, вибравши протокол зі списку доступних інтерфейсів.

2. Встановіть з'єднання: після визначення протоколу сканер встановлює з'єднання з автомобілем. Зазвичай це досягається шляхом ініціалізації зв’язку та обміну даними між сканером і автомобілем для перевірки сумісності та налаштування параметрів підключення. Запалювання має бути ввімкнене.

3. Надіслані команди: однією з основних функцій системи OBD2 є набір стандартних команд під назвою PID (ID параметра). Ці команди використовуються для отримання даних про різні параметри транспортного засобу, такі як швидкість, швидкість двигуна, температура різних компонентів тощо.

4. Отримати відповідь на надіслану команду: автомобіль обробляє запит, надісланий діагностичним сканером, і надсилає відповідь у вигляді даних або статусу на основі параметрів запиту.

5. Інтерпретація даних: діагностичний сканер або програмне забезпечення інтерпретує дані, отримані від автомобіля, і перетворює їх у зручний формат даних. Це може включати декодування кодів помилок, перетворення одиниць вимірювання або створення графіків і діаграм для візуалізації даних.

Слід зазначити, що основні дані, які може отримати бортовий комп’ютер автомобіля, можна отримати тільки при заведеному двигуні.

3.2 Формат даних і частота передачі

Усі дані від відправника будуть передані у форматі JSON. Для гнучкості цієї системи необхідно правильно відокремити дані від набору показників.

Тому розглянемо формат даних для передачі інформації про пристрій.

У цій системі в якості основи буде використовуватися наступна структура даних:

> Об'єкт індикатора однієї поїздки

> Об'єкти для наявних діагностичних помилок

> Об'єкт конфігурації передавача

> Об'єкт властивостей передавача

> Об'єкт поточного стану передавача

3.2.1 Огляд структури даних індикатора

Один об’єкт індикатора поїздки виглядає так:

```
export interface TripMetricModel {
  readonly id: number;
  readonly tripId: number;
  readonly timestamp: Date;
  processed: boolean;
  status: tripMetricStatus;

  mileage: number | null;
  coolantTemp: number | null;

  fuelLevelMainTank: number | null;
  fuelLevelSecondaryTank: number | null;
  batteryLevel: number | null;

  averageEngineLoadPerInterval: number | null;
  averageSpeedPerInterval: number | null;
  averageRPMPerInterval: number | null;
  averageLongFuelTrimPerInterval: number | null;

  readonly updatedAt: Date;
  readonly createdAt: Date;
}
```

Рисунок 3.1 – Об’єкт індикатора одиночної поїздки

У цьому об'єкті поле пробігу буде використовуватися в інтерфейсі для відображення загальної відстані, пройденої автомобілем.

Основні поля, які використовуватимуться для формування сповіщень та аналізу в майбутньому, це поля, які визначають середнє навантаження на двигун (averageEngineLoadPerInterval), середню швидкість (averageSpeedPerInterval), середні оберти на хвилину (averageRPMPerInterval) і середню довгострокову корекцію палива (averageLongFuelTrimPerInterval) для стовпці, передані між надсиланням метрик Вихідне значення для кожного інтервалу часу. Цей інтервал налаштовується в інтерфейсі кожного автомобіля та за замовчуванням становить 5 секунд. Тут важливо

усереднювати за короткий проміжок часу, щоб уникнути великої кількості пакетів, але в той же час зберегти точність отриманих значень.

Також варто звернути увагу на такі значення, як averageLongFuelTrimPerInterval. Цей параметр у системі керування двигуном автомобіля відображає процентну зміну кількості палива, що впорскується в камеру згоряння протягом тривалого періоду часу. Ця корекція забезпечує оптимальне співвідношення палива й повітря для згоряння, тим самим впливаючи на ефективність двигуна, викиди вихлопних газів і економію палива. Якщо система виявляє, що суміш надто насичена (забагато палива) або надто бідна (забагато повітря), вона регулює кількість палива, що надходить у двигун. На основі цих значень можна проаналізувати якість палива, що використовується автомобілем. Це корисно як для власників автопарків, так і для особистого використання.

3.2.2 Огляд структури даних діагностичних помилок

Коди проблем (DTC) використовуються для виявлення проблем в електронних системах автомобіля. Коли система автомобіля виявляє проблему або несправність, відповідний код DTC зберігається в пам’яті автомобіля. Це дозволяє читати та аналізувати ці коди та передавати їх на сервер.

Ці коди можуть вказувати на різні проблеми, від незначних до серйозних. Окрім самого коду, система OBD2 також зберігає іншу корисну інформацію, що передається разом із кодом.

Об'єктами таких помилок, що виявляються системою OBD2, є:

```
export interface DTCModel {
  id: DTCId;
  carId: CarId;
  code: DTCCode;
  description: DTCDescription;
  status: DTCStatus;
  appearedAt: DTCAppearedAt;
  disappearedAt: DTCDisappearedAt;
  mileageAppeared: DTCMileageAppeared;
  mileageDisappeared: DTCMileageDisappeared;
  createdAt: Date;
  updatedAt: Date;
}
```

Рисунок 3.2 – Діагностичний об’єкт помилки

Коди DTC мають стандартну п'ятизначну структуру, що складається з літери та чотирьох цифр. Літера вказує на тип системи, у якій виникла проблема (наприклад, P-двигун, B-кузов, C-шасі, U-мережа), а чотири цифри представляють конкретний код несправності. За допомогою цих кодів ви можете визначити потенційну причину проблеми та керувати процесом ремонту.

Використання кодів DTC спрощує процес діагностики автомобіля, дозволяє швидко виявити причину несправності та забезпечити більш ефективний ремонт.

Крім того, цей формат став загальноприйнятим, що в майбутньому дозволить інтегрувати розроблені системи з сервісами, які забезпечуватимуть можливі рішення проблем лише за допомогою кодів DTC.

3.2.3 Огляд структури даних налаштування передавача

Об'єкт налаштувань пристрою виглядає так:

```
carId: number | null;
thresholds: {
  speed: number | null;
  rpm: number | null;
  coolantTemp: number | null;
}
settings: {
  adapterProtocol: string | null;
  tripMetricsSendingInterval: number,
}
```

Рисунок 3.3 – Діагностичний об’єкт помилки

При підключенні трансмітера до обраного автомобіля, налаштування передаються на пристрій по протоколу WebSocket. Ліміти (пороги) можуть бути використані в майбутньому для алгоритмів, наприклад, для виявлення агресивного стилю водіння автомобіля.

3.2.4 Огляд структури даних поточного стану передавача

Для системи важливою є структура даних, яка зберігає поточний стан передавача. Це включає ідентифікаційний номер автомобіля в системі (до якого підключено передавач), дату останнього підключення передавача до діагностичного сканера, поточну версію операційної системи та програмних додатків, а також стан пам’яті передавача.

Такий об'єкт буде виглядати так:

```
export interface TransmitterStateInfoModel {
  carId: CarId;
  phoneNumber: PhoneNumber;
  connectedBluetoothDevice: BluetoothDeviceModel | null;

  connectedToScanner: ConnectedToScanner;
  lastConnectedToScannerAt: LastConnectedToScannerAt;

  connectedToECU: ConnectedToECU;
  lastConnectedToECUAt: LastConnectedToECUAt;

  activeOBDProtocol: ActiveOBDProtocol;
  appVersion: AppVersion;
  appLastUpdatedAt: LastUpdateAt;
  osVersion: OsVersion;
  osRelease: OsRelease;
  usedStorageCapacityInBytes: UsedStorageCapacityInBytes;
  settings: CarSettingsModel;
  thresholds: CarThresholdsModel;
}
```

Рисунок 3.4 – Об’єкт поточного стану пристрою

Більшість із цих полів відображаються в інтерфейсі користувача та мають вирішальне значення для розуміння того, чи здатний пристрій підключитися до бортового комп’ютера автомобіля. Крім того, цей об'єкт містить поточні налаштування пристрою, який має поточний протокол підключення OBD2 - ця інформація буде дуже корисна при пошуку причини можливих проблем з підключенням автомобіля.

Також в об’єкті є номер мобільного телефону, який зчитується з сім-карти: використовується для оплати місячного тарифного плану.

3.2.5 Огляд структури даних характеристик передавача

У цій системі є об’єкт характеристик пристрою, який містить поля, що описують фізичні характеристики пристрою, такі як: обсяг оперативної

пам’яті, загальний обсяг пам’яті, тип платформи тощо. Такий об'єкт буде виглядати так:

```
export interface TransmitterSpecsModel {
  hardwarePlatform: HardwarePlatform;
  osPlatform: OsPlatform;
  ramCapacityInBytes: RamCapacityInBytes,
  storageCapacityInBytes: StorageCapacityInBytes
  cpuModel: CpuModel;
}
```

Рисунок 3.5 – Об’єкт властивостей передавача

Цей об'єкт передає інформацію про характеристики випромінювача. Щоб зменшити розмір передавача, його видаляють із поточного стану передавача. Надсилається один раз, коли пристрій підключено до сервера, і містить назву платформи відправника даних (у цьому випадку Raspberry), назву операційної системи, обсяг оперативної та загальної пам’яті, а також назву пристрою. процесор.

3.3 Аналіз даних на сервері API

Щоб представити дані користувачам у зрозумілому форматі, дані потрібно обробити. По суті, сервер отримуватиме показники та діагностичні помилки для кожної поїздки. Як вже здогадалися вище, при працюючому двигуні індикатор буде з'являтися кожні 5 секунд, а помилка діагностики - при кожному запуску двигуна.

Для індикаторів аналіз наприкінці «подорожі» має вирішальне значення. Такт — це умовна назва робочого діапазону двигуна від моменту його запуску до моменту його зупинки.

Зберігаючи помилки діагностики, важливо не створювати копію тієї самої помилки щоразу, коли виникає та сама помилка. Для цього необхідно розробити спеціальні алгоритми.

3.3.1 Обробка показників маршруту

Цей індикатор створюється, коли надходить індикатор з унікальним ідентифікатором поїздки, який ще не існує в системі. Унікальний ідентифікатор генерується самим емітером. Крім того, кожен пакет має статус поточної поїздки, і значення цього статусу може бути:

> Почати

> йти

> Повне

Початок стану відбувається, коли надсилається перша метрика, і відбувається для кожного наступного пакета, доки механізм не зупиниться. Коли двигун зупиняється, зібрані дані надсилаються на сервер із значеннями статусу, включеними в пакет завершення.

При отриманні посилки зі статусом «Виконано» починається обробка всієї поїздки. Він включає розрахунок пройденої відстані, кількості спожитого палива, середньої витрати палива та аналіз отриманих показників, що перевищують встановлені ліміти.

3.3.2 Обробка діагностичних помилок

Робота системи діагностики автомобіля OBD2 має свої особливості. Перевіряйте всі системи перед запуском двигуна та під час роботи. Тим не менш, на цих етапах роботи автомобіля можуть виникати помилки. Також можливо, що проблема, яка викликала помилку, може зникнути (наприклад, після ремонту деталі автомобіля або після того, як значення деяких датчиків прийдуть в норму) - в цьому випадку помилки не зникнуть повністю, а лише зміняться. їх ціннісний статус. Крім того, слід враховувати, що діагностичні

помилки можна видалити вручну - в цьому випадку вони повністю зникнуть з пам'яті бортового комп'ютера. Система була розроблена з урахуванням усіх цих особливостей.

Всього діагностична помилка може мати 3 стани:

• Дійсний (очікує на розгляд)

• Збережено

• Видалено (історія)

Щоразу, коли двигун запускається, передавач зчитує наявні несправності та порівнює їх з несправностями, збереженими в попередніх скануваннях. Коли виявляється зміна, пристрій надсилає HTTP-запит або створює нову помилку (знаходиться в виникає нова помилка) або оновлює статус існуючої помилки (якщо вона зникає або змінює статус).

Саме така реалізація дозволяє підтримувати всю історію помилок і їх поточний статус. Крім того, цей децентралізований аналіз помилок значно зменшує потенційне навантаження на сервер, оскільки якби передавачі надсилали діагностичні помилки кожного разу, коли запускали двигун, необхідно було б завжди отримувати всі доступні помилки для кожного передавача та виконувати подібний аналіз до того, що відбувається на передавач у цьому прикладі. Коли кількість автомобілів, підключених до системи, збільшується, може статися збій сервера через високе навантаження.

Також слід зазначити, що реалізовано спеціальний механізм для моніторингу успішності передачі помилки на сервер: у разі успішної передачі, якщо інформація про помилку більше не присутня в пам’яті, відправник видаляє відповідну помилку з інформація внутрішньої пам'яті. У пам'яті бортового комп'ютера, якщо виникне помилка при відправленні або немає мережі, передавач збереже помилку в списку помилок для подальшої передачі на сервер. Ця реалізація зменшує навантаження на сервер і забезпечує узгодженість даних.

3.3.3 Формування статистики подорожей

Як зазначалося на початку розділу аналізу даних, після отримання всіх показників для кожної поїздки проаналізуйте показники. Далі запис поїздки в базі даних оновлюється шляхом додавання значень, розрахованих під час аналізу.

Таким чином, база даних буде зберігати необхідне значення, яке було розраховано:

> Середнє споживання

> Загальне споживання

> Відстань їзди

Отже, щоб отримати статистику, ви просто вибираєте поїздку протягом потрібного періоду часу, виконуєте мінімальні розрахунки за допомогою бази даних і надаєте результати користувачеві.

3.4 Особливості роботи програмного забезпечення передавача

Програмне забезпечення передавача реалізовано на мові програмування Python. В основному цей вибір залежить від наявності різних бібліотек, розроблених для цієї мови програмування, а саме бібліотеки, яка інкапсулює роботу протоколу OBD-2 - Python-OBD.

3.4.1 Використання бібліотеки Python-OBD

Python-OBD — це бібліотека для мови програмування Python, яка забезпечує функціональні можливості для читання діагностичних даних автомобіля за допомогою протоколу OBD-II (бортова діагностика). За допомогою цієї бібліотеки ви можете підключитися до свого автомобіля за допомогою адаптера OBD-II, підключеного до діагностичного порту вашого автомобіля. Бібліотечні функції потім можна використовувати для читання різних параметрів автомобіля, таких як швидкість, оберти двигуна, температура охолоджуючої рідини тощо.

До переваг цієї бібліотеки можна віднести простоту використання: Python-OBD забезпечує простий і зрозумілий інтерфейс для читання даних автомобіля. Досить кількох рядків коду, щоб підключитися до автомобіля та отримати дані. Він також підтримує кілька протоколів OBD2: Python-OBD підтримує різні протоколи, включаючи ISO 9141-2, ISO 14230-4 (KWP2000), ISO 15765-4 (CAN), SAE J1850 PWM і SAE J1850 VPW. Це означає, що бібліотека працюватиме з більшістю автомобілів, які використовують OBD-II. Однак головною перевагою цієї бібліотеки є те, що вона підтримує асинхронне читання.

Асинхронність є важливою концепцією в програмуванні, особливо в сучасних програмах, які мають справу з мережевою взаємодією, введенням/виведенням (I/O) і багатозадачністю. Асинхронність дуже важлива для розробки систем. Наприклад, під час сканування найближчих пристроїв Bluetooth передавач також повинен повідомити сервер про те, що сканування триває, що неможливо з неблокуючим вводом-виводом.

3.4.2 Структура програмного забезпечення передавача

Проект складається з двох основних сервісних компонентів і компонента веб-сокета. Послуги включають:

1. bluetooth_service.py – Сервіс для пристроїв Bluetooth.

2. file_system_service.py – Служби для використання файлової системи платформи.

3.network_service.py – Служба для використання з мережею запуску.

4. obd_service.py – Сервіс для взаємодії з системою OBD2 автомобіля.

Структура проекту показана на рисунку. 3.6:

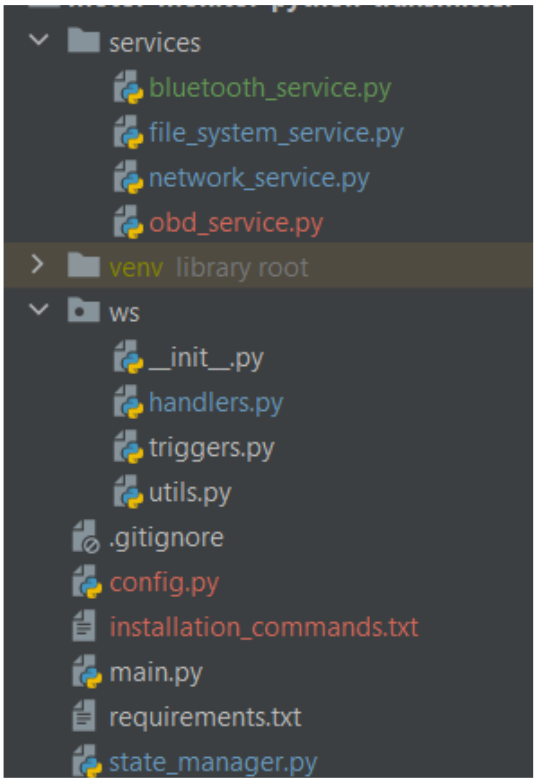


Рисунок 3.6 – Структура програмного забезпечення передавача

У проекті також є файл, який містить команди, необхідні для встановлення додаткових інструментів і компонентів, необхідних для основних бібліотек, таких як Python-OBD і PyBluez.

Результатом цієї роботи є розробка системи з використанням методів і компонентів, описаних у розділі 2.

Зверніть особливу увагу на підключення трансмітера до діагностичного сканера та сервера. Враховуючи специфіку умов його експлуатації (постійна зміна положення), були розроблені спеціальні алгоритми для встановлення зв'язку з діагностичним сканером і подальшого дистанційного налаштування самого передавача. Крім того, зміна місця розташування впливає на вибір компонентів, що відповідають за мобільний Інтернет під час руху автомобіля: USB-3G-модем буде використовуватися для забезпечення передавача даних Інтернетом. Основні причини такого рішення:

- Нижча вартість порівняно з платою розширення Raspberry pi
- Немає необхідності писати окремі модулі

Налаштування плати розширення

Детально розглянуто алгоритм пошуку і підключення до Bluetooth-сканерів, докладно описаний кожен крок. Також буде представлено фрагмент коду, що демонструє алгоритм підключення Bluetooth-сканера

Крім того, наведено основні структури даних розробленої системи, а також продемонстровано їх призначення та структуру.

4 ТЕСУВАННЯ ТА АНАЛІЗ СТВОРЕНОЇ СИСТЕМИ

4.1 Розрахунок вартості системної інтеграції

Для інтеграції розробленої системи в автомобіль необхідні компоненти наведені в таблиці 4.1:

Таблиця 4.1 – Розрахунок вартості компонентів для одиниці обладнання

Назва	**Вартість**
Діагностичний сканер ELM 327	300 грн
Одноплатний коп'ютер Raspberry Pi 3 Model A+	2 800 грн
3G GSM модем Alcatel 1K3M	460 грн
	Загальна вартість: 3560 грн

Отже, можна зробити висновок, що ціна інтеграції автомобільної системи може бути знижена майже в 9 разів у порівнянні з конкурентом цієї системи «IOSiX OBD-II Data Logger».

Також слід зазначити, що сума інтеграції досить велика, і сюди не входить місячна вартість користування мобільним інтернетом. Однак, як було сказано вище, вибір компонентів залежить не тільки від цінових критеріїв, а й від складності написання програмного коду для них, щоб у майбутньому деякі компоненти можна було замінити на більш дешеві аналоги.

4.2 Огляд інтерфейсу системи розробки

Інтерфейсу розробленої системи приділено багато уваги, оскільки це частина системи, з якою в основному взаємодіють користувачі, тому вона була розроблена з урахуванням початкової вимоги: «Інтерфейс системи повинен бути простим та інтуїтивно зрозумілим для користувач».

Скріншот початкової сторінки наведено нижче (рис. 4.1). Відображає список зареєстрованих в системі автомобілів і основні відомості про них. Ліворуч також є меню з вкладками, які можна переглянути:

• Список зареєстрованих автомобілів

• Інформаційні панелі для всіх автомобілів (майбутня функція)

• Список сповіщень для всіх автомобілів (майбутня функція)

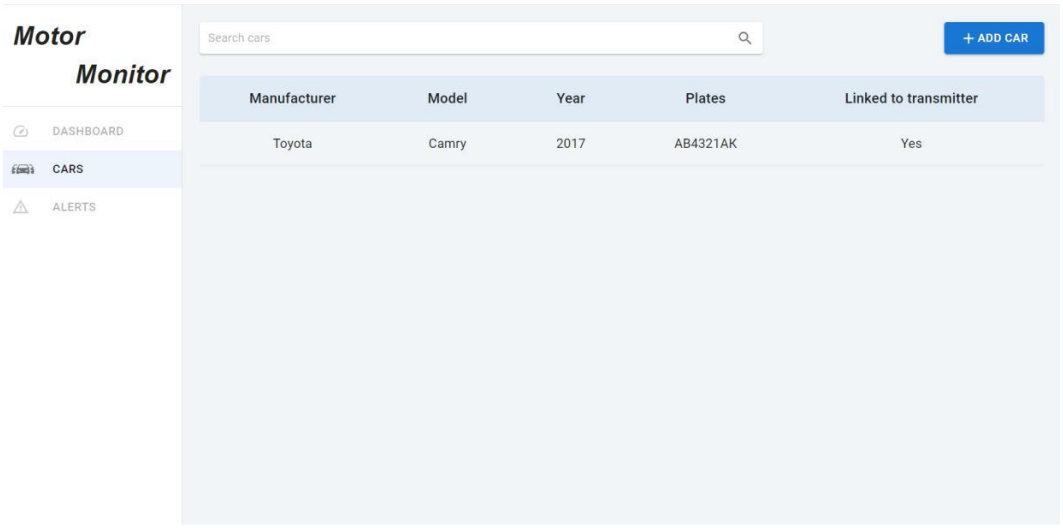


Рисунок 4.1 – Початкова сторінка інтерфейсу

На цій сторінці також можна додавати нові автомобілі (Рисунок 4.2):

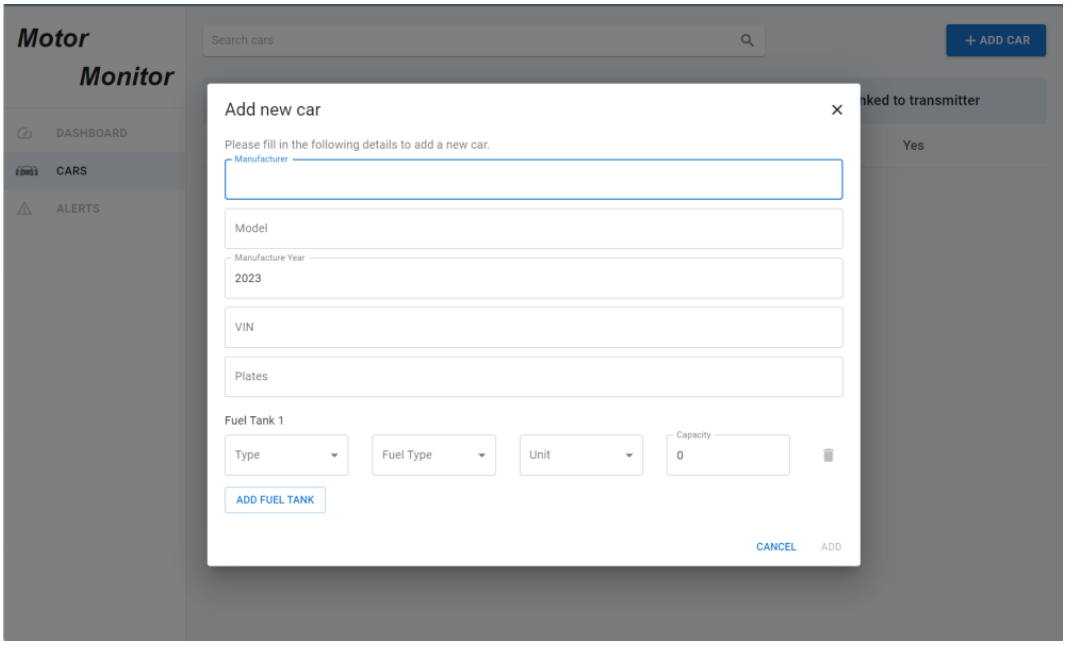


Рисунок 4.2 – Створення реєстраційної форми автомобіля

При натисканні на елемент у списку зареєстрованих автомобілів відкривається детальна інформація про вибраний автомобіль (рис. 4.3).

На цій сторінці ви можете змінити обмеження транспортного засобу (Рисунок 4.4), підключити/відключити діагностичний сканер до передавача або від’єднати автомобіль від передавача.

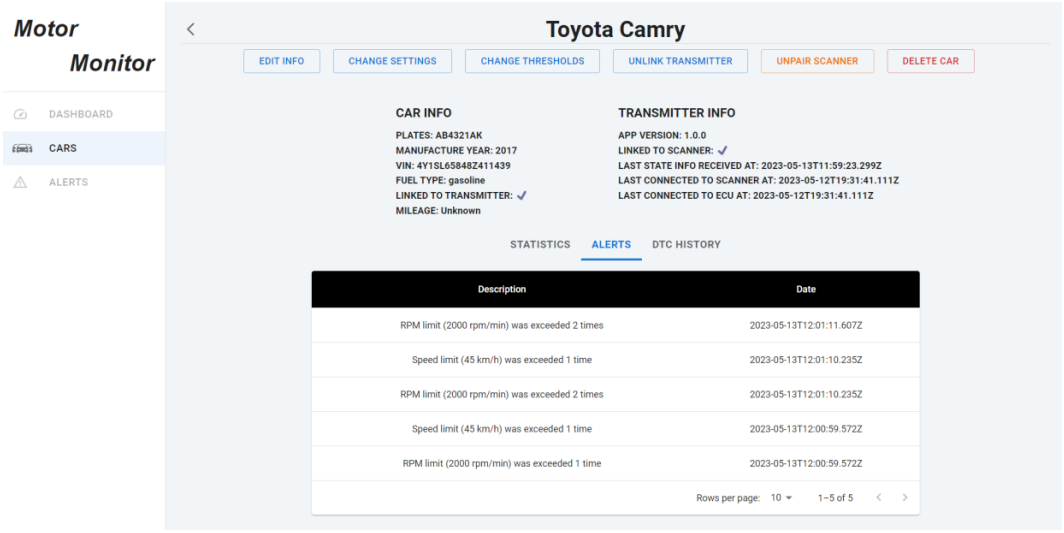


Рисунок 4.3 – Сторінка відомостей про транспортний засіб

Наприклад, нижче наведено вікно для зміни лімітів автомобіля, яке буде використано надалі при аналізі вхідних показників:

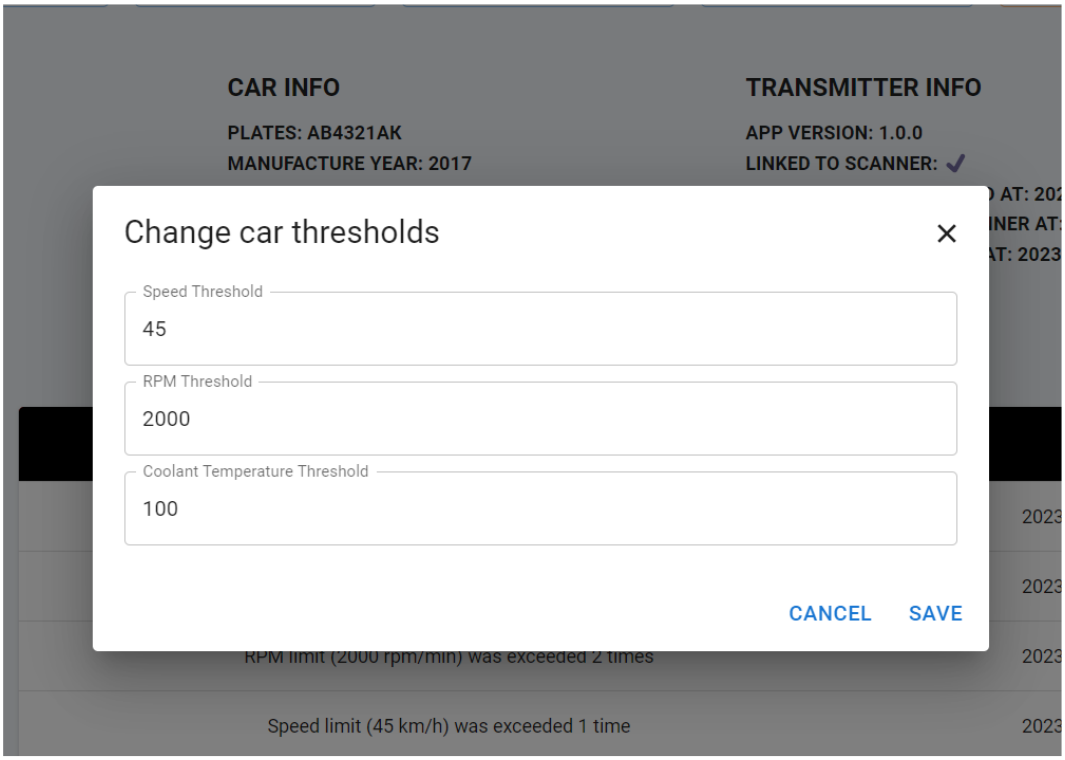


Рисунок 4.4 – Таблиця зміни порогів автомобіля

Крім того, на сторінці детальної інформації (див. Рисунок 4.3) ви можете переглянути наявні сповіщення про перевищення ліміту та статистику завершених поїздок (Рисунок 4.5).

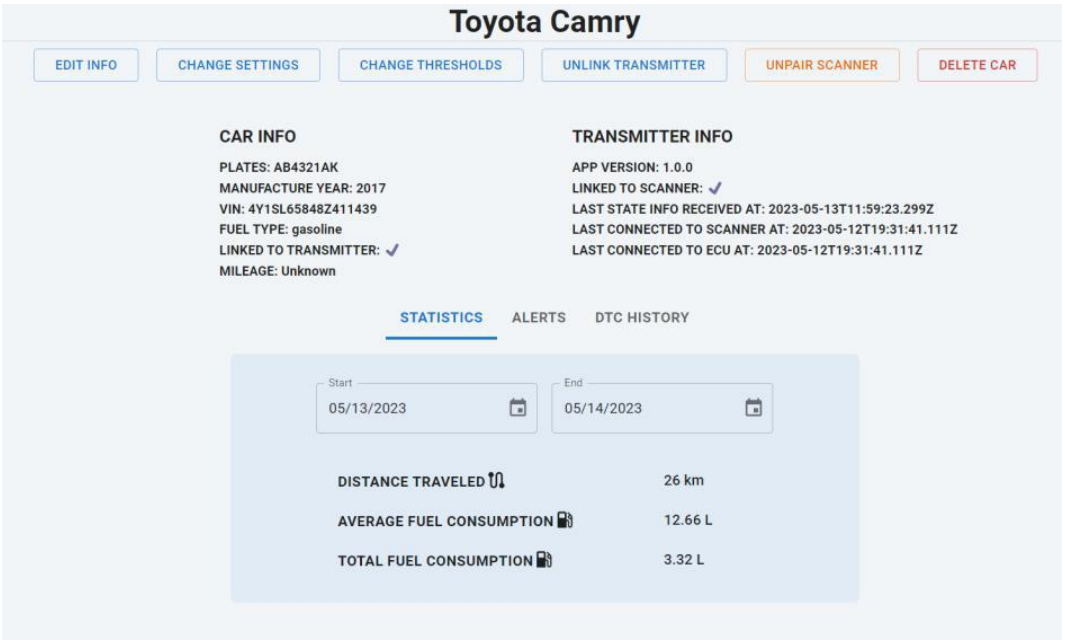


Рисунок 4.5 – Вкладка «Статистика» здійснених поїздок

Історію діагностичних помилок можна переглянути на третій вкладці (рис. 4.6):

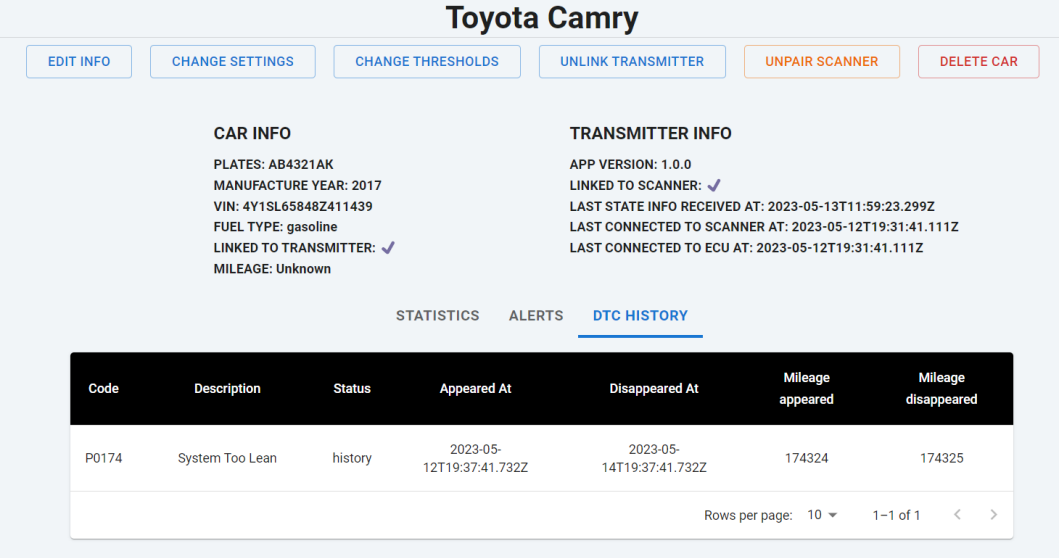


Рисунок 4.6 – Вкладка журналу діагностичних помилок

Кожна вкладка, показана на зображенні вище, представлена окремим компонентом, який має кнопки для навігації даними для зручного перегляду.

4.3 Дослідити правильність створення повідомлення

Основна мета цієї системи - краще контролювати автомобіль і спосіб його руху, тому для цього був розроблений спеціальний механізм, який сповіщає власника про те, що певні раніше встановлені значення перевищують ліміти. Дуже важливо, щоб ця функція працювала належним чином. Тому було проведено наступне дослідження: встановлення максимальної швидкості 50 км/год, а потім коротка поїздка, під час якої значення швидкості 50 км/год було перевищено двічі. Усі наявні сповіщення до цього часу видалено. вийти

Під час експерименту в інтерфейсі користувача з’явилося повідомлення про перевищення швидкості (рис. 4.7):

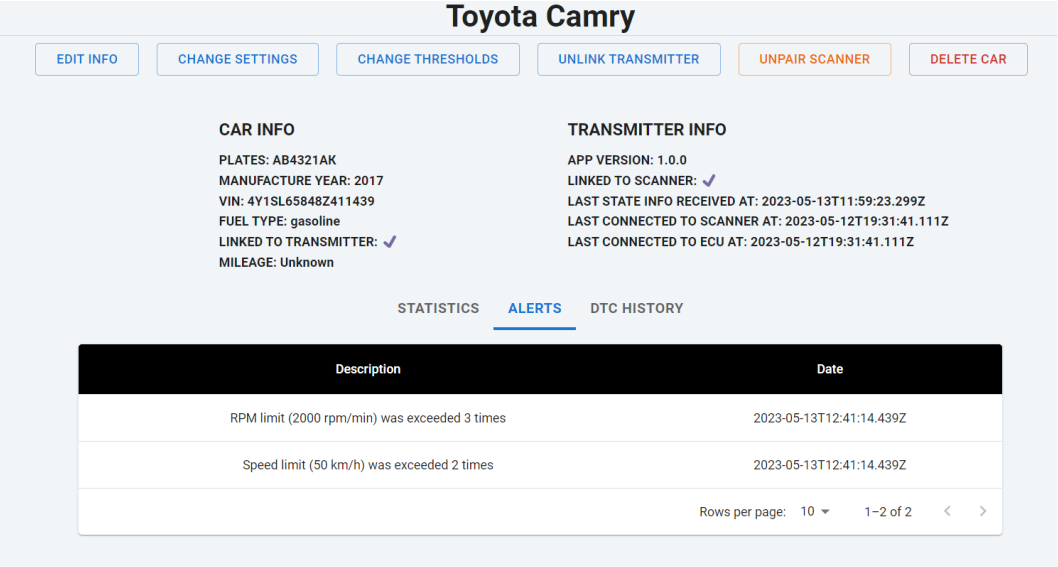


Рисунок 4.7 – Вкладка з результатами експерименту

Загалом інтерфейс користувача інтуїтивно зрозумілий і повністю функціональний, тому можна вважати, що початкові вимоги виконано.

4.4 Пропозиції щодо розробки та вдосконалення програми

Розроблена система може бути суттєво вдосконалена як у програмному напрямку, так і в напрямку вибору системного обладнання (компонентів).

Що стосується програмного забезпечення, то систему можна покращити шляхом впровадження нових функцій у передавач, таких як GPS-відстеження транспортних засобів і дистанційне оновлення мікропрограми.

Можливість відстежувати автомобіль за допомогою системи GPS особливо корисна для власників автопарків: якщо автомобіль викрадено, система відстеження GPS може допомогти поліції швидко знайти його. Це також корисно для логістичних компаній, оскільки відстеження транспортних засобів може допомогти оптимізувати маршрути, щоб заощадити паливо та час. Крім того, поява такого функціоналу в майбутньому дозволить створювати функції, які виявляють ДТП і негайно надсилають координати місця розташування рятувальним службам у разі аварії.

Що стосується функції дистанційного оновлення ПЗ на трансмітері, то ця функція допоможе легко оновити програмну частину самого трансмітера без ручного втручання, що полегшить процес інтеграції нових функцій і виправлення можливих помилок.

Вартість інтеграції цієї системи в автомобіль також можна істотно знизити, якщо замінити один із найдорожчих компонентів – одноплатний комп'ютер Raspberry Pi 3 Model A+ – на більш простий ПК Orange PI або навіть перейти на платформу Arduino.

Розділ 4 розраховує кінцеві витрати на інтеграцію розробленої системи в автомобіль і детально розглядає інтерфейси розробленої системи.

Крім того, робота основних функцій підтверджена експериментами з тест-драйвом автомобіля. Випробування показали, що система працювала належним чином: перед тест-драйвом було встановлено максимальне обмеження швидкості 50 км/год і швидкість двигуна 2000 об/хв, а потім було проведено тест-драйв на автомобілі з інтегрованою системою після перевищення При перевищенні максимальної швидкості в 3 рази та перевищенні максимальної швидкості в 2 рази, відповідні сповіщення генеруються в інтерфейсі користувача.

Також аналізуються можливі шляхи розвитку та можливі вдосконалення системи. Загалом було розглянуто два напрямки для покращення. Під час аналізу майбутніх удосконалень ми вирішили, що найбільш доцільним наступним кроком буде додавання в систему GPS-модуля, який значно розширить функціональність системи.

ВИСНОВКИ

У цій роботі створено систему для збору й автоматичного аналізу даних, зібраних із систем OBD2 автомобіля. Система надає власникам можливість централізовано переглядати статистику поїздок, сповіщення про перевищення встановлених значень і історію діагностичних помилок з системи OBD2 автомобіля.

Основною метою цієї роботи є зниження витрат і спрощення інтеграції таких систем в автомобілі порівняно з конкуруючими системами на ринку, про що йшлося в розділі 1. Враховуючи, що компоненти системи можна розмістити в будь-якому місці автомобіля (крім діагностичного сканера), можна зробити висновок, що мета спрощення процесу інтеграції досягнута. Що стосується зниження собівартості, то, з одного боку, розроблена система значно дешевша порівняно з конкурентами, але її вартість все ще висока, тому можна зробити висновок, що ця мета частково досягнута.

Завдяки правильному вибору компонентів і алгоритмів під час роботи над другою частиною цієї роботи можна досягти надійної та високопродуктивної системи з правильною архітектурою, що дозволяє легко змінювати. Крім того, структури даних і методи обробки даних, розглянуті в частині III, допомагають полегшити підтримку коду, який ви пишете, спрощують пошук помилок і зменшують навантаження на мережу, що важливо для таких систем. Також розроблено простий та інформативний інтерфейс користувача, який представлений у розділі 4 цієї роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OBD [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/OBD.

2. Офіційна сторінка сервісу «Zubie» [Електронний ресурс] – Режим доступу до ресурсу: https://zubie.com

3. Офіційна сторінка сервісу «Geotab» [Електронний ресурс] – Режим доступу до ресурсу: https://www.geotab.com/vehicle-telematics/

4. Офіційна сторінка сервісу «IOSi» [Електронний ресурс] – Режим доступу до ресурсу: https://iosix.com/product/obd-ii-data-logger/

5. M. R. Faezipour, M. Nourani and A. Saeed, "Towards an Integrated OBD-II and CAN Network Vehicle Diagnostic System," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 2, pp. 533-545, Feb. 2016. https://ieeexplore.ieee.org/document/7312460

6. Node.js Architecture From A to Z [Електронний ресурс] – Режим доступу до ресурсу: https://litslink.com/blog/node-js-architecture-from-a-to-z

7. A. Emad, M. A. Hannan, A. Hussain, S. A. Samad and A. Mutalib, "Automotive On-Board Diagnostics (OBD-II) Interface and System for Vehicle Condition Monitoring," in *IEEE* Transactions on Instrumentation and Measurement, vol. 68, no. 11, pp. 4475-4485, Nov. 2019. https://ieeexplore.ieee.org/document/8675982

8. Fastify Benchmarks [Електронний ресурс] – Режим доступу до ресурсу: https://www.fastify.io/benchmarks/

9. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PostgreSQL

10. The Best Guide to Know What Is React [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

11. Material UI – Overview [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/material-ui/getting-started/overview/

12. Офіційна сторінка продукту Orange Pi PC [Електронний ресурс] – Режим доступу до ресурсу: https://orangepi.com/index.php?route=product/product&product_id=864

13. S. P. Singh, A. K. Jain and M. Agrawal, "Development of an OBD-II Based Vehicle Diagnostic System," in Proceedings of the 2018 IEEE International Conference on Vehicular Electronics and Safety *(ICVES)*, Madrid, Spain, 2018, pp. 120-125. https://ieeexplore.ieee.org/document/8500524

14. Стасюк Я.В., Бобало Ю.І., Гаврилюк В.П., Куц І.В. та ін. (2019) "Розробка системи моніторингу автомобільних датчиків на базі OBD-II" в Технічні науки: Міжвідомчий науково-технічний збірник вип. 4(110), с. 11-17. Видавець: Національний університет "Львівська політехніка". Доступно: Національний університет "Львівська політехніка"

14. Офіційна сторінка продукту Raspberry Pi 3 Model A+ [Електронний ресурс] – https://www.raspberrypi.com/products/raspberry-pi-3-model-a-plus/

15. Сторінка продажу ELM 327 в інтернет-магазині Rozetka [Електронний ресурс] – Режим доступу до ресурсу: https://rozetka.com.ua/ua/218637907/p218637907/

16. How Does Elm327 Work? [Електронний ресурс] – Режим доступу до ресурсу: https://www.sdd-technology.com/blog/how-does-elm327-work

17. Y. Zhu, L. Wang, D. Liu and X. Zhang, "Design of a Real-Time Vehicle Monitoring System Based on OBD-II and CAN Bus," in *Proceedings of the 2016* IEEE International Conference on Vehicular Electronics and Safety (ICVES), Beijing, China, 2016, pp. 387-392.: https://ieeexplore.ieee.org/document/7548276

18. Калитін М.О., Томашевський Я.О., Слободянюк Р.Є. та ін. (2017) "Мобільна система моніторингу стану автомобіля на базі інтерфейсу OBD-II" в Автоматика та приладобудування, вип. 1(50), с. 51-60. Видавець: Хмельницький національний університет.

Додаток А
(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: « Автоматизована система контролю стану автомобілю через OBD2 інтерфейс»

Тип роботи: Бакалаврська дипломна робота
(БДР, МКР)

Підрозділ КСУ, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 89,2% Схожість 10,8%

Аналіз звіту подібності (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Володимир ДУБОВОЙ
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek щодо роботи.

Автор роботи _____ Вадим КРУЧКО
(підпис) (прізвище, ініціали)

Керівник роботи _____ Олена НИКИТЕНКО
(підпис) (прізвище, ініціали)

ДодатокБ
(обов'язковий)

ВНТУ

ЗАТВЕРДЖЕНО
Завідувач кафедри КСУ
д.т.н.,проф.
_____ В'ячеслав КОВТУН

«_24_» травня 2024р.

ТЕХНІЧНЕЗАВДАННЯ

на виконання бакалаврської дипломної роботи

Автоматизована система контролю стану автомобіля через OBD2 інтерфейс

Студент групи АКІТ-22мс
_____Вадим КРУЧКО

Керівник: к.т.н., доц. каф. КСУ
_____Олена НИКИТЕНКО

Вінниця 2024

1. Назва та галузь застосування

1.1. Назва – Розробка ефективного класифікатора для автоматизованої підсистеми допуску персоналу на виробництві.
1.2. Галузь застосування – Автоматизація та комп'ютерно-інтегровані технології.

2. Підстава для проведення розробки.
Тема бакалаврської дипломної роботи затверджена наказом по ВНТУ № 80 від 11 березня 2024 р.

3. Мета та призначення розробки.
Метою бакалаврської дипломної роботи є підвищення ефективності систем прийняття рішень за рахунок розробки покращених методів кластеризації і класифікації.

4. Джерела розробки.
Бакалаврська дипломна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:
1. OBD [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/OBD.
2. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PostgreSQL
3. Node.js Architecture From A to Z [Електронний ресурс] – Режим доступу до ресурсу: https://litslink.com/blog/node-js-architecture-from-a- to-z

5. Вимоги до розробки.
5.1. Перелік головних функцій:
- Зчитування даних з OBD2;
- Діагностика помилок;
- Відображення інформації на інтерфейсі користувача;
- Збереження історії;

5.2. Основні технічні вимоги до розробки.
5.2.1. Вимоги до програмної платформи:
- Python, C ++ або Java
- Windows, Linux або MacOS.

5.2.2. Умови експлуатації системи:
- Підтримка різних типів автомобілів
- Стійкість до змін
- Забезпечення безпеки

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

- Аналіз методів, принципів, підходів і засобів для автоматизації процесів контролю стану автомобіля — 13.05.2024 р.
- Визначення технічних характеристик програмно-апаратної системи для контролю стану автомобіля — 17.05.2024 р.
- Розробка програмного забезпечення системи — 27.05.2024 р.
- Тестування та верифікація програмного забезпечення — 04.06.2024 р
- Оформлення пояснювальної записки — 08.06.2024 р.

6.2 Графічні матеріали:

- Розробка діаграм — 29.05.2024 р.
- Створення порівняльних таблиць — 30.05.2024 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «08» червня 2024 р.

7.2. Атестація роботи здійснюється на попередньому захисті. Попередній захист бакалаврської дипломної роботи провести до «12» червня 2024 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК.

7.4. Захист бакалаврської дипломної роботи провести до «19» червня 2024р.

Додаток В
(довідковий)
Лістинги

Текст програмного коду модуля для обробки метрик

```
import Errors from '../../../errors';
import { InfoMessage } from '../../common/types';
import { CarId } from '../../cars/service/cars.types';
import { START, FINISH } from '../constants/trips.metrics.statuses';
import Decimal from 'decimal.js';
import tripsRepository, { TripsRepository } from './trips.repository';
import carsService, { CarsService } from '../../cars/service';
import alertsService, { AlertsService } from '../../alerts/service';
import {
CalculateAverageFuelConsumptionFuncParams,
CalculateTotalFuelConsumptionFuncParams,
CreateTripFuncParams,
CreationTripMetricFields,
TripMetricModel,
tripMetricStatus, TripsStatisticsModel,
} from './trips.types';
export class TripsService {
constructor(private readonly tripsRepository: TripsRepository, private readonly carsService:
CarsService, private readonly alertsService: AlertsService) {
this.tripsRepository = tripsRepository;
this.carsService = carsService;
this.alertsService = alertsService;
}
async saveTripMetric(tripData: CreationTripMetricFields): Promise<InfoMessage> {
const { carId, ...tripMetric } = tripData;
const { tripId, timestamp, status, lat, lng } = tripMetric;
const trip = await this.tripsRepository.getBaseTripInfoById(tripId);
if (!trip) {
await this._createTrip({ tripId, carId, startLat: lat, startLng: lng });
} else {
if (trip.completed) throw new Errors.ConflictError(`Trip with id=${tripId} is considered complete, metrics
are no longer accepted`);
await this._checkMetricAbsenceWithProvidedTimestampForTrip({ tripId, timestamp });
if ([START, FINISH].includes(status)) await
```

```
this._checkTripMetricAbsenceWithProvidedStatusForTrip({ tripId, status });
}
if (status === FINISH) {
await this.tripsRepository.createTripMetricWithMarkingTripAsCompleted(tripMetric);
this._startTripMetricsProcessing({ tripId, carId });
} else {
await this.tripsRepository.createTripMetric(tripMetric);
}
return { message: 'Trip metric was successfully saved' };
}
async getTripsStatisticsForCar(carId: CarId, startDate: Date, endDate: Date):
Promise<TripsStatisticsModel> {
await this.carsService.checkCarExistenceById({ carId });
const statistics = await this.tripsRepository.getTripsStatisticsForCar(carId, startDate, endDate);
if (!statistics) throw new Errors.NotFoundError(`Statistics for car with id=${carId} were not found`);
return statistics;
}
async _createTrip({ tripId, carId, startLat, startLng }: CreateTripFuncParams): Promise<void> {
await this.carsService.checkCarExistenceById({ carId });
await this.tripsRepository.createTrip({ tripId, carId, startLat, startLng });
}
async _checkMetricAbsenceWithProvidedTimestampForTrip({ tripId, timestamp }: { tripId: number,
timestamp: Date }) {
const tripMetric = await this.tripsRepository.getFullMetricInfoByTimestampForTrip({ tripId, timestamp });
if (tripMetric) throw new Errors.ConflictError(`Trip with
timestamp=${timestamp.toISOString()} for trip with id=${tripId} already exists in the system`);
}
async _checkTripMetricAbsenceWithProvidedStatusForTrip({ tripId, status }: { tripId:
number, status: tripMetricStatus }) {
const tripMetric = await this.tripsRepository.getFullTripMetricInfoByTripIdAndStatus({ tripId, status });
if (tripMetric) throw new Errors.ConflictError(`Trip with status='${status}' for trip with id=${tripId} already
exists in the system`);
}
async _startTripMetricsProcessing({ tripId, carId }: { tripId: number, carId: CarId }):
Promise<void> {
try {
const metrics = await this.tripsRepository.getAllTripMetrics(tripId);
const duration = this._calculateTripDuration(metrics);
const distanceTraveled = this._calculateTripDistance(metrics);
```

```
const totalFuelConsumptionMainTank = this._calculateTotalFuelConsumption({ metrics, levelField: 'fuelLevelMainTank' });
const totalFuelConsumptionSecondaryTank = this._calculateTotalFuelConsumption({ metrics, levelField: 'fuelLevelSecondaryTank' });
const totalBatteryConsumption = this._calculateTotalFuelConsumption({ metrics, levelField: 'batteryLevel' });
const averageFuelConsumptionMainTank =
this._calculateAverageFuelConsumption({ distanceTraveled, fuelConsumed: totalFuelConsumptionMainTank });
const averageFuelConsumptionSecondaryTank =
this._calculateAverageFuelConsumption({
distanceTraveled,
fuelConsumed: totalFuelConsumptionSecondaryTank
});
const averageBatteryConsumption = this._calculateAverageFuelConsumption({ distanceTraveled, fuelConsumed: totalBatteryConsumption });
await this.tripsRepository.updateTripInfo({
tripId,
tripDataToUpdate: {
processedAt: new Date(),
duration,
distanceTraveled,
averageFuelConsumptionMainTank,
totalFuelConsumptionMainTank,
averageFuelConsumptionSecondaryTank,
totalFuelConsumptionSecondaryTank,
averageBatteryConsumption,
totalBatteryConsumption,
}
});
await this.alertsService.processMetricsForAlertsCreation({ tripId, carId, metrics });
} catch (error) {
console.log(error);
}
}
_getStartAndFinishMetrics(metrics: Array<TripMetricModel>): Array<TripMetricModel | undefined> {
const startMetric = (metrics[0].status === START) ? metrics[0] : metrics.find((metric) => metric.status === START);
const finishMetric = (metrics[metrics.length - 1].status === FINISH) ?
```

```
        metrics[metrics.length - 1] : metrics.find((metric) => metric.status === FINISH);
        return [startMetric, finishMetric];
        }
        _calculateTripDuration(metrics: Array<TripMetricModel>): number | null {
        const [startMetric, finishMetric] = this._getStartAndFinishMetrics(metrics);
        if (!startMetric || !finishMetric) return null;
        const startMetricTimestamp = new Date(startMetric.timestamp).getTime();
        const finishMetricTimestamp = new Date(finishMetric.timestamp).getTime();
        return parseFloat(new
        Decimal(finishMetricTimestamp).minus(startMetricTimestamp).toFixed(2));
        }
        _calculateTripDistance(metrics: Array<TripMetricModel>): number | null {
        const [startMetric, finishMetric] = this._getStartAndFinishMetrics(metrics);
        if (!startMetric || !finishMetric) return null;
        if ((!startMetric.mileage && startMetric.mileage !== 0) || (!finishMetric.mileage &&
        finishMetric.mileage !== 0)) return null;
        return parseFloat(new
        Decimal(finishMetric.mileage).minus(startMetric.mileage).toFixed(2));
        }
        _calculateTotalFuelConsumption({ metrics, levelField }:
        CalculateTotalFuelConsumptionFuncParams): number | null { const fuelLevels = metrics.map((metric) =>
metric[levelField]);
        if (fuelLevels.every((level) => level === null)) return null;
        return fuelLevels.reduce((accumulator: number, currentValue: number | null, currentIndex: number,
array) => {
        const previousValue = array[currentIndex - 1];
        if (previousValue === undefined || previousValue === null || currentValue === null) return accumulator;
        if (previousValue > currentValue) accumulator += (previousValue - currentValue);
        return accumulator;
        }, 0);
        }
        _calculateAverageFuelConsumption({ distanceTraveled, fuelConsumed }:
        CalculateAverageFuelConsumptionFuncParams): number | null {
        if (distanceTraveled === null || fuelConsumed === null) return null;
        if (!distanceTraveled) return null; // car did not move when engine was on
        return parseFloat(new Decimal(fuelConsumed).div(new
        Decimal(distanceTraveled).div(100)).toFixed(2));
        }
        }
```

```
export default new TripsService(tripsRepository, carsService, alertsService);
```

Текст програмного коду модуля для створення сповіщень

```
import carsService, { CarsService } from '../../cars/service';
import alertsRepository, { AlertsRepository } from './alerts.repository';
import tripsRepository, { TripsRepository } from '../../trips/service/trips.repository';
import postgresClient from '../../../modules/postgresDB';
import { Knex } from 'knex';
import { CreateAlertFuncParams, GetAllAlertsReturnModel,
GetAllAlertsWithPaginationFuncParams } from './alerts.types';
import { CarId } from '../../cars/service/cars.types';
import { TripId, TripMetricModel } from '../../trips/service/trips.types';
const limitationsDescriptionBuilders = {
speedLimitExceeded: ({ speedThreshold, speedLimitExceedsCount }) => {
const isPlural = speedLimitExceedsCount > 1;
return `Speed limit (${speedThreshold} km/h) was exceeded ${speedLimitExceedsCount} time${isPlural ?
's' : ''}`;
},
rpmLimitExceeded: ({ rmpThreshold, rpmLimitExceedsCount }) => {
const isPlural = rpmLimitExceedsCount > 1;
return `RPM limit (${rmpThreshold} rpm/m) was exceeded ${rpmLimitExceedsCount} time${isPlural ? 's' :
''}`;
},
coolantTemperatureLimitExceeded: ({ coolantTempThreshold,
coolantTempLimitExceedsCount }) => {
const isPlural = coolantTempLimitExceedsCount > 1;
return `Coolant temperature limit (${coolantTempThreshold} °C) was exceeded
${coolantTempLimitExceedsCount} time${isPlural ? 's' : ''}`;
}
};
export class AlertsService {
constructor(
private readonly alertsRepository: AlertsRepository,
private readonly tripsRepository: TripsRepository,
private readonly carsService: CarsService,
private readonly postgresClient: Knex,
) {
this.alertsRepository = alertsRepository;
this.tripsRepository = tripsRepository;
```

```
this.carsService = carsService;
this.postgresClient = postgresClient;
}
async getAllAlertsBaseInfoWithPagination({
carId,
limit,
skip,
sort,
order
}:
GetAllAlertsWithPaginationFuncParams): Promise<GetAllAlertsReturnModel> {
await this.carsService.checkCarExistenceById({ carId });
const [records, total] = await Promise.all([
this.alertsRepository.getAllCarsBaseInfoWithPagination({ carId, limit, skip, sort, order }),
this.alertsRepository.getAllAlertsTotalCountInOrganisation({ carId }),
]);
return { records, total };
}
async processMetricsForAlertsCreation({
tripId,
carId,
metrics
}: { tripId: TripId, carId: CarId, metrics: Array<TripMetricModel> }): Promise<void> {
const carThresholds = await this.carsService.getCarThresholds({ carId });
const alertsToCreate: CreateAlertFuncParams[] = [];
const tripMetricIds = metrics.map((metric) => metric.id);
const speedLimitExceedsCount = this.getSpeedLimitExceedsCount({ metrics, speedThreshold:
carThresholds.speed });
const rpmLimitExceedsCount = this.getRpmLimitExceedsCount({ metrics, rpmThreshold:
carThresholds.rpm });
const coolantTempLimitExceedsCount = this.getCoolantTemperatureLimitExceedsCount({ metrics,
coolantTempThreshold: carThresholds.coolantTemp });
if (speedLimitExceedsCount) {
const description = limitationsDescriptionBuilders.speedLimitExceeded({
speedThreshold: carThresholds.speed, speedLimitExceedsCount });
alertsToCreate.push({ carId, tripId, description });
}
if (rpmLimitExceedsCount) {
const description = limitationsDescriptionBuilders.rpmLimitExceeded({ rmpThreshold:
```

```
carThresholds.rpm, rpmLimitExceedsCount });
alertsToCreate.push({ carId, tripId, description });
}
if (coolantTempLimitExceedsCount) {
const description = limitationsDescriptionBuilders.coolantTemperatureLimitExceeded({
coolantTempThreshold: carThresholds.coolantTemp,
coolantTempLimitExceedsCount
});
alertsToCreate.push({ carId, tripId, description });
}
await this.postgresClient.transaction(async (transactionQuery) => {
for (const alertToCreate of alertsToCreate) {
await this.alertsRepository.createAlert(alertToCreate, transactionQuery);
}
await this.tripsRepository.updateTripMetricsBulk(tripMetricIds, { processed: true }, transactionQuery);
});
}
getSpeedLimitExceedsCount({ metrics, speedThreshold }) {
return metrics.reduce((accumulator: number, currentMetric: TripMetricModel | null,
currentIndex: number, array) => {
const previousMetric = array[currentIndex - 1];
const limitExceedInPreviousMetric = previousMetric && this._isSpeedLimitExceeded({ speedThreshold,
averageSpeedPerInterval: previousMetric.averageSpeedPerInterval
});
const limitExceedInCurrentMetric = currentMetric && this._isSpeedLimitExceeded({ speedThreshold,
averageSpeedPerInterval: currentMetric.averageSpeedPerInterval
});
if (!limitExceedInPreviousMetric && limitExceedInCurrentMetric) accumulator += 1;
return accumulator;
}, 0);
}
getRpmLimitExceedsCount({ metrics, rpmThreshold }) {
return metrics.reduce((accumulator: number, currentMetric: TripMetricModel | null,
currentIndex: number, array) => {
const previousMetric = array[currentIndex - 1];
const limitExceedInPreviousMetric = previousMetric && this._isRPMLimitExceeded({ rpmThreshold,
averageRPMPerInterval: previousMetric.averageRPMPerInterval
});
const limitExceedInCurrentMetric = currentMetric && this._isRPMLimitExceeded({ rpmThreshold,
```

```
averageRPMPerInterval: currentMetric.averageRPMPerInterval
});
if (!limitExceedInPreviousMetric && limitExceedInCurrentMetric) accumulator += 1;
return accumulator;
}, 0);
}
getCoolantTemperatureLimitExceedsCount({ metrics, coolantTempThreshold }) {
return metrics.reduce((accumulator: number, currentMetric: TripMetricModel | null, currentIndex: number, array) => {
const previousMetric = array[currentIndex - 1];
const limitExceedInPreviousMetric = previousMetric && this._isCoolantTemperatureLimitExceeded({
coolantTempThreshold, coolantTemp: previousMetric.coolantTemp
});
const limitExceedInCurrentMetric = currentMetric &&
this._isCoolantTemperatureLimitExceeded({
coolantTempThreshold,
coolantTemp: currentMetric.coolantTemp
});
if (!limitExceedInPreviousMetric && limitExceedInCurrentMetric) accumulator += 1;
return accumulator;
}, 0);
}
_isSpeedLimitExceeded({ speedThreshold, averageSpeedPerInterval }) {
if (!speedThreshold || !averageSpeedPerInterval) return false;
return averageSpeedPerInterval > speedThreshold;
}
_isRPMLimitExceeded({ rpmThreshold, averageRPMPerInterval }) {
if (!rpmThreshold || !averageRPMPerInterval) return false;
return averageRPMPerInterval > rpmThreshold;
}
_isCoolantTemperatureLimitExceeded({ coolantTempThreshold, coolantTemp }) {
if (!coolantTempThreshold || !coolantTemp) return false;
return coolantTemp > coolantTempThreshold;
}
}
export default new AlertsService(alertsRepository, tripsRepository, carsService, postgresClient.getClient());
```

ДодатокГ
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Автоматизована система контролю стану автомобіля через OBD2 інтерфейс

Студент групи АКІТ-22мс

Вадим КРУЧКО

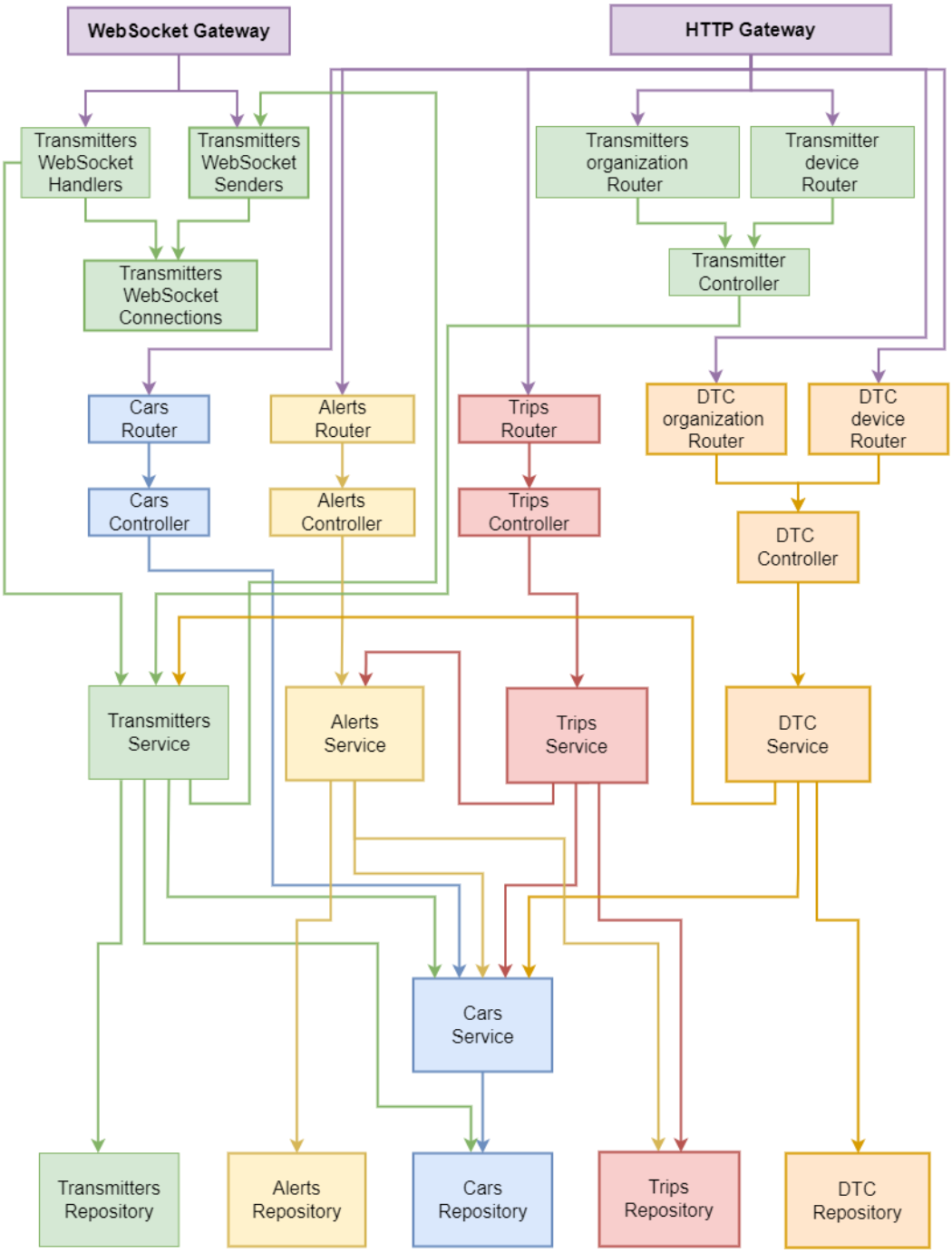
Керівник к.т.н., доц. каф. КСУ

Олена НИКИТЕНКО

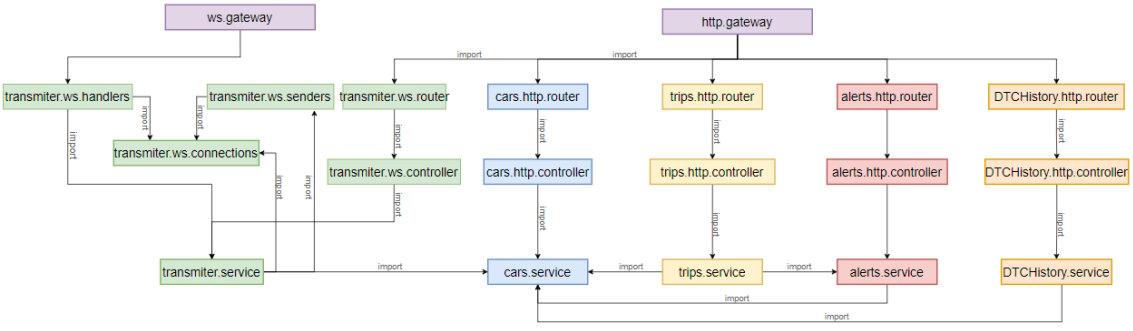
Вінниця 2024

Перелік ілюстративних матеріалів:

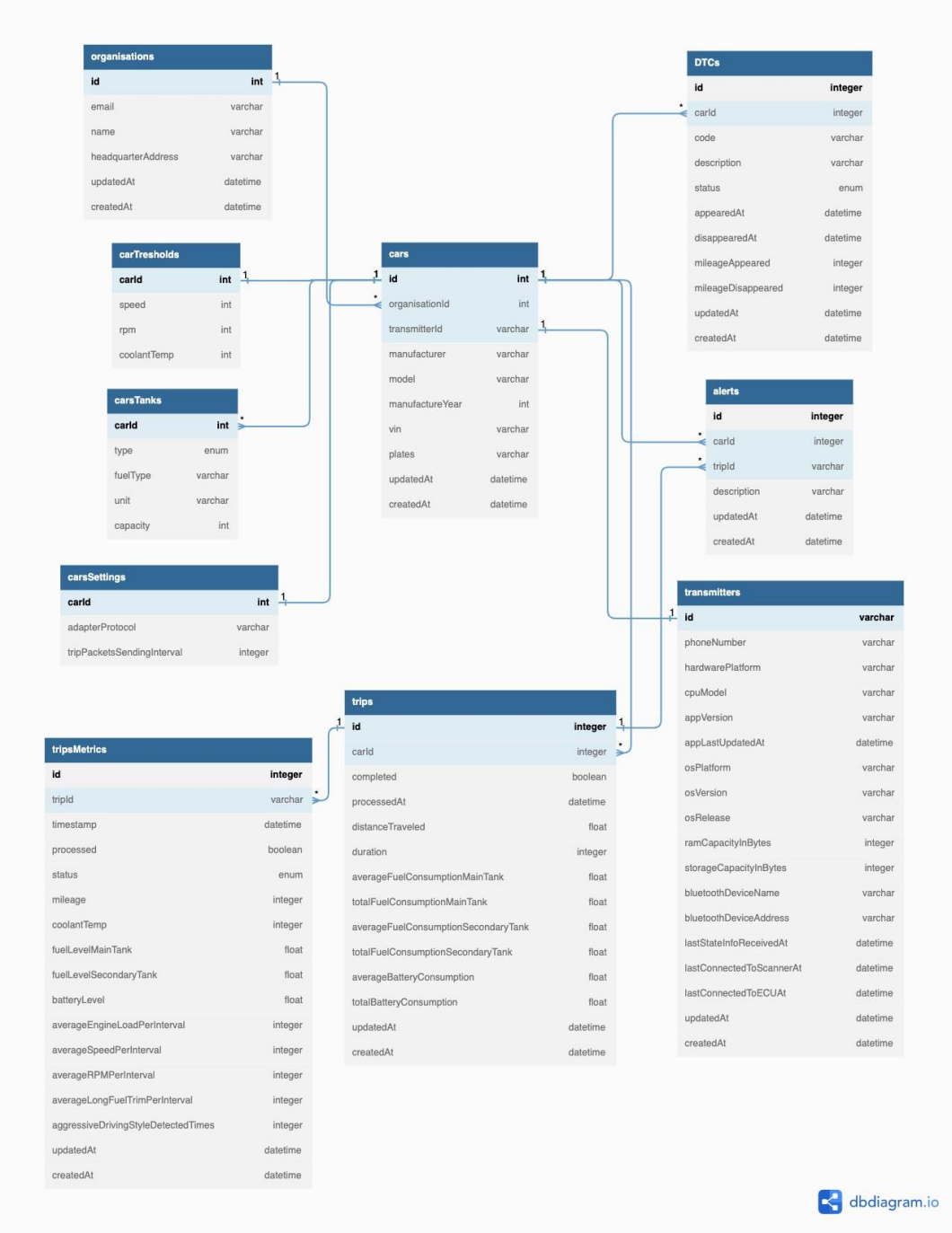
Взаємодія компонентів системи



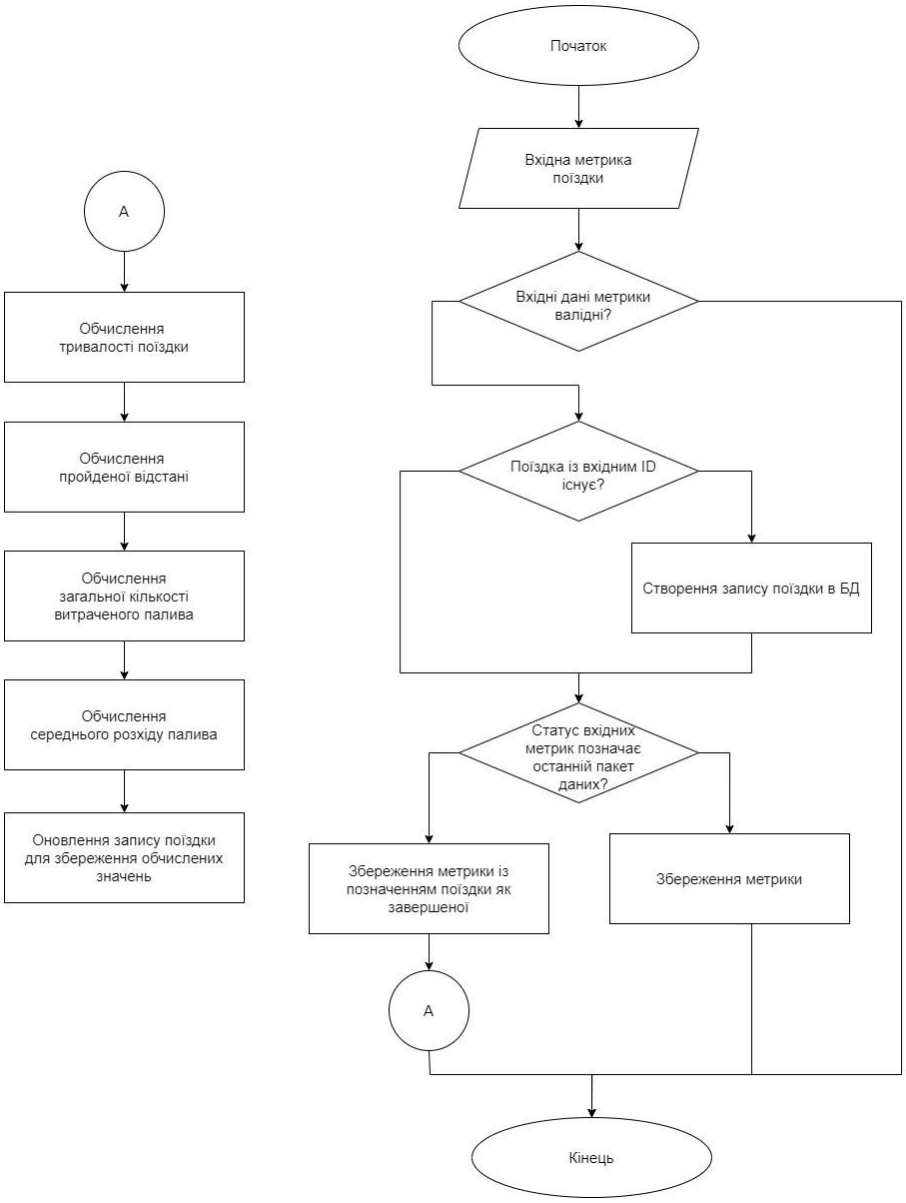
Базова структурна схема системи



Спрощена діаграма архітектури серверного додатку



Модель бази даних



Базовий алгоритм роботи системи

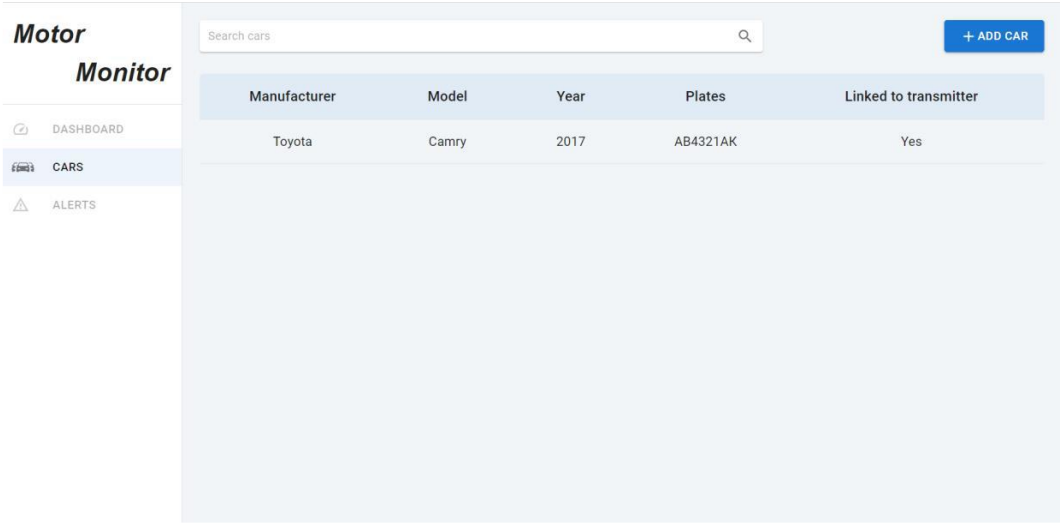
Діагностичний сканер ELM 327 [14]

```
export interface DTCModel {
  id: DTCId;
  carId: CarId;
  code: DTCCode;
  description: DTCDescription;
  status: DTCStatus;
  appearedAt: DTCAppearedAt;
  disappearedAt: DTCDisappearedAt;
  mileageAppeared: DTCMileageAppeared;
  mileageDisappeared: DTCMileageDisappeared;
  createdAt: Date;
  updatedAt: Date;
}
```

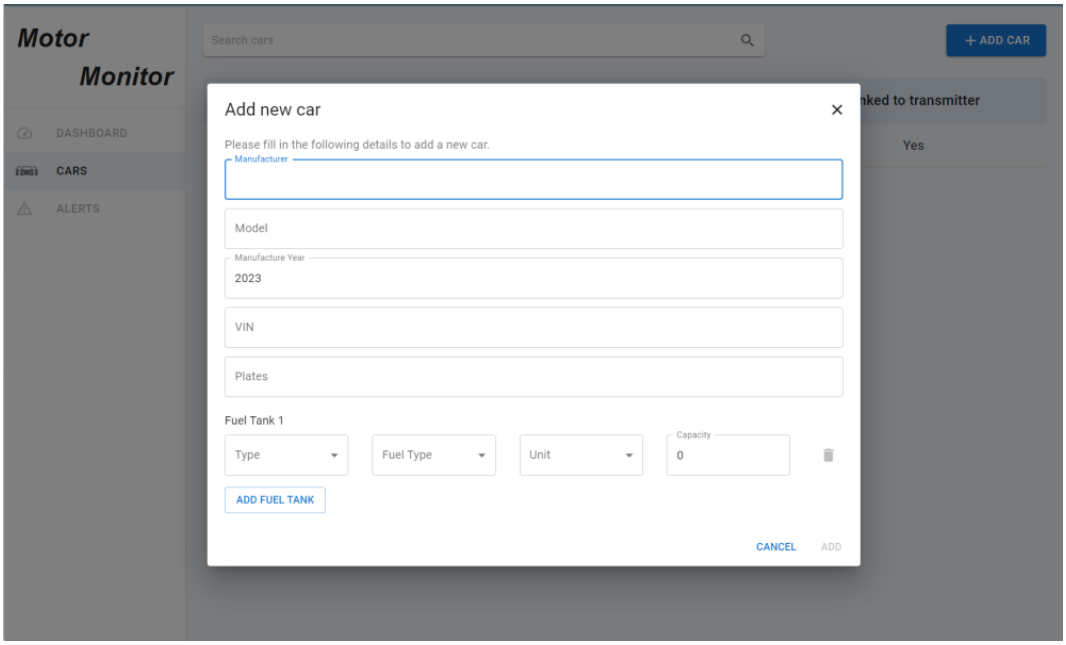
Об’єкт діагностичної помилки

```
export interface TransmitterStateInfoModel {
  carId: CarId;
  phoneNumber: PhoneNumber;
  connectedBluetoothDevice: BluetoothDeviceModel | null;

  connectedToScanner: ConnectedToScanner;
  lastConnectedToScannerAt: LastConnectedToScannerAt;

  connectedToECU: ConnectedToECU;
  lastConnectedToECUAt: LastConnectedToECUAt;

  activeOBDProtocol: ActiveOBDProtocol;
  appVersion: AppVersion;
  appLastUpdatedAt: LastUpdateAt;
  osVersion: OsVersion;
  osRelease: OsRelease;
  usedStorageCapacityInBytes: UsedStorageCapacityInBytes;
  settings: CarSettingsModel;
  thresholds: CarThresholdsModel;
}
```

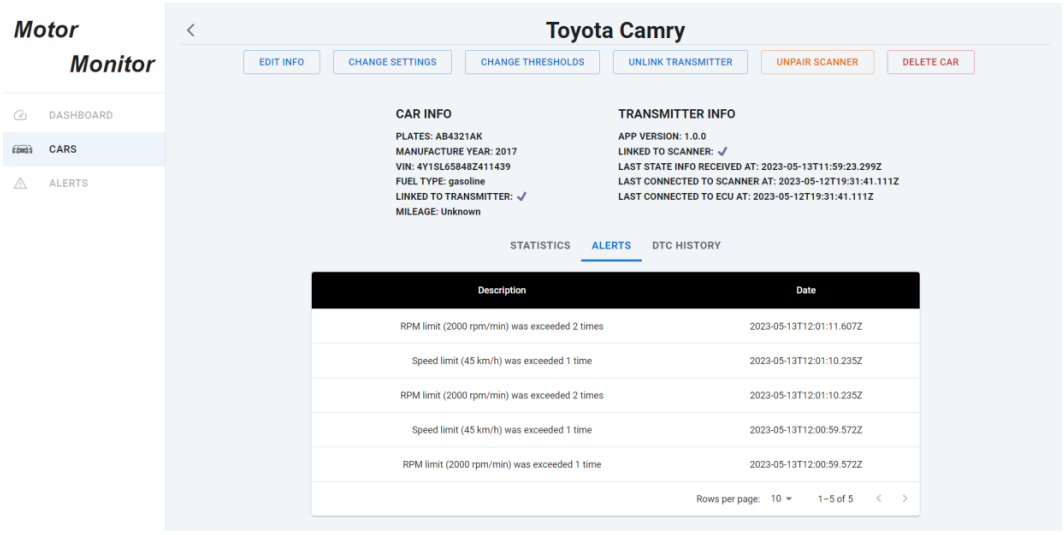
Об’єкт поточного статусу девайса



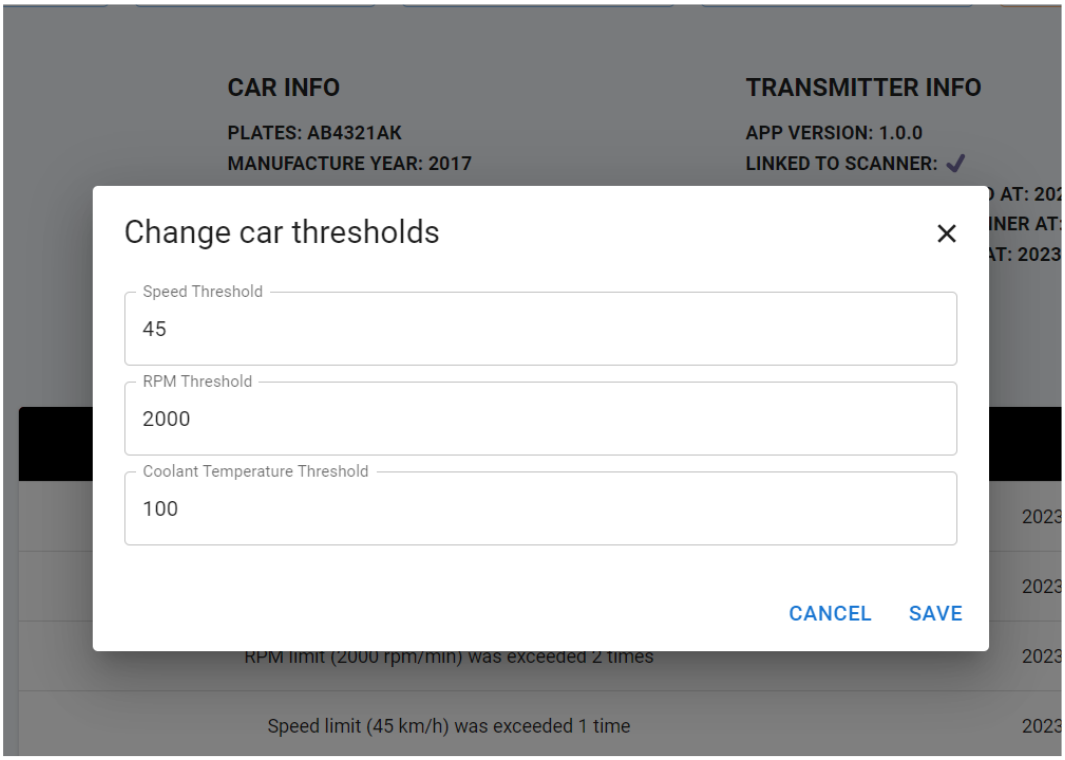
Початкова сторінка інтерфейсу



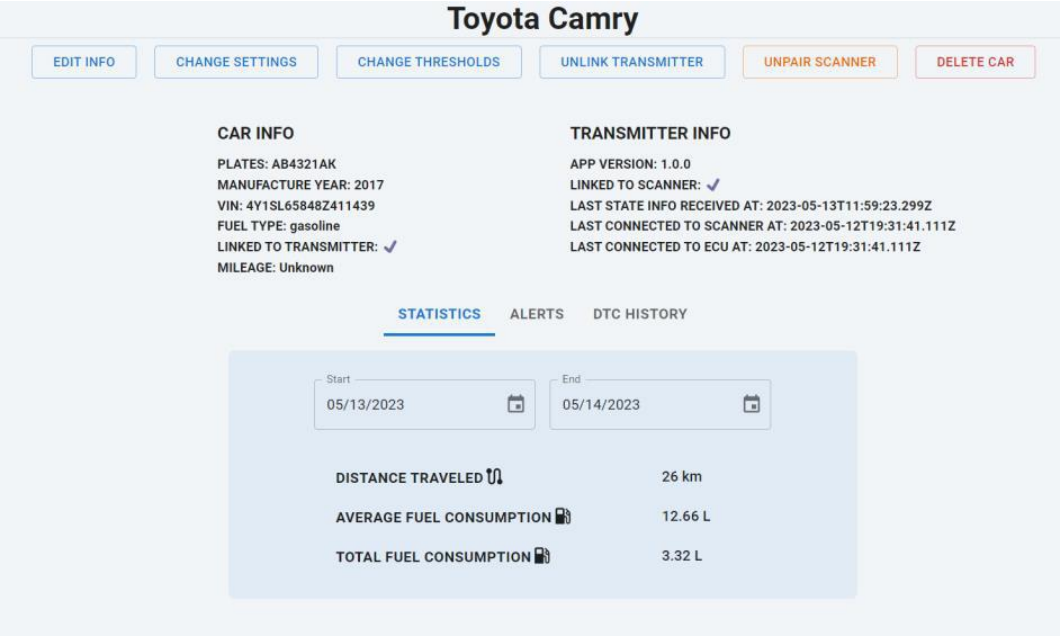
Форма для створення реєстрації нового авто



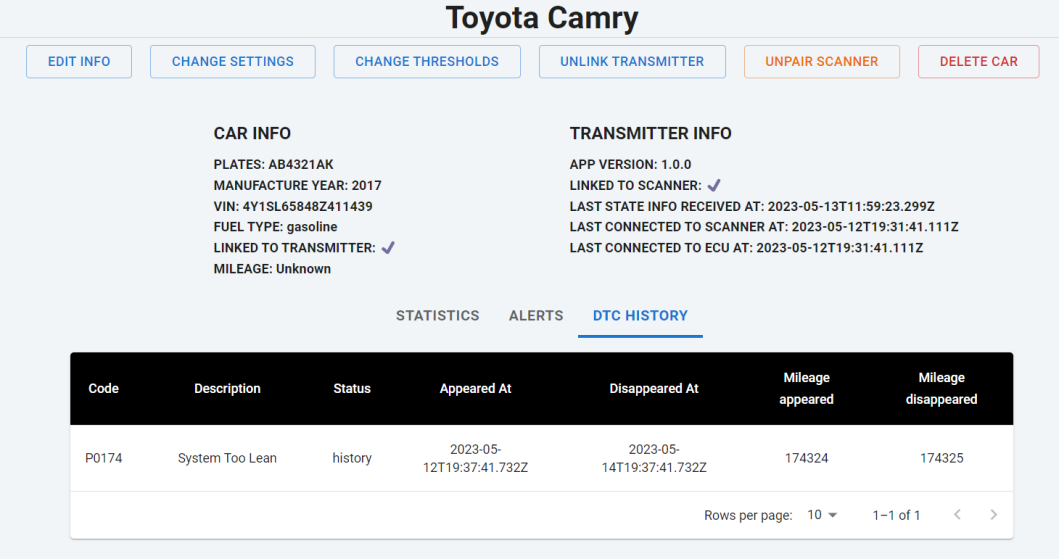
Сторінка детальної інформації про автомобіль



Форма для зміни порогових значень для автомобіля



Вкладка статистики за здійсненими поїздкам



Вкладка історії діагностичних помилок