НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО:
В. о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____202_ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

спеціальності 121 Інженерія програмного забезпечення
на тему: «Програмне забезпечення для прогнозування споживання пального на основі моделей машинного навчання»

Виконала:
студентка IV курсу, групи ТВ-11
Гундяк Валерія Русланівна _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
Доцент, к.е.н., Гусєва Ірина Ігорівна _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ — 2025

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

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійна програма: «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«___»_____202_ р.

ЗАВДАННЯ
на дипломну роботу студенту
Гундяк Валерія Русланівна
(прізвище, ім’я, по батькові)

1. Тема роботи
Програмне забезпечення для прогнозування споживання пального на основі моделей машинного навчання
керівник роботи Гусєва Ірина Ігорівна, к.е.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом по університету від «00 » місяць 2025 р. № 2197-с

2. Строк подання студентом роботи 9 червня 2025 р.
3. Вихідні дані до роботи мова програмування Kotlin, середовище розробки Android Studio, мова програмування Python, середовище розробки VS Code, середовище для управління контейнерами Docker
4.Зміст пояснювальної записки (перелік питань, які потрібно розробити) розробити моделі машинного навчання для прогнозування споживання пального за даними з смартфона; розробити мобільний застосунок з інтеграцією навчених моделей машинного навчання, здатний до збору даних з датчиків смартфона.

5. Перелік ілюстративного матеріалу Діаграма взаємодії частин системи, Діаграма модулів архітектури сервера, Діаграма прецедентів програмного застосунку, Діаграма структури бази даних.

6. Дата видачі завдання «30» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітки
1.	Отримання завдання	30.10.2024	Виконано
2.	Дослідження предметної області	31.10.2024 – 31.12.2024	Виконано
3.	Постановка вимог до проєктування системи	01.01.2025 – 15.02.2025	Виконано
4.	Дослідження існуючих рішень	16.02.2025 – 01.04.2025	Виконано
5.	Розробка програмного продукту	01.04.2025 – 05.05.2025	Виконано
6.	Тестування програмного продукту	05.05.2025 – 12.05.2025	Виконано
7.	Захист програмного продукту	12.05.2025	Виконано
8.	Оформлення дипломної роботи	12.05.2024 – 23.05.2024	Виконано
9.	Передзахист	02.06.2024	Виконано
10.	Захист	09.06.2024	

Студент _____ _____Гундяк В.Р._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Гусєва І.І._____
(підпис) (прізвище та ініціали,

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 50 сторінки, 20 рисунків, 2 додаток та 17 посилань.

Метою роботи є створення програмного забезпечення для прогнозування споживання пального з використанням моделей машинного навчання на основі даних сенсорів смартфона.

Для досягнення поставленої мети було проведено аналіз існуючих методів прогнозування споживання пального пального за допомогою моделей машинного навчання, спроєктовано архітектуру системи, створено алгоритм трансформації вхідних даних для проведення прогнозування, реалізовано та протестовано декілька моделей машинного навчання для прогнозування споживання пального.

Розроблено мобільний застосунок, що в режимі реального часу прогнозує споживання пального трьома різними моделями машинного навчання, зберігає дані про поїздку та надає загальну статистику поїздок користувача.

Практичне значення одержаних результатів полягає у створенні програмного забезпечення, що надасть водію прогноз споживання пального на данну поїздку, а також статистичні дані щодо попередніх.

Ключові слова: мобільний застосунок, споживання пального, моделі машинного навчання, сенсори смартфона.

ABSTRACT

Structure and scope of the thesis. The work contains 50 pages, 20 figures, 2 appendices and 17 references.

The purpose of the work is to create a software for predicting fuel consumption using machine learning models based on smartphone sensors data.

To achieve this goal, existing methods for predicting fuel consumption using machine learning models were analyzed, system architecture was designed, algorithm for transforming input data for forecasting was created, several machine learning models for predicting fuel consumption were implemented and tested.

A mobile application has been developed that predicts fuel consumption in real time using three different machine learning models, stores trip data, and provides general statistics on user trips.

The practical significance of the obtained results lies in creating software that provides the driver with a forecast of fuel consumption for a given trip, as well as statistics on previous ones.

Keywords: mobile application, fuel consumption, machine learning models, smartphone sensors.

ЗМІСТ

ВСТУП 8

1 ПОСТАВНОВКА ЗАВДАННЯ ПРОГНОЗУВАННЯ СПОЖИВАННЯ ПАЛЬНОГО 11

Висновки до розділу 1 12

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 13

2.1 Прогнозування споживання пального 13

2.2 Випадкові ліси 13

2.3 Нейромережа зворотного поширення похибки 14

2.4 Баєсова гребенева регресія 16

2.5 Порівняння обраних моделей машинного навчання 17

2.6 Аналіз аналогічних систем 18

Висновки до розділу 2 21

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАСТОСУНКУ 22

3.1 Обґрунтування вибору мови програмування 22

3.2 Обґрунтування вибору системи керування базами даних 24

3.2 Обґрунтування вибору середовищ роботи 25

Висновки до розділу 3 27

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 28

4.1 Архітектура програмного застосунку 28

4.2 Моделювання взаємодії між користувачем та системою 33

4.3 Архітектура бази даних 35

4.4 Прогнозування споживання пального 36
4.5 Тренування моделей машинного навчання 39
Висновки до розділу 4 43
5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ 44
5.1 Встановлення та налаштування програмного застосунку 44
5.2 Прогнозування споживання пального 48
Висновки до розділу 5 50
ВИСНОВКИ 51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52
ДОДАТОК А 54
ДОДАТОК Б 70

ВСТУП

Моделі машинного навчання вже достатньо широко інтегровані у різні сфери життя людей. Рекомендаційні алгоритми різного роду медіа сервісів, моделі розпізнавання голосового вводу для текстової трансформації, визначення об'єктів на зображеннях – є лише обмеженою частиною систем, що базуються на машинному навчанні. Звичайно, сфера транспортних засобів та пов'язані з нею напрямки не оминули цю тенденцію. Одними з найбільш очевидних прикладів є система транспортного автопілота та система прокладання оптимального маршруту.

Актуальність теми зумовлена сучасною проблематикою у сфері транспорту, яка пов'язана з економічними та екологічними чинниками. В умовах зростання цін на пальне, а також потреби у зменшенні шкідливих викидів в атмосферу, особливої вагомості набуває точний моніторинг і прогнозування споживання палива. Використання моделей машинного навчання у цій сфері дозволяє перейти від загальних нормативних розрахунків до персоналізованих і точних прогнозів. Тобто, технології штучного інтелекту мають можливість зробити вагомий внесок у підвищення енергоефективності та скорочення витрат у транспортній галузі.

Смартфони є не тільки багатофункціональними пристроями, а ще й є високо інтегрованими у побут. Більше чотирьох мільярдів людей у світі володіють смартфоном [1]. Вони оснащені GPS-модулями, акселерометрами, гіроскопами та іншими сенсорами, що дозволяють в режимі реального часу отримувати великі обсяги даних, що характеризують переміщення, в тому числі безпосередньо поїздки в автомобілі. Це надає можливість аналізу поведінки водія, опираючись на певні ознаки отримані з даних сенсорів смартфона і відповідно прогнозування потенційного споживання палива. Використання таких рішень є не лише технічно доцільним, але й практичним, оскільки не потребує дорогого устаткування.

Традиційні методи оцінки споживання пального ґрунтуються на показниках, які не враховують індивідуальні особливості водіння, дорожні умови

та стан автомобіля (безпосередньо у чистих даних). У результаті такі підходи зазнають певної втрати точності. Водночас машинне навчання дозволяє знаходити закономірності у великих обсягах в тому числі нелінійних даних, що дозволить забезпечити точніше прогнозування.

Практична цінність розробки полягає у тому, що вона поєднує мобільні технології збору даних та машинне навчання в єдину систему. Такий підхід відповідає сучасним технологічним тенденціям та має значний потенціал для подальшого розвитку.

Згідно з результатами соціологічного опитування агенції Info Sapiens [2] близько 49% українців володіють автомобілем, проте ці числа є неточними, позаяк соціологічні опитування не можуть враховувати кожного громадянина країни. Натомість станом на 19 березня 2024 року, за даними Єдиного державного реєстру транспортних засобів [3], на обліку в Україні перебувало 13,16 млн транспортних засобів, з яких 9,8 млн — легкових.

Відповідно, на даний період часу Україна не поступається рівнем попиту в транспортній сфері глобальним тенденціям. Водночас разом з цим паливо, як ресурс, який напряму прив’язаний до транспорту, так само потрібний як з позиції пересічних громадян, так й державних установ та приватних транспортних бізнесів.

Незважаючи на те, що початкова дестабілізація продажу палива та його дефіцит на початку повномасштабного вторгнення була врегульована, все ще існує потреба вміння аналізувати споживання пального, для їх регулювання. Тому інтеграція моделей машинного навчання для прогнозування споживання палива відповідно до поведінки водія, є логічним кроком для транспортної сфери.

Метою роботи є створення програмного забезпечення для прогнозування споживання пального з використанням моделей машинного навчання на основі даних сенсорів смартфона, що дозволить водію аналізувати власний стиль водіння для зменшення своїх витрат, а також мати змогу ознайомитись з вибіркою моделей, які будуть використанні для прогнозування, та порівняти їхню ефективність.

Об’єктом дослідження є прогнозування споживання палива моделями

машинного навчання.

Відповідно, предметом дослідження є програмне забезпечення збору і прогнозування споживання палива автомобілем. Вони забезпечать можливість створення сприятливих умов для економії палива та порівняння ефективності різних моделей машинного навчання.

1 ПОСТАВНОВКА ЗАВДАННЯ ПРОГНОЗУВАННЯ СПОЖИВАННЯ ПАЛЬНОГО

Метою роботи є створення програмного забезпечення для прогнозування споживання пального з використанням моделей машинного навчання на основі даних сенсорів смартфона. Застосунок дозволить водіям аналізувати власний стиль водіння для зниження споживання палива, а також порівнювати ефективність різних моделей прогнозування.

Завдання, що потребують вирішення для досягнення мети:

– провести аналіз існуючих методів прогнозування споживання пального за допомогою моделей машинного навчання;

– дослідити та підготувати відповідні дані для навчання моделей (включаючи GPS-дані, швидкість, час, тощо);

– розробити архітектуру системи;

– розробити базу даних системи;

– розробити мобільний застосунок зі збору та обробки даних з сенсорів смартфона;

– розробити алгоритми перетворення даних та прогнозування споживання;

– реалізувати та протестувати декілька моделей машинного навчання для прогнозування споживання пального.

Вхідними даними є дата та час збору даних, висота над рівнем моря, широта та довгота GPS-модуля смартфона, зміна координат, прискорення вздовж осей X, Y, Z з акселерометра та кутові прискорення вздовж осей X, Y, Z з гіроскопа.

Вихідними даними є прогнозоване споживання пального для поїздки, узагальнена статистика споживання пального на основі збережених поїздок, таблиці з результатами прогнозування від різних моделей.

Основні функції системи включають: збір даних з сенсорів смартфона (GPS, акселерометр, гіроскоп), обробку та трансформацію зібраних даних, прогнозування

споживання пального за допомогою моделей машинного навчання, збереження історії поїздок і результатів прогнозів, візуалізацію прогнозованого споживання пального, надання користувачу доступу до статистики та порівняння ефективності різних моделей.

Користувачі застосунку:

– водії, які бажають контролювати та оптимізувати своє споживання пального;

– приватні транспортні компанії, які можуть інтегрувати застосунок для аналізу ефективності водіїв.

Висновки до розділу 1

У розділі 1 було описано мету роботи та завдання, які необхідно вирішити для досягнення поставленої мети. Визначено головні функції системи, які полягають у трансформації даних, прогнозування споживання та виведення результатів користувачу.

Було описано вхідні та вихідні дані та визначено користувачів застосунку.

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Прогнозування споживання пального

Предметна область охоплює аналіз і прогнозування споживання пального автомобіля на основі персоналізованих даних користувача, що збираються його смартфоном під час поїздки без збереження для прогнозування в реальному часі та після завершення в повному обсязі передаються на сервер для внесення фінальних даних та прогнозів у базу даних.

Прогнозування споживання пального є завданням, метою якого є передбачити кількість пального, що буде спожита транспортним засобом, базуючись на вибірці вхідних параметрів. Такими параметрами можуть бути швидкість, прискорення, маршрут, топографічні особливості дороги, поведінка водія тощо.

У сфері інтелектуального транспорту застосовуються різні моделі прогнозування, як класичні статистичні методи так й складніші моделі машинного навчання (дерева рішень, нейронні мережі тощо). Ключовими критеріями ефективності прогнозної моделі є точність, стійкість до шуму у даних та здатність адаптуватися до індивідуального стилю водіння.

Використання прогнозування споживання пального дозволяє покращити економність водіння та покращити досвід водія.

2.2 Випадкові ліси

Випадкові ліси є підходом машинного навчання, що вирішує проблему класифікації та регресії. Цей метод використовує ансамблеве навчання (вирішення складних завдань шляхом інтеграції багатьох класифікаторів), та базується на використанні значного обсягу дерев рішень. Алгоритм усереднює результати прогнозів різних дерев, формуючи кінцевий прогноз для користувача. Відповідно,

зі збільшенням вибірки дерев точність прогнозу зростає (рисунок 2.1) [4].

Головними перевагами даного методу є стійкість до перенавчання, здатність встановлювати зв’язки між ознаками та вміння оцінювати вагу кожної з ознак.

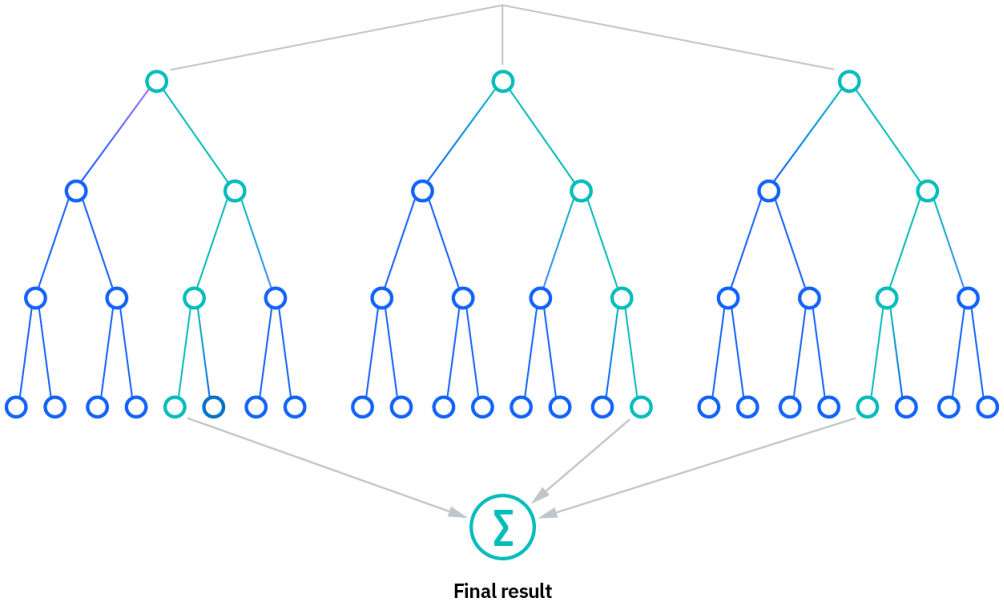


Рисунок 2.1 – Структура моделі Випадкових лісів

Тобто модель випадкових лісів є хорошим вибором для застосунку позаяк:

– добре працює з великою кількістю вхідних параметрів (швидкість, прискорення, сповільнення тощо);

– стійка до шумів у даних, які є поширеними у даних отриманих з сенсорів, особливо зважаючи на потенційне тремтіння смартфона при поїздці;

– не вимагає масштабування даних;

– забезпечує точні прості у сприйняті прогнози.

2.3 Нейромережа зворотного поширення похибки

Зворотне поширення похибки є популярною та легкою моделлю навчання для складних, багатошарових мереж. Вона має вхідний та вихідні прошарки та хоча б один (переважно кількість варіюється від одного до двох) прихований прошарок

(рисунок 2.2) [5].

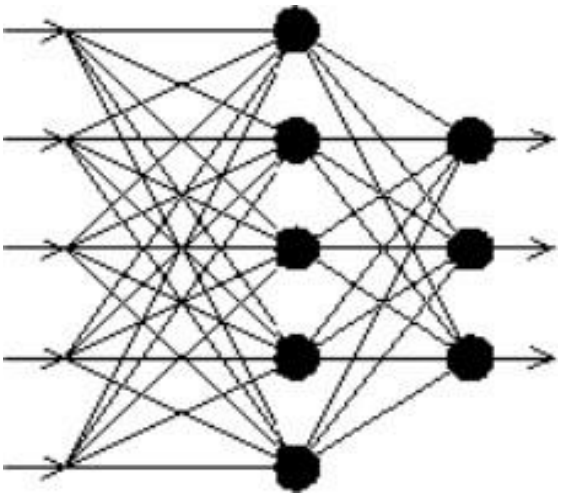


Рисунок 2.2 – Структура нейромережі зворотного поширення похибки

Під час навчання мережа змінює ваги з’єднань (які переважно обираються випадково при першому проходженні) між нейронами за допомогою методу зворотного поширення помилки. Помилка на виході передається назад і використовується для коригування ваг згідно з градієнтним спуском. Основна особливість методу полягає в тому, що його модель має змогу визначати складні, нелінійні зв’язки між ознаками.

Підсумовуючи, модель було обрано з таких причин:

– здатність моделювати складні залежності між ознаками (особливості гальмування при сповільненні, тривалість поїздки тощо);

– гнучкість архітектури;

– здатність до самонавчання, при наявності нових даних.

Натомість до недоліків входить потреба масштабувати дані та схильність до перенавчання.

2.4 Баєсова гребенева регресія

Баєсова регресія, підвидом якої є Баєсова гребенева регресія, дозволяє природному механізму пережити недостатню кількість даних або погано розподілені дані шляхом формулювання лінійної регресії з використанням розподільників ймовірностей, а не точкових оцінок.

Передбачається, що вихідні дані отримані з розподілу ймовірностей, а не оцінені як одне значення. Тобто завдяки узагальненню модель дозволяє оцінити невизначеність у прогнозах і забезпечує більшу стабільність при роботі з невеликими або неоднорідними вибірками, тому її використання вигідне, наприклад, у випадках, коли дані містять певний шум (рисунок 2.3) [6].

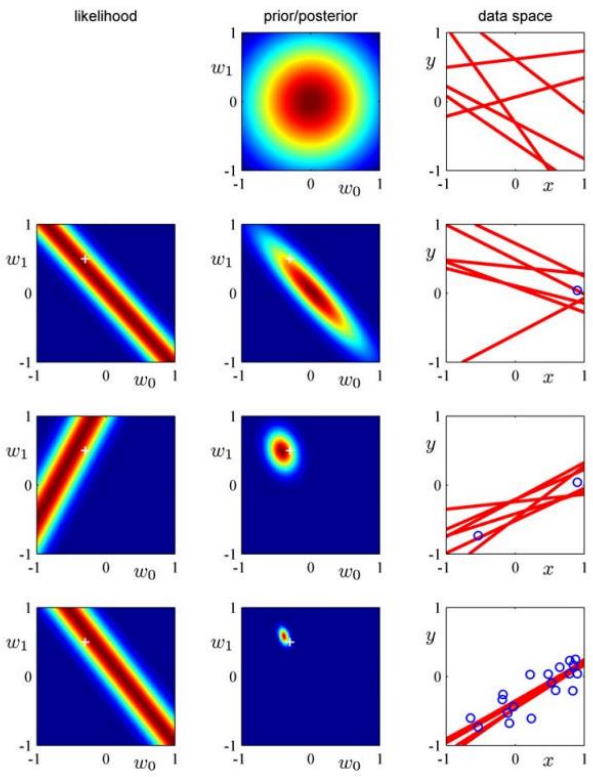


Рисунок 2.3 – Візуалізація Баєсової регресії

Баєсова гребенева регресія особлива тим, що обчислює не тільки однозначні ваги, як у звичайній регресії, а й рівень невизначеності в прогнозі, що важливо в умовах нестабільних даних.

Відповідно ця модель обрана, тому що:

– можливо, швидко отримувати прогноз при малій кількості даних (наприклад, у випадку коротких поїздок);

– здійснює оцінку невизначеності;

– зменшує ймовірність перенавчання завдяки регуляризації.

У підсумку було обрано три моделі, результати прогнозування, яких передаватимуться у застосунок користувачу:

– випадкові ліси;

– нейромережа зворотного поширення похибки;

– Баєсова гребенева регресія.

2.5 Порівняння обраних моделей машинного навчання

Отже, для розробки програмного забезпечення прогнозування споживання пального для дипломної роботи було розглянуто три моделі машинного навчання: випадкові ліси, нейромережа зворотного поширення похибки та Баєсова гребенева регресія. Проаналізувавши кожну з них, було виділено такі архітектурні та концептуальні відмінності.

Випадкові ліси:

— фінальний результат формується шляхом усереднення;

— дозволяє легко інтерпретувати важливість ознак;

— добре впорається з реальними, «зашумленими» даними;

— порівняно швидке навчання (масштабування на великі дані).

Нейромережа зворотного поширення похибки:

— здатність моделювати нелінійні залежності та взаємодії між змінними;

— надмірна схильність до перенавчання;

— невизначено довгий процес навчання.

Баєсова гребенева регресія:

— дозволяє враховувати невизначеність прогнозу;

— добре працює на невеликих та неоднорідних вибірках;

— схильність до уникнення перенавчання;

— схильність не вловлювати нелінійні взаємозв’язки.

Випадкові ліси є хорошим вибором для роботи даними сенсорів смартфона, позаяк одна з основних потреб в данних обчисленнях є стійкість до шуму та уникнення необхідності нормалізувати дані. В той час як нейромережа зворотного поширення похибки сприятиме створенню точніших зв’язків між ознакою та результатом. Баєсова гребенева є простою та швидкою моделлю, яка добре працює з невеликими обсягами даних, що є корисним при коротких поїздках.

2.6 Аналіз аналогічних систем

Одним з прикладів існуючого рішення для оцінки споживання пального та стилю водіння є система EcoDrive, розроблена компанією Ruptela. Ця система є частиною ширшого телематичного комплексу для моніторингу автопарку.

Система Ruptela EcoDrive дозволяє в режимі реального часу:

— відстежувати параметри поїздки (швидкість, прискорення, гальмування, холостий хід);

— оцінювати стиль водіння за шкалою ефективності (екорейтинг);

— визначати втрату пального з використанням алгоритмів оцінки за GPS даними.

Однією з ключових функцій є індекс якості водіння, що визначається за допомогою визначення кількості «шкідливих маневрів»: різке прискорення, гальмування, перевищення швидкості та агресивне входження в повороти. Отримані дані доступні для диспетчерів і керівників автопарку через онлайн-платформу TrustTrack.

Тобто головними недоілками Ruptela EcoDrive є орієнтація на корпоративне управління автопарком, а також потреба у додактовій апаратурі (рисунок 2.4), яка

представлена компанією, для запису даних з її сенсорів. Також Ruptela EcoDrive не прогнозує споживання в даний момент, а більше орієнтована на втрату в споживанні пального внаслідок «шкідливої» поведінки водія.

Рисунок 2.4 – Панель Ruptela EcoDrive

На відміну від Ruptela EcoDrive, розроблюване програмне забезпечення:

— працює без потреби у зовнішньому GPS-трекері чи доступі до CAN-шини, використовуючи лише сенсори смартфона;

— має орієнтацію на приватного користувача, а не корпоративне управління автопарком;

— надає прогнозування споживання пального в процесі поїздки.

Таким чином Ruptela EcoDrive є ефективним інструментом для моніторингу стилю водіння в професійному середовищі, в той час як запропоноване рішення є мобільною, універсальною альтернативою для широкого кола користувачів, що не потребує спеціалізоване обладнання чи професійне обслуговування [7].

Також одним із провідних телематичних рішень для моніторингу стилю водіння та оптимізації споживання пального є система EcoDrive, інтегрована в платформу MyACTIAFLEET, розроблену компанією ACTIA. Так само, як і

попередня система, це рішення орієнтоване на операторів громадського транспорту та великих автопарків і поєднує апаратне забезпечення (телематичні модулі, дисплеї) з хмарною платформою для аналітики (рисунок 2.5).

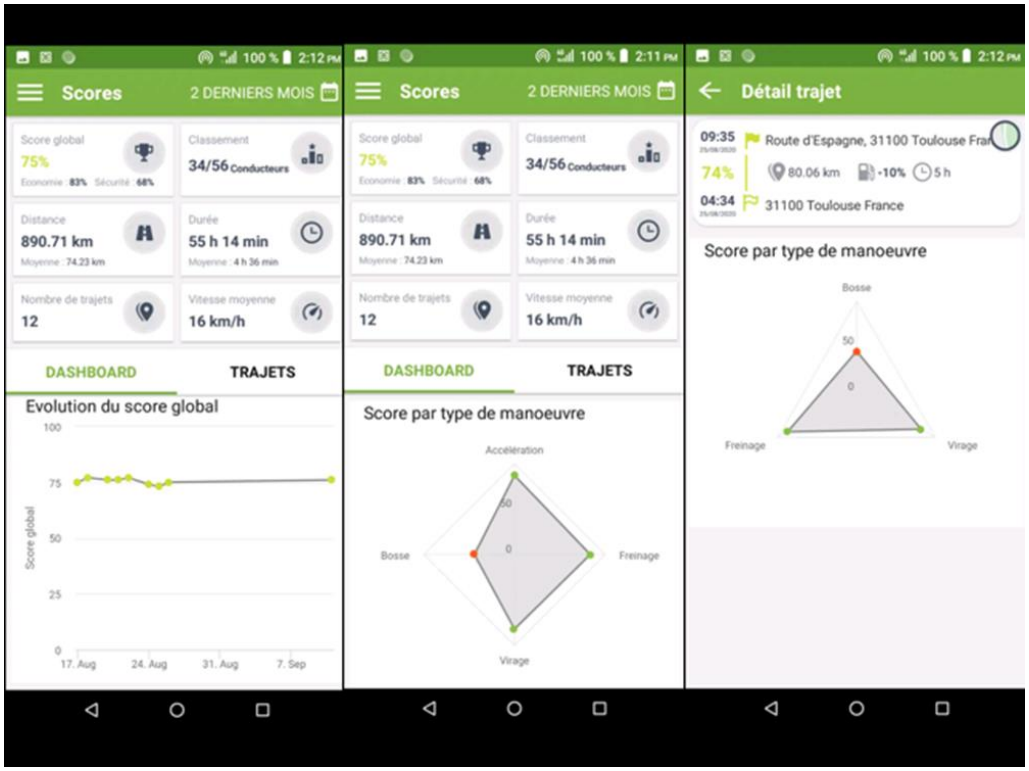


Рисунок 2.5 – Застосунок ACTIA EcoDrive

Система забезпечує:

— моніторинг споживання пального та викидів CO_2 для кожного транспортного засобу або водія;

— оцінку стилю водіння, що враховують такі параметри, як прискорення, гальмування, повороти та інші маневри;

— аналітичні звіти та дашборди для менеджерів автопарку, включаючи рейтинги водіїв, аналіз подій та рекомендації щодо навчання;

— підтримку різних типів енергії: дизель, газ, електро та гібридні транспортні засоби;

— прогнозування втрат пального на основі стилю водіння.

Проте так само, як й Ruptela EcoDrive, система потребує наявність додаткового обладнання та орієнтована на корпорації, а не на персональне користування [8].

Висновки до розділу 2

У розділі 2 було проведено всебічний аналіз предметної області програмного забезпечення для прогнозування споживання пального на основі даних сенсорів смартфона. Було оглянуто та обґрунтовано три моделі машинного навчання: випадкові ліси, нейромережа зворотного поширення похибки та Баєсова гребенева регресія. Кожна з моделей має переваги в контексті обробки шумних або обмежених даних та здатність моделювати складні залежності.

Також було проаналізовано існуючі комерційні рішення, такі як Ruptela EcoDrive та ACTIA EcoDrive. Ці системи демонструють високу ефективність у контексті керування автопарками, проте мають низку обмежень: потребують додаткове обладнання, орієнтовані переважно на корпоративне використання та не пропонують повноцінного прогнозування споживання пального в реальному часі.

На відміну від них, запропоноване рішення є мобільним застосунком, що не потребує додаткового обладнання й використовує лише сенсори смартфона, що значно покращує його доступність та спрощує використання. Крім того, застосунок орієнтований на індивідуального користувача, забезпечує прогнозування споживання пального впродовж поїздки та може адаптуватися до персонального стилю водіння.

Отже, у цьому розділі було закладено методологічну основу для реалізації програмного забезпечення та чітко визначено контекст, конкурентне середовище та унікальні переваги запропонованої системи порівняно з існуючими аналогами.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Обґрунтування вибору мови програмування

Під час вибору мови програмування було розглянуто такі основні потреби, як підтримка фреймворків машинного навчання та зручна робота з моделями безпосередньо у коді, ефективна робота та швидкість.

У межах проєкту було обрано мову програмування Python з використанням FastAPI, Uvicorn та асинхронним типом архітектури.

Ключовою частиною архітектури програмного забезпечення виступає сервер, який виконує основні завдання безпосередньо пов’язані з прогнозуванням: отримання даних з мобільного застосунку, перетворення даних для навчання, зберігання даних про поїздку, передача перетворених даних до моделей машинного навчання, повернення результатів прогнозування користувачу та обчислення простих статистичних обрахунків на основі наявних поїздок. Позаяк взаємодія між сервером та застосунком взаємодія з клієнтом відбувається за допомогою HTTP-запитів, була необхідність обрати продуктивний, ефективний, з мінімізацією затримок та витривалий до багатьох паралельних підключень вебфреймворк. Для цього було обрано FastAPI, який, окрім того, має підтримку асинхронності.

Асинхронна модель програмування забезпечить уникнення блокування під час виконання I/O-операцій, як, наприклад під час запису даних до БД, виконання прогнозування моделями машинного навчання тощо. Оскільки користувачі можуть одночасно надсилати велику кількість запитів, а система повинна гарантувати швидке отримання результату, асинхронність допоможе серверу опрацьовувати велику кількість запитів без суттєвої втрати продуктивності.

Зокрема, однією з вагомих причин вибору Python є наявність великої кількості бібліотек для машинного навчання та обробки даних (scikit-learn, numpy, pandas, xgboost, tensorflow), які легкі в підключенні та зрозумілі в користуванні.

Завдяки цьому Python дозволяє інтегрувати обчислювальну частину без необхідності звертатись до інших мов та систем. Оскільки у проєкті для отримання та передачі інформації використовуються JSON-структури, наявне в фреймворку Python FastAPI серіалізування/ десеріалізування даних через моделі на базі pydantic значно покращує роботу з кодом.

Таким чином, обрана мова програмування, завдяки додатковим бібліотекам, фреймворкам та їх функціоналам дозволяє реалізувати масштабований, швидкий, зручний сервер, який здатний до обробки великої кількості паралельних запитів, інтеграції з моделями машинного навчання, а також швидкій реакції при взаємодії з користувачем [9-11].

Поскільки обраний тип програмного забезпечення є мобільний застосунок з'явилась потреба вибору мови програмування не тільки для серверної частини, а й для фронт енд частини, яка буде взаємодіяти з користувачем.

Для реалізації мобільного застосунку було обрано Kotlin як основну мову програмування, а для побудови інтерфейсу користувача — сучасний фреймворк Jetpack Compose. Такий вибір зумовлений низкою практичних і технічних чинників, які прямо пов'язані з потребами проєкту та особливостями мобільного середовища.

Насамперед застосунок має забезпечувати збір даних із сенсорів смартфона (GPS, акселерометр, гіроскоп), візуалізацію збережених поїздок та інтерактивну взаємодію користувача з результатами прогнозування, що надходять із серверної частини. Усі ці задачі вимагають тісної інтеграції з функціоналом Android-пристрою, низького рівня доступу до апаратних компонентів, а також швидкої, реактивної побудови інтерфейсу.

Kotlin є рекомендованою мовою для Android-розробки від Google. Kotlin має лаконічний, безпечний синтаксис та сучасні функції. Це спрощує реалізацію збору даних з сенсорів смартфона та взаємодію між застосунком та сервером завдяки REST API.

Kotlin дає можливість легко взаємодіяти з Android API та без проблем

отримувати дані GPS, сенсорів руху, пам’яті пристрою тощо, що є ключовою задачею фронт енд частини. Також ця мова програмування дозволяє використовувати асинхронні операції через coroutines, мінімізувати null помилки (наприклад, у роботі з API або вихідними даними від сервера).

Для створення інтерфейсу було вирішено використати Jetpack Compose як фреймворк. Jetpack Compose дозволяє реалізувати мінімалістичний, інтуїтивний, порівняно адаптивний інтерфейс, зі швидким оновленням відповідно до вхідних даних (час, координати, попередні поїздки, результати прогнозу тощо). А головне – скорочує обсяг шаблонного коду порівняно з традиційним XML-підходом.

Отже, Kotlin та Jetpack Compose ідеально виконують потребу застосунку до збору та передачі даних сенсорів, дозволяють створити реактивний динамічний інтерфейс, забезпечують стабільність, швидкість і масштабованість мобільного застосунку для створення задовільного досвіду для користувача [12, 13].

3.2 Обґрунтування вибору системи керування базами даних

Для збереження даних про поїздки користувача, а також самих користувачів, з’явилась потреба у виборі бази даних. Було обрано реляційну систему керування базами даних (СКБД) PostgreSQL і відповідні інструменти для роботи з нею SQLAlchemy, Psycopg (для asyncpg) та Alembic.

Основні причини вибору цієї бази даних полягають у таких її перевагах:

– використання зовнішніх ключів та транзакцій, що забезпечує цілісність даних;

– зручна структурованість;

– хороша підтримка аналітичних запитів (у випадку подальшого розширення функціоналу);

– підтримка JSON-полів;

– хороша масштабованість;

– сумісність з асинхронним доступом/запитами через asyncpg.

Для взаємодії з СКБД було обрано SQLAlchemy, позаяк вона дозволяє структурувати таблиці через Python-класи. Це значно покращує узгодженість коду. Міграції у базу даних створювались за допомогою бібліотеки Alembic.

Завдяки оформленню кожної міграції у вигляді файлу міграцій, а також можливості генерувати міграцію на основі створеною в кодовій частині моделлю, ця бібліотека є дуже зручним інструментом, який полегшує роботу з кодом та покращує його структуру. Також, зважаючи на те, що як сам сервер, так й база даних були контейнеризовані, наявність міграцій у вигляді файлів значно полегшує відновлення бази даних в контейнері. Тобто Alembic забезпечує:

- контрольоване оновлення структури бази;
- автоматичну генерацію міграцій за моделями;
- документування змін у базі даних.

Отже, обрані СКБД та інструменти до неї дозволяють структуровано зберігати дані про поїздки та користувачів, забезпечувати ефективну роботу сервера з асинхронним підключенням, а також відстежувати будь-які зміни структури бази даних.

Завдяки цьому проєкт є стабільним, розширюваним і технічно ефективним у використанні [14].

3.2 Обґрунтування вибору середовищ роботи

Програмне забезпечення розроблялось з допомогою таких середовищ: Android Studio для розробки фронт енд частини проєкту, Docker для розгортання серверної частини, VS Code для роботи з серверною логікою та конфігураційними файлами.

Android Studio – це офіційне середовище розробки від Google для створення Android-застосунків.

Основні причини вибору Android Studio:

- інтеграція з Kotlin та підтримка Jetpack Compose;

– вбудовані емулятори пристроїв (та можливість під'єднати фізичний пристрій за потреби);

– автоматизація збірки через Gradle.

Серверна частина проєкту реалізується за допомогою FastAPI, Uvicorn, моделей машинного навчання та бази даних PostgreSQL. Для забезпечення зручності розгортання та тестування було обрано контейнеризувати серверну частину та базу даних через Docker.

Основні причини вибору контейнеризації та Docker:

– автоматичне розгортання у контрольованому оточенні;

– легке масштабування;

– відсутність проблем сумісності між бібліотеками;

– можливість локального тестування повного циклу взаємодії між клієнтом, сервером і базою даних;

– проста відтворюваність середовища.

Використання Docker значно знижує ризики при оновленні або міграції системи.

Visual Studio Code (VS Code) – зручне та продуктивне крос платформне середовище розробки, використовувалося для:

— редагування серверного коду на Python;

— роботи з конфігураційними файлами;

— налагодження скриптів перед перенесенням до контейнера.

VS Code було обрано через широкий вибір розширень (в тому числі для зручного форматування коду), швидкості запуску та зручності для дрібних правок [15].

Висновки до розділу 3

У розділі 3 було обґрунтовано вибір мови програмування, системи керування базою даних, середовищ розробки мобільного застосунку та його серверної частини, а також середовища для контейнеризації серверної частини.

Інструменти обрані для роботи з програмним забезпеченням створять необхідні умови для ефективної розробки застосунку та сприятимуть досягненню, поставлених для досягнення головної мети роботи, завдань.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура програмного застосунку

Архітектуру програмного забезпечення формують чотири основні елементи: серверна частина бекенду, що відповідає за роботу з даними, мобільний застосунок фронт енд частини, що відповідає за збір даних та взаємодію з користувачем, база даних для збереження даних про користувачів та поїздки та моделі машинного навчання (рисунок 4.1).

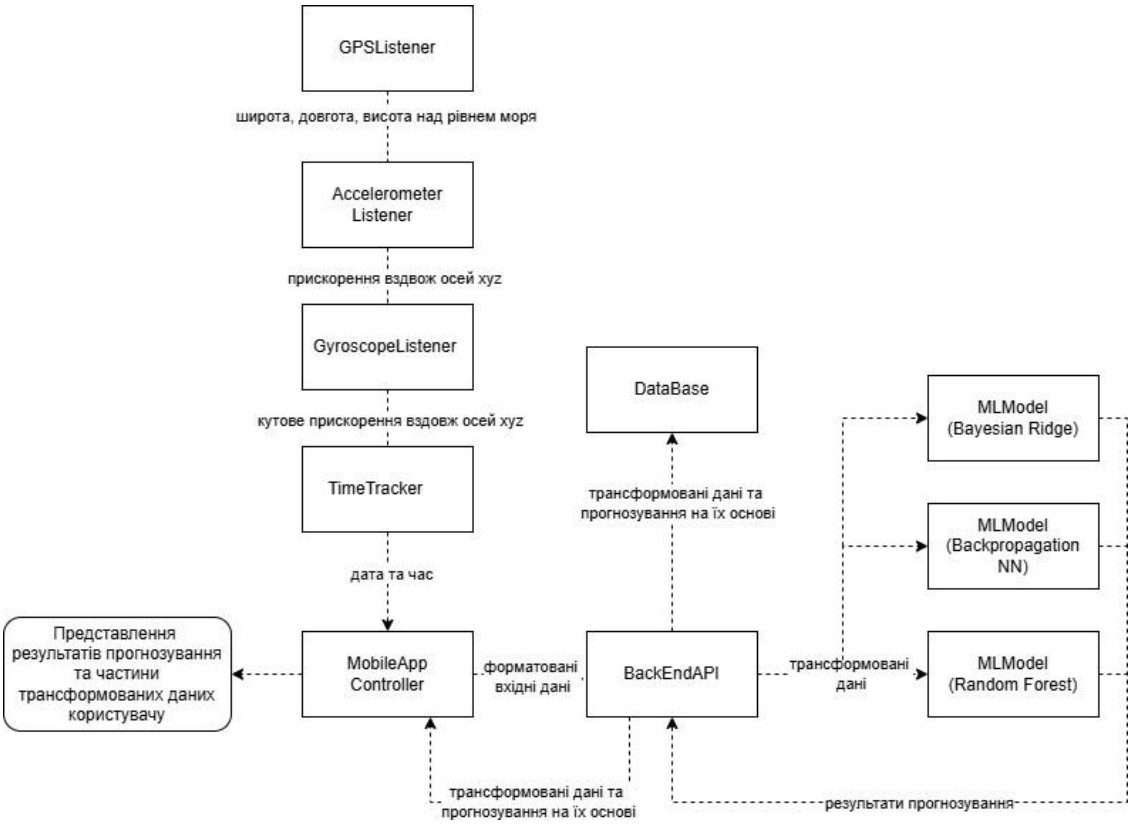


Рисунок 4.1 – Діаграма взаємодії частин системи

Моделі машинного навчання, як вже зазначалось збережені у натренованому вигляді і під час взаємодії з користувачем моделі виконують тільки роль прогнозування, до них не вноситься змін.

Натомість загальний функціонал бекенд частини сервера значно ширший.

При реєстрації або входу користувача сервер відповідає за створення токенів, адже авторизація в застосунку реалізована за допомогою збереження внутрішньо системно згенерованого JWT токена. Якщо користувач реєструється в застосунку, його інформація вноситься в базу даних асинхронного підключення.

З боку фронт енд частини користувач має змогу запустити таймер, який відраховуватиме час поїздки і водночас записуватиме дані сенсорів смартфона. Під час виконання цієї дії, в бекенд частину одразу надсилатиметься запит на створення поїздки. В базу даних заноситься лише час початку та автоматично створюється ID. Коли поїздка закінчується, користувач має змогу обрати чи зберігатиме він поїздку, чи ні, в випадку відмови від збереження з бази даних видаляється поїздка і нічого не зберігається. При виборі збереження всі дані, , які записувались впродовж поїздки, передаються зі смартфона на бекенд.

Сервер трансформує ці дані для подальшої передачі їх в моделі машинного навчання. Попередньо сервер отримує поточний час, довготу та широту з GPS-модуля смартфона, прискорення автомобіля по осях X, Y, Z та кутові прискорення по осях X, Y, Z, тобто на вході на бекенд передається одинадцять змінних:

- phone_time;
- phone_latitude;
- phone_longitude;
- phone_elevation;
- phone_speed;
- phone_x_acceleration;
- phone_y_acceleration;
- phone_z_acceleration;
- phone_x_angl_acceleration;
- phone_y_angl_acceleration;
- phone_z_angl_acceleration.

І також ID самої поїздки, для того, аби мати змогу перевірити існування

поїздки та визначити, в яку таблицю заносити дані. Опісля відбувається перетворення даних та прогнозування за ними.

Для реалізації саме прогнозування споживання в реальному часі з боку мобільного застосунку було реалізовано передачу даних з сенсорів смартфона кожні 5 секунд (що весь час записуються у фронт енд частині системи). Для даного прогнозування непотрібна передача ID поїздки, позаяк прогнозування, яке відбувається в реальному часі, не зберігається в базу даних.

Важливо зазначити, що передаються не ізольовані дані записані на кожні п’ять секунд шляху, а всі наявні дані з інтервалами 5 секунд (тобто за кожну передачу сервер отримуватиме набір даних з додатковими даними за минулі 5 секунд). Також, існує тенденція покращення точності прогнозів з наростанням кількості даних, тому для кращих початкових результатів прогнозування починається після двох хвилин поїздки.

Результати зберігаються у таблицю поїздок у базу даних, з додатковим змінними, які розраховуються для зручності у сприйнятті інформації користувачу:

— фінальний час;

— середня швидкість;

— середня швидкість без урахування нерухомого/майже нерухомого станів;

— середні прискорення та сповільнення;

— середній час поїздки;

— час cruising;

— прогнозування споживання пального на цю поїздку від усіх трьох моделей машинного навчання;

— відповідні дані з розрахунком на сто кілометрів шляху та сумарна відстань подолана за поїздку.

Окрім того, сервер відповідає за обчислення та передачу статистичних даних за всіма поїздками. Обчислення відбувається простим методом знаходження середніх значень для усіх поїздок завершених користувачем.

Архітектура сервера поділена на логічні модулі (рисунок 4.2).

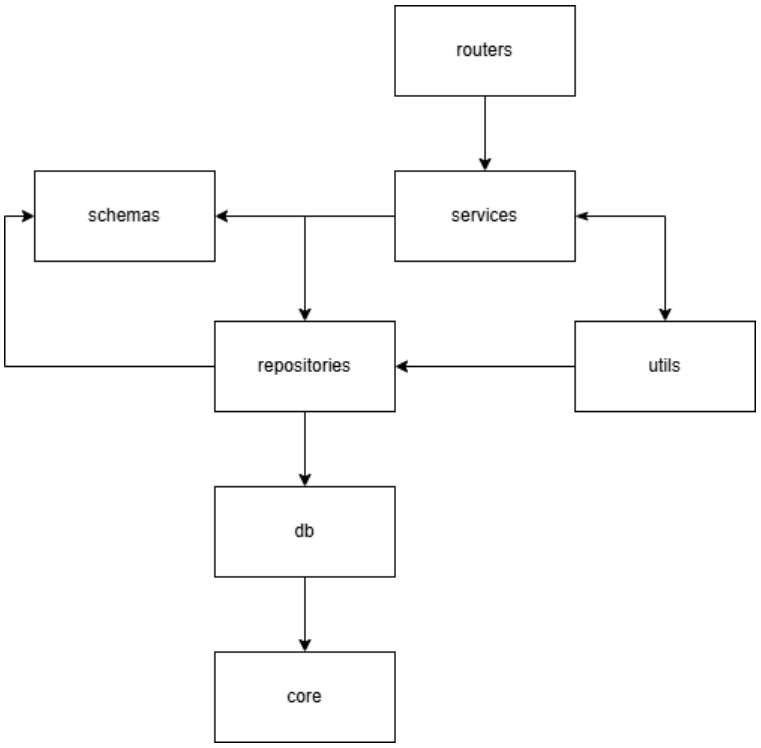


Рисунок 4.2 – Діаграма модулів архітектури сервера

Модуль core відповідає за основну конфігурацію застосунку і містить в собі код, що зберігає змінні середовища як глобальні налаштування, наприклад, параметри підключення до бази даних та особливості запуску самого серверу.

Модуль для взаємодії з базою даних db містить в собі код з ORM-моделями, які описують структуру сутностей, а також конфігурацію сесії підключення до СКБД асинхронним методом.

Шаром доступу до даних, який інкапсулює SQL-логіку є шар repositories. Всередині нього зберігається логіка базової (тобто потенційно спільної для всіх таблиць) взаємодії з таблицями та спеціалізована логіка, яка різниться залежно від потреб у функціоналі, що вимагається від кожної таблиці. Це дозволяє відділити SQL-запити від бізнес-логіки, що спрощує підтримку коду.

Шар schemas містить Pydantic-схеми, які визначають вхідні/вихідні формати даних (наприклад, запити на реєстрацію або на оновлення даних у БД). Схеми

використовуються для валідації та серіалізації/десеріалізації даних при обміні даними між сервером та користувачем.

Шар services – це шар основної логіки застосунку. В ньому відбувається проміжний етап перетворення даних, різного роду перевірок та прогнозування.

Окрім того, присутній шар маршрутизації (рутери), які викликають відповідні сервіси і передають або приймають запити.

Останній шар utils є збіркою утилітів та загальних допоміжних функцій, як рандомайзери, генератори токенів, файли pkl розширення з натренованими моделями машинного навчання тощо. Одним з ключових елементів цього шару є клас, який відповідає за внутрішньо серверну генерацію та перевірку JWT-токенів, які використовуються для авторизації користувачів. Відповідно цей шар є тією частиною програмного забезпечення, що дозволяє повторно використовувати загальну логіку у різних частинах коду, централізовано обробляти конфігурації та підвищувати загальну структурованість проєкту.

Даний підхід розподілу серверної логіки на чіткі шари сприяє створенню впорядкованої та зрозумілої структури проєкту, яка дозволяє забезпечити максимальну оптимізованість коду та уникнути повторень (рисунок 4.3).

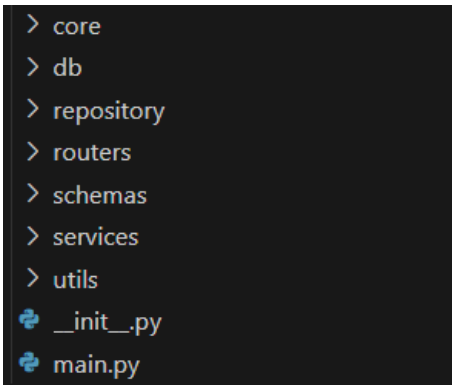


Рисунок 4.3 – Структуризація серверної частини

Підсумовуючи загальну логіку взаємодії маємо такий процес: з мобільного застосунку користувач надсилає HTTP-запит (наприклад, передає дані поїздки),

маршрутизатор (рутер) приймає запит, трансформуючи його через схеми (schemas) для подальшої роботи з даними і передає дані в сервісний шар (services), сервісна функція обробляє дані, звертається до репозиторію (repository) для роботи з БД або викликає відповідну модель машинного навчання з utils, результати передаються назад через схеми (schemas) і йдуть у відповідь користувачу.

4.2 Моделювання взаємодії між користувачем та системою

Користувач взаємодіє з системою через мобільний застосунок, який відповідає за збір даних сенсорів смартфона та надання результатів прогнозування користувачу.

Можливості користувача подані на діаграмі прецедентів (рисунок 4.4).

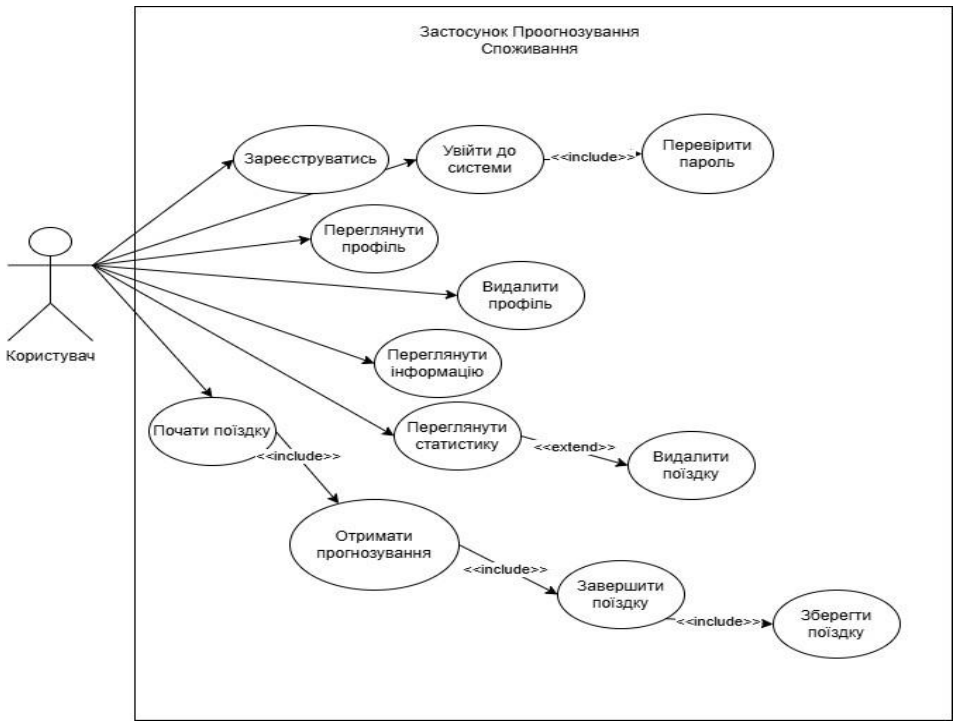


Рисунок 4.4 – Діаграма прецедентів програмного застосунку

Першою взаємодією доступною користувачу при використанні застосунку є реєстрація. Перед тим, як користувач у будь-якому вигляді (реєструючись, чи входячи в існуючий акаунт) авторизується, йому доступні лише дві сторінки, власне

входу та реєстрації. Мобільний застосунок має внутрішню перевірку валідності введення email комірки. Також для покращення застосунку з боку фронт енду одразу проводиться і перевірка на заповненість усіх комірок вводу. Лише тоді, коли електронна пошта є валідною, а всі комірки заповнені, для користувача розблоковується кнопка входу та реєстрації, яка відповідає за надсилання даних до серверної частини.

Після успішної авторизації користувачу надається доступ до повного функціоналу застосунку.

Для полегшення розуміння моделей була створена окрема сторінка з назвами та короткими поясненнями про моделі, де подані відсоткові ефективності кожної з них.

Окрім того, користувач має доступ до ще трьох сторінок. Сторінка профілю надає змогу переглянути кількість поїздок, власну інформацію, а також вийти з акаунту, або перманентно видалити його. Сторінка статистики містить середні дані усіх поїздок, а також дані кожної поїздки окремо з доступною функцією видалення поїздки за потреби. До того ж на сторінці статистики наявні графіки з порівняннями даних прогнозування на поїздку й середніх статистичних даних для кожної моделі. Відповідно графік порівняння продубльований для кожної моделі.

Ключова сторінка мобільного застосунку це сторінка поїздки, на ній теж присутні статистичні дані й таймер при запуску якого починається запис сенсорів смартфона. Позаяк для проведення прогнозування моделі потребують певний обсяг даних, безпосереднє прогнозування в режимі реального часу відбувається після двох хвилин поїздки. До перетину цього часового терміну користувач може бачити лише середні статистичні дані своїх поїздок.

Після завершення поїздки, як згадувалось раніше, користувач має вибір зберегти чи видалити поїздку, подібна система дозволить покращити не тільки безпосередньо користувацький досвід а й загальні статистичні дані, позаяк користувач матиме змогу не зберігати ті поїздки, які не є важливими безпосередньо для нього статистично.

4.3 Архітектура бази даних

Для реалізації повного функціоналу застосунку було спроєктовано реляційну базу даних зі структурою, яка враховує потреби збереження користувацьких даних, історії поїздок, результатів прогнозування споживання пального. Для реалізації подібного функціоналу було передбачено дві таблиці (рисунок 4.5).

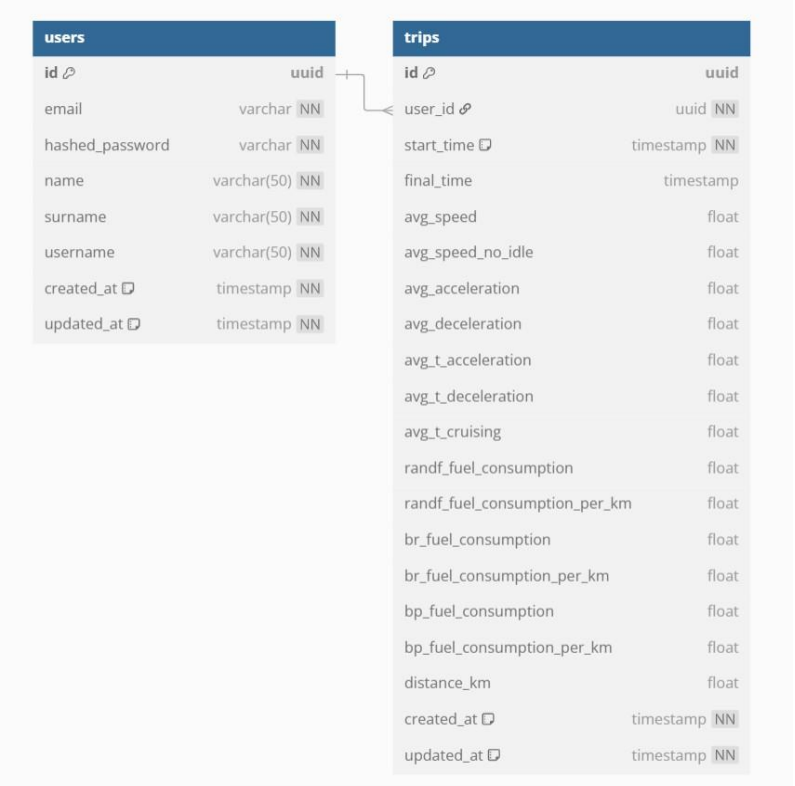


Рисунок 4.5 – Діаграма структуру БД

Для побудови схеми БД було використано ORM SQLAlchemy та зроблено міграції на основі моделей за допомогою Alembic.

В якості основного типу ідентифікаторів використано UUID (uuid4), щоб гарантувати унікальність записів навіть при розподіленому зберіганні.

Кожна таблиця містить службові поля: created_at і updated_at для контролю часу створення та останньої модифікації, що дозволяє відслідковувати життєвий цикл кожного об'єкта у базі.

Основні сутності бази даних – це таблиця користувачів User та таблиця поїздок Trip. Перша таблиця зберігає дані користувача, які використовуються для аутентифікації та ідентифікації. Друга таблиця містить зведену інформацію про кожну поїздку користувача, включаючи розраховані метрики поведінки під час руху та результати прогнозів споживання пального, отриманих з моделей машинного навчання.

Кожна поїздка належить одному користувачу, що було реалізовано через ForeignKey та relationship (в моделях).

База даних побудована за принципами нормалізованої реляційної моделі, а використання SQLAlchemy як ORM дозволило реалізовувати логіку високого рівня, мінімізуючи потребу в написанні голих SQL-запитів убезпечення системи.

Використання асинхронного підходу до роботи з базою даних (через AsyncSession) забезпечує підвищену продуктивність і масштабованість застосунку. Позаяк ефективність при обробці великої кількості одночасних запитів є критично важливою для веб-застосунків із високим навантаженням. Асинхронний підхід дозволяє ефективно використовувати ресурси серверу та забезпечити кращий зворотній зв’язок з користувачем.

4.4 Прогнозування споживання пального

У межах даної дипломної роботи програмне забезпечення реалізовується через мобільний застосунок, який збиратиме дані та передаватиме вихідні результати користувачу, взаємодіючи з сервером, що відповідатиме за трансформацію, збереження та за потреби видалення даних та прогнозування на основі цих даних через моделі машинного навчання.

Алгоритм прогнозування подано на схемі (рисунок 4.6):

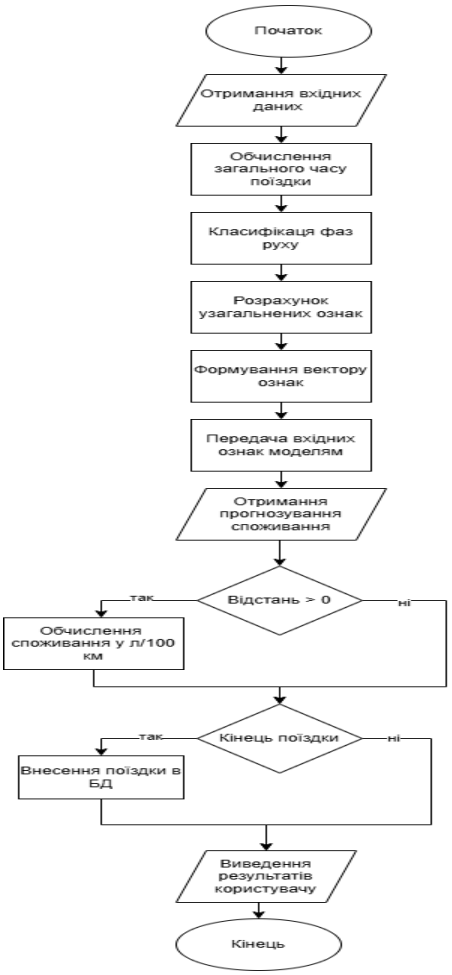


Рисунок 4.6 – Схема алгоритму прогнозування

Такими даними є:

— дата та час поїздки – фіксуються системним годинником смартфона;

— геолокаційні координати (довгота, широта) – збираються за допомогою GPS-модуля;

— висота над рівнем моря – надається GPS-модулем;

— швидкість руху автомобіля – вираховується на основі GPS-координат;

— лінійні прискорення по осях X, Y, Z – збираються за допомогою акселерометра;

— кутові прискорення (обертальні рухи) по осях X, Y, Z – вимірюються гіроскопом.

Після отримання даних користувача відбувається перетворення в дані для прогнозування, обчислюється: середня швидкість поїздки, середня швидкість без урахування простою, середнє прискорення та гальмування, відносна тривалість прискорення, гальмування та руху на постійній швидкості.

Прогнозування здійснюється за допомогою трьох моделей машинного навчання, які отримують на вхід вищевказані перетворені параметри.

Мобільний застосунок фіксує всі дані в режимі реального часу та передає їх на сервер, де виконується попередня обробка та розрахунки. Після цього прогнозовані значення споживання пального передаються у застосунок для візуалізації у вигляді таблиць та статистики.

Безпосередньо в програмному забезпеченні не відбувається визначення стилю водіння користувача. Це відбувається опосередковано виключно в моделях машинного навчання, при встановленні взаємозв’язків між певним типом гальмування/прискорення тощо, моделі визначають наскільки сильно вони впливають на споживання пального. Відповідно, користувач не матиме змоги переглянути власний стиль водіння, як одну чітку змінну, проте дані, які використовують моделі машинного навчання для прогнозування в сукупності є даними, що характеризують поведінку водія.

Проте, варто зазначити, що завдяки збереженню усіх даних (перетворених ознак, що використовуються для прогнозування) та візуалізації їх у вигляді статистики разом з кінцевим прогнозуванням споживання (на всю поїздку та на 100 км з урахуванням стилю водіння конкретної поїздки, що розглядається) у вигляді таблиці, користувач має змогу також самостійно аналізувати та визначати взаємозв’язок між своєю поведінкою на дорозі та відповідною різницею споживання порівняно з іншими його поїздками.

4.5 Тренування моделей машинного навчання

Для тренування моделей машинного навчання для застосунку потрібен великий об’єм даних. Для помірно ефективної системи було б оптимально мати дані з хоча б 200 поїздок. Й специфіка датасету, який потребує не тільки даних споживання з OBD, а й відповідні дані з сенсорів смартфона, які при навчанні створювали б вибір зіставивши час OBD та телефону, аби визначити на який момент часу припадає спільний набір даних. Тому було прийнято рішення створити генератор, який дозволить записати необхідний набір штучних, але максимально наближених до реальності даних.

Для кожної симульованої поїздки дані генеруються з частотою в одну секунду, тобто для кожної секунди генерується такий набір даних:

— швидкість (phone_speed) – розраховується з урахуванням періоду поїздки (початок, основна частина кінцівка) і не перевищує 120 км/год;

— геолокація – широта, довгота, висота (phone_latitude, phone_longitude, phone_elevation) з незначним шумом для симуляції руху в межах міста;

— прискорення – лінійне (по трьох осях) та кутове (phone_x/y/z_acceleration, phone_x/y/z_angl_acceleration);

— оберти двигуна (RPM) – розраховуються як похідна від швидкості та випадкового шуму (з врахуванням холостих обертів та певного коефіцієнту при використанні швидкості);

— поточне споживання пального (obd_consumption) – враховуються базове холосте споживання та потенційні витрати на швидкість та оберти двигуна вирахувані раніше.

На основі цих даних функція перетворення обчислює зведені ознаки (так само як це робиться безпосередньо для користувача у вже готовому застосунку):

— середня швидкість (з урахуванням ігнорування простою);

— середнє прискорення/гальмування;

— пропорція часу у фазах прискорення;

— гальмування та рівномірного руху;

— загальне споживання пального (потрібне тільки для навчання).

Функція перетворення даних виконує наступні кроки:

— визначення загального часу поїздки на основі міток часу першого та останнього запису;

— обчислення середньої швидкості на основі всіх доступних значень швидкості;

— відфільтровування простоїв (значень, де швидкість менша за поріг (0.5 м/с)), щоб обчислити середню швидкість без урахування зупинок;

— розрахунок прискорення та уповільнення між кожною парою послідовних точок на основі зміни швидкості в часі (фаза прискорення ($a > 0.1$ м/с²), фаза гальмування ($a < -0.1$ м/с²), фаза рівномірного руху ($|a| \leq 0.1$ м/с²));

— обчислення середнього прискорення та гальмування: усереднення значень прискорень у відповідних фазах;

— обчислення часу, проведеного у фазах прискорення, гальмування та рівномірного руху (відносно загального часу поїздки);

— обчислення відстані, пройденої під час поїздки, на основі формули середньої швидкості на кожному інтервалі часу ($S = \Delta t \cdot (v0 + v1)/2$).

Для кожного запуску тренування створюється вибірка розміром від 200 до 750 симульованих поїздок. Усі ці поїздки трансформуються до табличної структури, яка поділяється на вхідні ознаки X та цільову змінну Y (загальне споживання пального).

Для нейромережі зворотного поширення похибки використовується масштабування ознак через StandardScaler, оскільки модель чутлива до діапазонів значень.

Для кожної моделі було встановлено мінімальний поріг точності (R^2-метрика):

- RF ≥ 0.80;
- BR ≥ 0.80;

– BP ≥ 0.70.

У випадку збору реальних даних дозволена похибка значно зменшиться, але з урахуванням штучності даних було прийнято рішення збільшити допустиму норму похибки.

Моделі тренуються до 10 разів (з поступовим збільшенням вибірки), поки не буде досягнуто заданої точності або вичерпано кількість спроби. Лише після успішного досягнення заданої точності модель зберігається, інакше вона покидає цикл не збережена.

Також у model_scores.json зберігаються фінальні метрики точності, які передаватимуться на інформаційну сторінку мобільного застосунку.

Тренування моделей відбувається у повністю автоматизованому режимі з використанням симуляції поїздок та обчислення ознак, що відображають стиль водіння користувача. Такий підхід дозволяє підготувати функціональні моделі навіть без початкового датасету та забезпечити базову персоналізацію прогнозування споживання пального.

Тобто вхідними та вихідними даним для алгоритмів для тренування моделей будуть:

– вхідні дані: параметри генерації симульованих поїздок, список моделей (випадкові ліси, баєсова гребенева регресія та нейромежа зворотного поширення похибки);

– вихідні дані: моделі збережені в форматі (*.pkl) та файл з метриками точності (model_scores.json).

Кроки алгоритму:

– визначити мінімальні пороги точності (R^2) для кожної моделі (RF ≥ 0.80, BR ≥ 0.80, BP ≥ 0.70);

– ініціалізувати лічильник спроб (задати кількість спроб);

– повторювати тренування, поки точність менше порогу та лічильник спроб менше заданої кількості спроб;

– генерувати як вхідні дані для тренування моделей таку n-кількість наборів, щоб вона дорівнювала сумі заданої початкової кількості наборів (триста в даній роботі) та добутку спроби на заданий коефіцієнт збільшення вибірки (п’ятдесят в даній роботі);

– трансформувати поїздки у набір ознак: середні швидкості з та без простою, середні лінійні та кутові прискорення, середній час помірної (cruise) поїздки, повна кількість спожитого пального (виступає цільовою змінною);

– розділити набір на тренувальні та тестувальні дані;

– для нейромережі зворотного поширення похибки – нормалізувати вхідні дані;

– навчити моделі та обчислити R^2-метрику на тестовій вибірці;

– за умови отримання метрики, що перевищує найкращий показник точності моделі оновити найкращий показник та зберегти модель;

– збільшити спробу;

– за умови, що після циклу модель досягла порогової точності – зберегти модель у файл *.pkl, а точність у model_scores.json.

Перевагами подібного підходу є:

– гнучкість – адаптивність під різні сценарії водіння;

– автономність – не потребує зовнішнього датасету (в даному випадку це є критично важливим, позаяк отримання датасету потрібних обсягів не є можливо в межах роботи);

– персоналізація – після збору реальних поїздок можливе покращення моделі, донавчання тощо;

– безпечність – відсутність залежності від персональних даних користувача на етапі розробки.

Висновки до розділу 4

У розділі 4 надано опис архітектури та загальної структури серверної частини застосунку. Надано інформацію щодо тренування моделей машинного навчання та алгоритмів перетворення даних отриманих з сенсорів смартфону для використання їх як ознак в прогнозуванні споживання пального.

Розглянуто структуру бази даних, її сутності визначені з урахуванням потреб програмного забезпечення. Надано опис загального функціоналу мобільного застосунку з описом можливостей користувача та їхнього зв’язку з безпосередньо серверною частиною мобільного застосунку. Для кращого розуміння структури, функціоналу та взаємодій були надані діаграми прецедентів, структури бази даних, модулів архітектури сервеної частини системи та загальної взаємодії між її частинами.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

5.1 Встановлення та налаштування програмного застосунку

Першим кроком встановлення програмного забезпеченння на пристрій з операційною системою Android є завнтаження APK файлу. Після інсталяції застосунку користувач повинен буде надати дозвіл на отримання геолокаційних даних та даних з сенсорів смартфона для коректної роботи застосунку.

Варто також навести вимоги до користувацької сторони для можливості користування застосунком:

— операційна система: Android 8.0 (API level 26) або новіша;

— підтримка GPS-модуля (для отримання координат);

— доступ до сенсорів акселерометра та гіроскопа;

— доступ до мережі (Wi-Fi або мобільний інтернет);

— дозвіл на надсилання HTTP/HTTPS-запитів до сервера;

— пам’ять для зберігання тимчасових даних та обробки інформації.

При відкритті застосунку вперше, користувач отримає доступ до сторінки реєстрації та входу. Для ілюстрації загального вигляду обох сторінок наведена сторінку входу (рисунок 5.1), з погляду вигляду їх відрізняє тільки кількість комірок вводу.

Якщо користувач не увійшов у свій акаунт, то це єдині сторінки, до яких він має доступ. Після входу перша сторінка, що відкривається – це сторінка профілю користувача (рисунок 5.2).

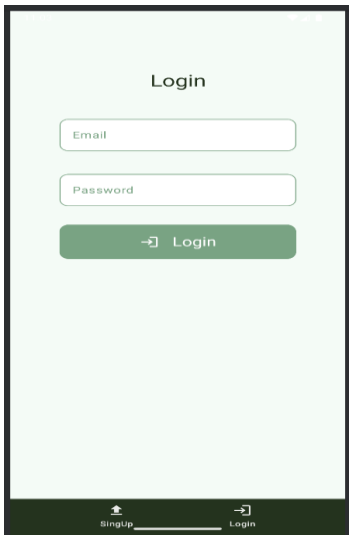


Рисунок 5.1 – Сторінка входу

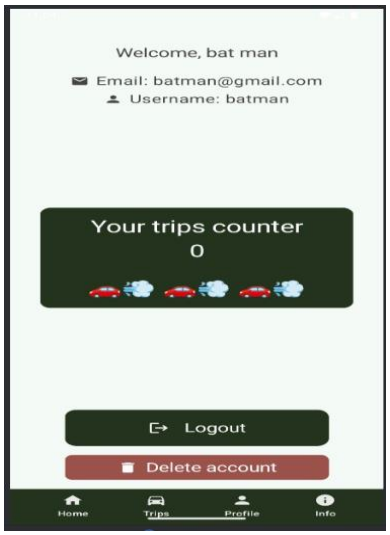


Рисунок 5.2 – Сторінка профіля

Функціонал сторінки входу не дуже широкий, позаяк не є основним. Користувач має можливість вийти з акаунту та перманентно видалити його (з цією дією з бази даних видаляються також усі поїздки прив’язані до користувача). Сторінка надає невелику довідку щодо даних користувача, а також лічильник кількості поїздок зроблених з застосунком. Потреба у створенні сторінки профілю полягала у необхідності прив’язати дані поїздок до конкретного користувача, для можливості надання статистики та списку поїздок з усіма даними та прогнозами.

Власне після входу користувач отримує доступ до низки інших сторінок. Серед них інформаційна сторінка (рисунок 5.3), з описом моделей машинного навчання, які використані в застосунку. Ця сторінка надає можливість користувачу

оцінити моделі машинного навчання з власної точки зору. При натисканні на блок з назвою моделі та її точністю розгортається інформація про неї.

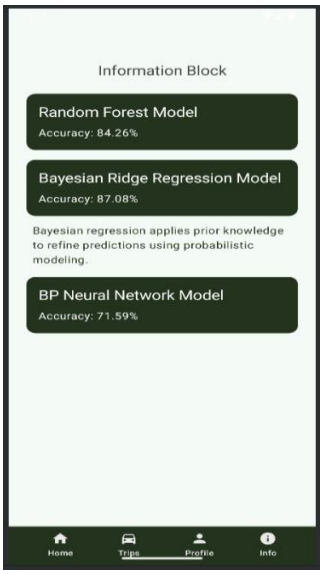


Рисунок 5.3 – Інформаційна сторінка

Важливою є сторінка зі статистичними даними. Її можна розділити на декілька основних блоків:

— статистика у вигляді числових даних: середні значення ознак, які використовуються для прогнозування споживання пального, а також середні значення самих прогнозувань (рисунок 5.4);

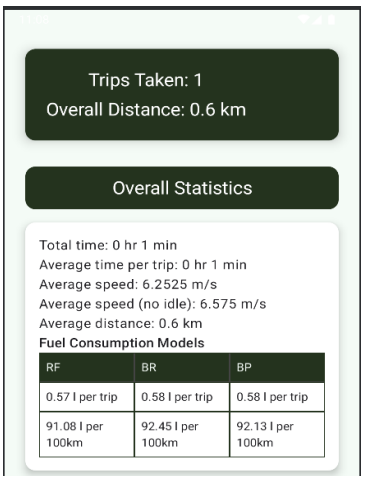


Рисунок 5.4 – Сторінка статистики, числові дані

— статистика у вигляді графіків: стовпчикові діаграми, що візуалізують різницю між середніми значеннями прогнозів споживання пального на 100 км шляху та прогнозами останніх п’яти (у випадку меншої кількості усіх наявних)

поїздок (рисунок 5.5);

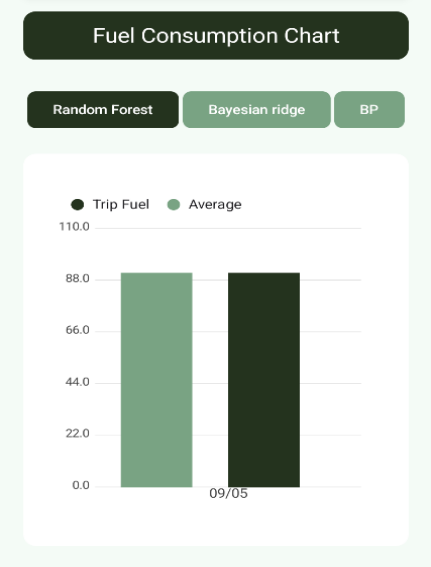


Рисунок 5.5 – Сторінка статистки, графіки

— дані про конкретні поїздки: блоки з інформацією про поїздку, перетвореними даними отриманими з поїздки та прогнозами кожної моделі оформленими у табличку (поїздки не мають найменувань і характеризуються часом та датою початку) (рисунок 5.6).

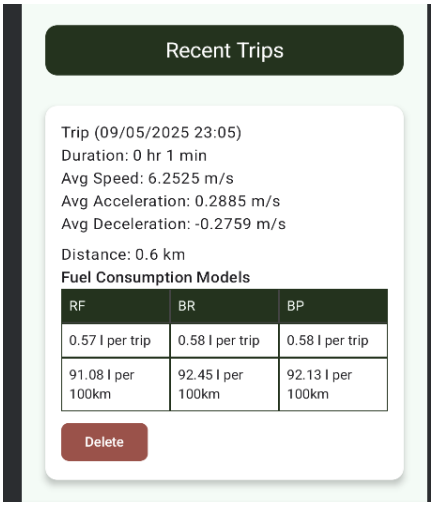


Рисунок 5.6 – Сторінка статистики, дані про поїздку

5.2 Прогнозування споживання пального

Ключовий функціональний екран застосунку – це сторінка поїздки (рисунок 5.7).

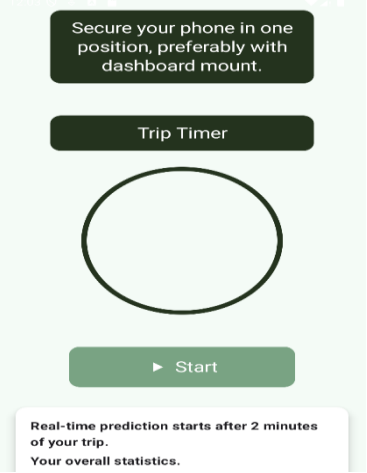


Рисунок 5.7 – Сторінка поїздки

На ній користувач отримує інструкції щодо рекомендованого положення смартфона. А також загальні статистичні дані, які доступні для перегляду перед поїздкою та в перші дві хвилини її тривалості.

Натиснувши кнопку "Старт", користувач запускає таймер і розпочинається запис поїздки й відповідно створення поїздки та передача даних до серверної частини.

Після двох хвилин поїздки починається прогнозування споживання пального в реальному часі (рисунок 5.8).

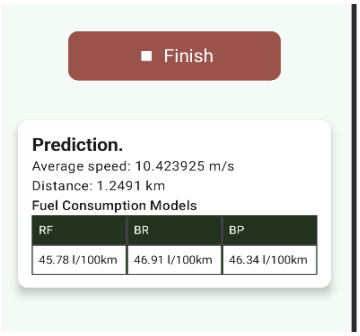


Рисунок 5.8 – Прогнозування споживання пального

Дані прогнозування оновлюються кожні п’ять секунд для того, аби моделі

машинного навчання мали актуальну інформацію й відповідно мали змогу надавати актуальний прогноз. Для легшого сприйняття, дані виводяться у вигляді таблички з трьома різними показниками відповідно до моделі машинного навчання.

Після завершення поїздки користувач натискає кнопку «Фініш» (вона замінює кнопку «Старт», коли починається поїздка), після чого з'являється вікно з підтвердженням (рисунок 5.9).

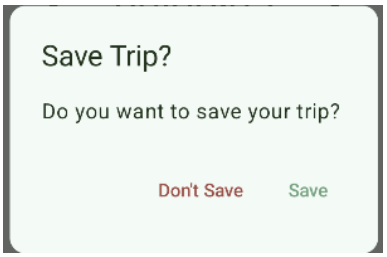


Рисунок 5.9 – Вікно підтвердження

Користувач має змогу відхилити збереження поїздки, якщо не вважає дані актуальними для себе. Незалежно від того, чи обирає користувач зберегти поїздку чи видалити (відхилити), виводиться плашка внизу екрану, що підтверджує дійсне виконання дії (рисунок 5.10).

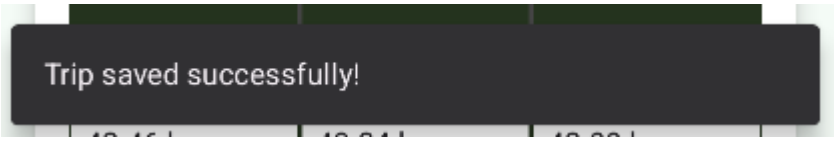


Рисунок 5.10 – Плашка про виконання дії

Отже, підсумовуючи, загальна логіка взаємодії має такий вигляд: користувач входить або реєструється, розпочинає поїздку, запускає таймер, після двох хвилин поїздки отримує реальні прогнози, завершує поїздку, приймає рішення щодо збереження, переглядає результати прогнозів на сторінці статистики, може оглянути профіль або переглянути довідку.

Висновки до розділу 5

У розділі було розглянуто та візуалізовано процеси безпосередньої взаємодії користувачем з мобільним застосунком.

Оглянуто інтерфейс застосунку та наведено його візуалізацію. Показано реалізацію серверного функціоналу у фронт енд частині. Описано загальну логіку взаємодії, можливості та обмеження користувача на кожному її етапі.

Продемонстровано, особливості реалізації інтерактивності, а саме: початок і завершення поїздки, обробка та збереження даних, оновлення прогнозу в реальному часі, для засвідчення високого ступеню інтеграції інтерфейсу з моделями машинного навчання.

Сформовано вимоги для користувача для успішного та безперебійного ефективного використання застосунку.

ВИСНОВКИ

У межах даної роботи було розроблено програмне забезпечення, яке дозволяє прогнозувати споживання пального на основі даних сенсорів смартфона за допомогою моделей машинного навчання. Отриманий застосунок надає можливість користувачам візуалізувати прогнози та порівнювати ефективність різних моделей прогнозування.

У процесі розробки було виконано всі основні завдання. Проведено аналіз існуючих методів прогнозування споживання пального з використанням моделей машинного навчання. Підготовлено дані для навчання моделей – створено функцію перетворення сирих даних (GPS, акселерометр, гіроскоп), а також реалізовано генератор штучних поїздок для формування навчального датасету у випадку відсутності реальних даних. Розроблено серверну частину, що здійснює обробку, зберігання та прогнозування Створено мобільний застосунок для Android на базі Kotlin та Jetpack Compose, що забезпечує збір сенсорних даних, взаємодію з сервером і зручну візуалізацію результатів. Реалізовано та протестовано три моделі машинного навчання: випадкові ліси, баєсова гребенева регресія, нейромережа зворотного поширення похибки.

Таким чином, проєкт демонструє можливість практичного застосування моделей машинного навчання в транспортній галузі з метою підвищення енергоефективності та контролю споживання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How Many People Own Smartphones in the World? (2024-2029) | Priori Data. Priori Data. URL: https://prioridata.com/data/smartphone-stats/ (date of access: 21.05.2025).
2. Громадська організація "Vision Zero". URL: https://visionzero.org.ua/images/cdf8b2fc9c7143a698cc6c7bece40247.pdf (дата звернення: 10.05.2025).
3. Під час війни кількість транспортних засобів в Україні зросла на третину та досягла 13 млн. Мінфін - все про фінанси: новини, курси валют, банки. URL: https://minfin.com.ua/ua/2024/03/20/123560792/ (дата звернення: 10.05.2025).
4. IBM. What Is Random Forest? |IBM. IBM – United States. URL: https://www.ibm.com/think/topics/random-forest (дата звернення: 09.05.2025).
5. ШІ КБ Л4 Нейромережи2 | Освітній портал. URL: https://learn.ztu.edu.ua/pluginfile.php/181431/mod_resource/content/1/ШІ_КБ_Л-4_Нейромережи2.pdf (дата звернення: 09.05.2025).
6. Bishop C. M. Pattern Recognition and Machine Learning. SpringerLink. URL: https://link.springer.com/book/9780387310732 (дата звернення: 20.05.2025).
7. Fuel Monitoring System for Vehicles - Ruptela. Ruptela. URL: https://ruptela.com/solutions/fuel-monitoring-control/ (дата звернення: 29.05.2025).
8. ACTIA eco-driving training assistant - Actia. Actia. URL: https://www.actia.com/en/train-your-drivers-in-eco-friendly-driving-with-actia/ (дата звернення: 29.05.2025).
9. General Python FAQ. Python documentation. URL: https://docs.python.org/3/faq/general.html (дата звернення: 12.05.2025)
10. asyncio - Asynchronous I/O. Python documentation. URL: https://docs.python.org/3/library/asyncio.html (дата звернення: 12.05.2025).
11. Features - FastAPI. FastAPI. URL: https://fastapi.tiangolo.com/features/ (дата звернення: 12.05.2025).

12. Kotlin for Android | Kotlin. Kotlin Help. URL: https://kotlinlang.org/docs/android-overview.html (дата звернення: 12.05.2025).

13. Why Compose | Jetpack Compose | Android Developers. Android Developers. URL: https://developer.android.com/develop/ui/compose/why-adopt (дата звернення: 12.05.2025).

14. PostgreSQL Advantages: Benefits of Using PostgreSQL. Prisma's Data Guide. URL: https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql (дата звернення: 12.05.2025).

15. What is Docker?. Docker Documentation. URL: https://docs.docker.com/get-started/docker-overview/ (дата звернення: 12.05.2025).

16. An Energy Efficient Smartphone Sensors' Data Fusion for High Rate Position Sampling Demands. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/8651755 (дата звернення: 21.05.2025).

17. Vehicle Fuel Consumption Prediction Method Based on Driving Behavior Data Collected from Smartphones. Biblioteca Digital de Bogotá | BibloRed. URL: https://www.bibliotecadigitaldebogota.gov.co/resources/3897252/ (дата звернення: 21.05.2025).

18. Concurrency and async / await - FastAPI. FastAPI. URL: https://fastapi.tiangolo.com/async/ (дата звернення: 21.05.2025).

19. Factor Про Податки та Бухоблік. Factor Про Податки та Бухоблік. URL: https://i.factor.ua/ukr/law-312/section-1100/article-16012/?srsltid=AfmBOorHrgtZFK7XH0GPBlRGF8MkE0QuSz2x7mxpXpLn9K4TPG7RWEkf (дата звернення: 21.05.2025).

ДОДАТОК А

Програмне забезпечення прогнозування споживання пального

Програмний код

Аркушів 20

Київ — 2025

trip_service.py

```
class TripService:
    def __init__(self, trip_repository: TripRepository):
        self.trip_repository = trip_repository
        self.randf_model = joblib.load("app/utils/randf_model.pkl")
        self.br_model = joblib.load("app/utils/br_model.pkl")
        self.bp_model = joblib.load("app/utils/bp_model.pkl")

    async def start_trip(self, user_id: UUID):
        if user_id is None:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Data wasn`t
given")
        db_trip = Trip(user_id=user_id)
        created_trip = await self.trip_repository.create(db_trip)

        return created_trip


    async def real_time_prediction(self, user_data: List[UserData], mock: bool = True):
        temp = await self.transform_user_data(user_data, mock)

        transformed, distance_m, current_time = temp[0], temp[1], temp[2]

        if current_time.tzinfo is not None:
            current_time = current_time.replace(tzinfo=None)


        input_features = [
            transformed["avg_speed"],
            transformed["avg_speed_no_idle"],
            transformed["avg_acceleration"],
            transformed["avg_deceleration"],
            transformed["avg_t_acceleration"],
            transformed["avg_t_deceleration"],
            transformed["avg_t_cruising"]
        ]
        predicted_consumption = [0, 0, 0]
        predicted_consumption[0] = self.randf_model.predict([input_features])[0]
        predicted_consumption[1] = self.br_model.predict([input_features])[0]
        predicted_consumption[2] = self.bp_model.predict([input_features])[0]

        l_per_100km = [0, 0, 0]

        if distance_m > 0:
            l_per_100km[0] = predicted_consumption[0] / (distance_m/1000) * 100
            l_per_100km[1] = predicted_consumption[1] / (distance_m/1000) * 100
            l_per_100km[2] = predicted_consumption[2] / (distance_m/1000) * 100
```

```
        return {"avg_speed": transformed["avg_speed"], "distance": round(distance_m/1000, 4),
"rand_f": l_per_100km[0], "br": l_per_100km[1], "bp": l_per_100km[2]}



    async def finish_trip(self, trip_id: UUID, user_data: List[UserData], mock: bool = True):
        trip = await self.trip_repository.get_one(trip_id)
        if trip is None:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Trip not
found")


        temp = await self.transform_user_data(user_data, mock)

        transformed, distance_m, current_time = temp[0], temp[1], temp[2]

        if current_time.tzinfo is not None:
            current_time = current_time.replace(tzinfo=None)


        input_features = [
            transformed["avg_speed"],
            transformed["avg_speed_no_idle"],
            transformed["avg_acceleration"],
            transformed["avg_deceleration"],
            transformed["avg_t_acceleration"],
            transformed["avg_t_deceleration"],
            transformed["avg_t_cruising"]
        ]
        predicted_consumption = [0, 0, 0]
        predicted_consumption[0] = self.randf_model.predict([input_features])[0]
        predicted_consumption[1] = self.br_model.predict([input_features])[0]
        predicted_consumption[2] = self.bp_model.predict([input_features])[0]

        l_per_100km = [0, 0, 0]

        if distance_m > 0:
            l_per_100km[0] = predicted_consumption[0] / (distance_m/1000) * 100
            l_per_100km[1] = predicted_consumption[1] / (distance_m/1000) * 100
            l_per_100km[2] = predicted_consumption[2] / (distance_m/1000) * 100

        updated_data = TripUpdateRequest(**{
            "final_time": current_time,
            "avg_speed": round(transformed["avg_speed"], 4),
            "avg_speed_no_idle": round(transformed["avg_speed_no_idle"], 4),
            "avg_acceleration": round(transformed["avg_acceleration"], 4),
            "avg_deceleration": round(transformed["avg_deceleration"], 4),
            "avg_t_acceleration": round(transformed["avg_t_acceleration"], 4),
            "avg_t_deceleration": round(transformed["avg_t_deceleration"], 4),
            "avg_t_cruising": round(transformed["avg_t_cruising"], 4),

            "randf_fuel_consumption": round(predicted_consumption[0], 4),
            "randf_fuel_consumption_per_km": round(l_per_100km[0], 4),
```

```
        "br_fuel_consumption": round(predicted_consumption[1], 4),
        "br_fuel_consumption_per_km": round(l_per_100km[1], 4),

        "bp_fuel_consumption": round(predicted_consumption[2], 4),
        "bp_fuel_consumption_per_km": round(l_per_100km[2], 4),

        "distance_km": round(distance_m/1000, 4),
    })

    updated_trip = await self.trip_repository.update(trip, updated_data)

    return updated_trip


async def check_trip_status(self, trip_id: UUID):
    trip = await self.trip_repository.get_one(trip_id)
    if not trip.final_time:
        return {"finished": False}
    return {"finished": True}

async def transform_user_data(self, data: List[UserData], mock: bool = True) -> dict:
    if not data or len(data) < 2:
        raise ValueError("Insufficient data for transformation.")
    start_time = data[0].phone_time
    end_time = data[-1].phone_time

    if start_time.tzinfo is not None and end_time.tzinfo is None:
        start_time = start_time.replace(tzinfo=None)
    elif start_time.tzinfo is None and end_time.tzinfo is not None:
        end_time = end_time.replace(tzinfo=None)

    total_time = (end_time - start_time).total_seconds()

    if mock:
        mockData = await self.generate_mock_trip(len(data))

        time_step = total_time / (len(data) - 1)

        for i in range(len(data)):
            data[i].phone_latitude = mockData[i]["phone_latitude"]
            data[i].phone_longitude = mockData[i]["phone_longitude"]
            data[i].phone_elevation = mockData[i]["phone_elevation"]
            data[i].phone_speed = mockData[i]["phone_speed"]
            data[i].phone_x_acceleration = mockData[i]["phone_x_acceleration"]
            data[i].phone_y_acceleration = mockData[i]["phone_y_acceleration"]
            data[i].phone_z_acceleration = mockData[i]["phone_z_acceleration"]
            data[i].phone_x_angl_acceleration = mockData[i]["phone_x_angl_acceleration"]
            data[i].phone_y_angl_acceleration = mockData[i]["phone_y_angl_acceleration"]
            data[i].phone_z_angl_acceleration = mockData[i]["phone_z_angl_acceleration"]

            data[i].phone_time = start_time + timedelta(seconds=i * time_step)
```

```
    idle_threshold = 0.5
    distance_m = 0

    acceleration_times = deceleration_times = cruising_times = 0.0
    accel_values = []
    decel_values = []


    speeds = [d.phone_speed for d in data]                    # м/с
    speeds_no_idle = [s for s in speeds if s > idle_threshold]

    for i in range(1, len(data)):


        dt = (data[i].phone_time - data[i - 1].phone_time).total_seconds()
        if dt == 0:
            continue

        v0, v1 = data[i - 1].phone_speed, data[i].phone_speed
        a = (v1 - v0) / dt

        distance_m += 0.5 * (v0 + v1) * dt


        if abs(v0) < idle_threshold and abs(v1) < idle_threshold:
            pass
        elif a > 0.1:
            acceleration_times += dt
            accel_values.append(a)
        elif a < -0.1:
            deceleration_times += dt
            decel_values.append(a)
        else:
            cruising_times += dt

    return [{
        "avg_speed": np.mean(speeds),                         # м/с
        "avg_speed_no_idle": np.mean(speeds_no_idle) if speeds_no_idle else 0,  # м/с
        "avg_acceleration": np.mean(accel_values) if accel_values else 0,
        "avg_deceleration": np.mean(decel_values) if decel_values else 0,
        "avg_t_acceleration": acceleration_times / total_time,
        "avg_t_deceleration": deceleration_times / total_time,
        "avg_t_cruising": cruising_times / total_time,
    }, distance_m, data[-1].phone_time]


async def get_user_trips(self, user_id: UUID):
    trips = await self.trip_repository.get_trips_by_user(user_id)
    return trips


async def delete_trip(self, trip_id: UUID):
```

```
        trip = await self.trip_repository.get_one(trip_id)
        if trip is None:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="trip not
found")
        await self.trip_repository.delete(trip)


    async def get_user_statistics(self, user_id: UUID):
        trips = await self.get_user_trips(user_id)
        trips = list(trips)
        avg_trips_data = {
            "overall_time": 0, #в хвилинах
            "avg_time": 0, #в хвилинах
            "avg_speed": 0,
            "avg_speed_no_idle": 0,
            "avg_acceleration": 0,
            "avg_deceleration": 0,
            "avg_t_acceleration": 0,
            "avg_t_deceleration": 0,
            "avg_t_cruising": 0,

            "randf_fuel_consumption": 0,
            "randf_fuel_consumption_per_km": 0,

            "br_fuel_consumption": 0,
            "br_fuel_consumption_per_km": 0,

            "bp_fuel_consumption": 0,
            "bp_fuel_consumption_per_km": 0,

            "trip_counter": 0,
            "avg_distance_km": 0,
            "overall_distance_km": 0,
        }

        for trip in trips:
            avg_trips_data["trip_counter"] += 1
            time_diff = trip.final_time - trip.start_time
            avg_trips_data["avg_time"] += time_diff.total_seconds()/60
            avg_trips_data["overall_time"] += time_diff.total_seconds()/60
            avg_trips_data["avg_speed"] += trip.avg_speed
            avg_trips_data["avg_speed_no_idle"] += trip.avg_speed_no_idle
            avg_trips_data["avg_acceleration"] += trip.avg_acceleration
            avg_trips_data["avg_deceleration"] += trip.avg_deceleration
            avg_trips_data["avg_t_acceleration"] += trip.avg_t_acceleration
            avg_trips_data["avg_t_deceleration"] += trip.avg_t_deceleration
            avg_trips_data["avg_t_cruising"] += trip.avg_t_cruising

            avg_trips_data["randf_fuel_consumption"] += trip.randf_fuel_consumption
            avg_trips_data["randf_fuel_consumption_per_km"] +=
trip.randf_fuel_consumption_per_km

            avg_trips_data["br_fuel_consumption"] += trip.br_fuel_consumption
```

```
            avg_trips_data["br_fuel_consumption_per_km"] += trip.br_fuel_consumption_per_km

            avg_trips_data["bp_fuel_consumption"] += trip.bp_fuel_consumption
            avg_trips_data["bp_fuel_consumption_per_km"] += trip.bp_fuel_consumption_per_km

            avg_trips_data["avg_distance_km"] += trip.distance_km
            avg_trips_data["overall_distance_km"] += trip.distance_km

        if avg_trips_data["trip_counter"] != 0:
            for key in avg_trips_data:
                if key != "trip_counter" and  key != "overall_time" and key != "overall_distance_km":
                    avg_trips_data[key] /= avg_trips_data["trip_counter"]
                if key != "trip_counter" :
                    avg_trips_data[key] = round(avg_trips_data[key], 4)
        return avg_trips_data

    async def generate_mock_trip(self, n_points: int = 900):
        trip = []
        current_speed = 0.0

        for i in range(n_points):
            if i < n_points * 0.2:
                current_speed += random.uniform(0.1, 0.8)
            elif i > n_points * 0.8:
                current_speed -= random.uniform(0.1, 0.8)
            else:
                current_speed += random.uniform(-0.2, 0.2)

            current_speed = max(0, min(current_speed, 33))  # м/с

            trip.append({
                "phone_latitude": 41.0 + random.uniform(-0.01, 0.01),
                "phone_longitude": 29.0 + random.uniform(-0.01, 0.01),
                "phone_elevation": random.uniform(0, 50),
                "phone_speed": current_speed,
                "phone_x_acceleration": random.uniform(-2, 2),
                "phone_y_acceleration": random.uniform(-2, 2),
                "phone_z_acceleration": random.uniform(-2, 2),
                "phone_x_angl_acceleration": random.uniform(-5, 5),
                "phone_y_angl_acceleration": random.uniform(-5, 5),
                "phone_z_angl_acceleration": random.uniform(-5, 5)
            })

        return trip


    async def get_model_accuracy(self):
        with open('app/utils/model_scores.json', 'r') as file:
            model_accuracy = json.load(file)

        return model_accuracy
```

synthetic_fuel_training.py

```
def generate_dummy_trip(duration_sec=900, start_time=None):
    if not start_time:
        start_time = datetime.now()
    trip = []
    current_speed = 0.0
    for i in range(duration_sec):
        timestamp = start_time + timedelta(seconds=i)

        if i < duration_sec * 0.2:#початок поїздки
            current_speed += random.uniform(0.1, 0.8)
        elif i > duration_sec * 0.8:#кінець поїздки
            current_speed -= random.uniform(0.1, 0.8)
        else:#середина поїздки
            current_speed += random.uniform(-0.2, 0.2)
        current_speed = max(0, min(current_speed, 33))  # 120 км/год максимум
        #холості оберти плюс швидкість й коефіцієнт, шуми
        rpm = 800 + current_speed * 50 + random.uniform(-100, 100)
        #холоста витрата, витрата по швидкості, витрата по обертам
        fuel_rate = 0.5 + 0.02 * current_speed + 0.001 * rpm
        fuel_rate = min(fuel_rate, 2.5)  # л/г
        trip.append({
            "obd_time": timestamp.isoformat(),
            "obd_consumption": fuel_rate,
            "phone_time": timestamp.isoformat(),
            "phone_latitude": 41.0 + random.uniform(-0.01, 0.01),
            "phone_longitude": 29.0 + random.uniform(-0.01, 0.01),
            "phone_elevation": random.uniform(0, 50),#по місту
            "phone_speed": current_speed,
            "phone_x_acceleration": random.uniform(-2, 2),
            "phone_y_acceleration": random.uniform(-2, 2),
            "phone_z_acceleration": random.uniform(-2, 2),
            "phone_x_angl_acceleration": random.uniform(-5, 5),
            "phone_y_angl_acceleration": random.uniform(-5, 5),
            "phone_z_angl_acceleration": random.uniform(-5, 5)
        })
    return trip


def transform_trip_data(trip_data: list[dict]) -> dict:
    phone_speeds = [point["phone_speed"] for point in trip_data]
    avg_speed = sum(phone_speeds) / len(phone_speeds)
    avg_speed_no_idle = sum(s for s in phone_speeds if s > 1) / max(len([s for s in phone_speeds if s
> 1]), 1)
    accelerations = [(phone_speeds[i] - phone_speeds[i - 1]) for i in range(1, len(phone_speeds))]

    accel_values = [a for a in accelerations if a > 0.1]
    decel_values = [a for a in accelerations if a < -0.1]

    avg_acceleration = sum(accel_values) / max(len(accel_values), 1)
    avg_deceleration = sum(decel_values) / max(len(decel_values), 1)

    accel_time = len([a for a in accelerations if a > 0.1])
    decel_time = len([a for a in accelerations if a < -0.1])
```

```
    cruise_time = len([a for a in accelerations if abs(a) <= 0.1 and a >= -0.1])
    total_time = len(accelerations)
    avg_t_acceleration = accel_time / total_time
    avg_t_deceleration = decel_time / total_time
    avg_t_cruising = cruise_time / total_time
    fuel_consumptions = [point["obd_consumption"] / 3600 for point in trip_data]
    total_fuel = sum(fuel_consumptions)
    return {
        "avg_speed": avg_speed,
        "avg_speed_no_idle": avg_speed_no_idle,
        "avg_acceleration": avg_acceleration,
        "avg_deceleration": avg_deceleration,
        "avg_t_acceleration": avg_t_acceleration,
        "avg_t_deceleration": avg_t_deceleration,
        "avg_t_cruising": avg_t_cruising,
        "fuel_consumption": total_fuel
    }


def generate_training_dataset(n_samples=300):
    return pd.DataFrame([transform_trip_data(generate_dummy_trip()) for _ in range(n_samples)])


def train_all_models():
    min_scores = {
        "RF": 0.80,
        "BR": 0.80,
        "BP": 0.70
    }

    models = {
        "RF": lambda: RandomForestRegressor(n_estimators=50, random_state=42),
        "BR": lambda: BayesianRidge(),
        "BP": lambda: MLPRegressor(hidden_layer_sizes=(5,), activation="tanh", solver="lbfgs",
max_iter=1000, random_state=42)
    }

    scores = {}

    for name, model_builder in models.items():
        best_score = -np.inf
        best_model = None
        retries = 0

        while best_score < min_scores[name] and retries < 10:
            dataset_size = 300 + retries * 50
            print(f"\nTraining {name} (attempt {retries + 1}) with {dataset_size} samples...")

            df = generate_training_dataset(n_samples=dataset_size)
            X = df.drop(columns=["fuel_consumption"])
            y = df["fuel_consumption"]
```

```
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)
            X_test_scaled = scaler.transform(X_test)

            model = model_builder()
            if name in ["BP", "BR"]:
                model.fit(X_train_scaled, y_train)
                score = model.score(X_test_scaled, y_test)
            else:
                model.fit(X_train, y_train)
                score = model.score(X_test, y_test)

            if score > best_score:
                best_score = score
                best_model = model

            retries += 1

        if best_score >= min_scores[name]:
            joblib.dump(best_model, f"{name.lower()}_model.pkl")
            scores[name] = round(best_score, 4)
            print(f"{name} R² Score: {best_score:.4f}")
        else:
            print(f"{name} didn`t reach R² ≥ {min_scores[name]}. Best: {best_score:.4f}")

    if scores:
        with open("model_scores.json", "w") as f:
            json.dump(scores, f, indent=2)
    else:
        print("\nNothing saved.")


if __name__ == "__main__":
    train_all_models()
```

PhoneSensorManager.kt

```
data class PhoneSensorData(
    val latitude: Double,
    val longitude: Double,
    val elevation: Double,
    val speed: Float,
    val xAcc: Float,
    val yAcc: Float,
    val zAcc: Float,
    val xAngAcc: Float,
    val yAngAcc: Float,
    val zAngAcc: Float
)

object PhoneSensorManager : SensorEventListener {
```

```
    private var sensorManager: SensorManager? = null
    private var fusedLocationClient: FusedLocationProviderClient? = null

    private var latestAccelerometer = FloatArray(3)
    private var latestGyroscope = FloatArray(3)
    private var isRegistered = false

    fun initialize(appContext: Context) {
        if (sensorManager == null && fusedLocationClient == null) {
            val appCtx = appContext.applicationContext // SAFE
            sensorManager = appCtx.getSystemService(Context.SENSOR_SERVICE) as
SensorManager
            fusedLocationClient = LocationServices.getFusedLocationProviderClient(appCtx)
        }
    }

    fun register() {
        if (!isRegistered) {
            sensorManager?.registerListener(
                this,
                sensorManager?.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_NORMAL
            )
            sensorManager?.registerListener(
                this,
                sensorManager?.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
                SensorManager.SENSOR_DELAY_NORMAL
            )
            isRegistered = true
        }
    }

    fun unregister() {
        if (isRegistered) {
            sensorManager?.unregisterListener(this)
            isRegistered = false
        }
    }

    override fun onSensorChanged(event: SensorEvent?) {
        event?.let {
            when (it.sensor.type) {
                Sensor.TYPE_ACCELEROMETER -> latestAccelerometer = it.values.clone()
                Sensor.TYPE_GYROSCOPE -> latestGyroscope = it.values.clone()
            }
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}

    @SuppressLint("MissingPermission")
    suspend fun getSensorData(): PhoneSensorData {
```

```
        val location = getLastKnownLocation()

        return PhoneSensorData(
            latitude = location?.latitude ?: 0.0,
            longitude = location?.longitude ?: 0.0,
            elevation = location?.altitude ?: 0.0,
            speed = location?.speed ?: 0f,
            xAcc = latestAccelerometer.getOrNull(0) ?: 0f,
            yAcc = latestAccelerometer.getOrNull(1) ?: 0f,
            zAcc = latestAccelerometer.getOrNull(2) ?: 0f,
            xAngAcc = latestGyroscope.getOrNull(0) ?: 0f,
            yAngAcc = latestGyroscope.getOrNull(1) ?: 0f,
            zAngAcc = latestGyroscope.getOrNull(2) ?: 0f
        )
    }

    @SuppressLint("MissingPermission")
    private suspend fun getLastKnownLocation(): Location? = suspendCancellableCoroutine { cont -
>
        fusedLocationClient?.lastLocation
            ?.addOnSuccessListener { location -> cont.resume(location) }
            ?.addOnFailureListener { cont.resume(null) }
    }
}
```

TripViewModel.kt

```
class TripViewModel(
    private val tripPreferences: TripPreferences,
    private val apiService: ApiService,
    private val token: String,
) : ViewModel() {

    var tripId: UUID? by mutableStateOf(null)
    var tripStarted by mutableStateOf(false)
    var tripStartTime by mutableStateOf<Instant?>(null)

    var realTimePrediction by mutableStateOf<TripPredictedRealTime?>(null)
    private var predictionJob: Job? = null


    val elapsedTime: State<Long> = derivedStateOf {
        tripStartTime?.let {
            Duration.between(it, Instant.now()).seconds
        } ?: 0L
    }

    private val _tripDataPoints = mutableStateListOf<TripDataPoint>()
    val tripDataPoints: List<TripDataPoint> get() = _tripDataPoints

    private var sensorJob: Job? = null

    init {
```

```
    viewModelScope.launch {
        Log.d("TripViewModel", "Initializing ViewModel")
        tripStarted = tripPreferences.tripStartedFlow.first()
        if (tripStarted) {
            tripPreferences.tripIdFlow.first()?.let {
                tripId = UUID.fromString(it)
            }
            tripStartTime = tripPreferences.getTripStartTime()
            startSensorCollection()
        }
    }
}

var statistics by mutableStateOf<TripStats?>(null)
    private set

private var hasLoadedStats = false

fun loadStatistics() {
    if (hasLoadedStats) return
    hasLoadedStats = true

    viewModelScope.launch {
        try {
            statistics = apiService.getStatistics("Bearer $token")
        } catch (e: Exception) {
            Log.e("Statistics", "Failed to load stats", e)
        }
    }
}



private fun startPredictionUpdates() {
    predictionJob = viewModelScope.launch {
        while ((tripStartTime?.let { Duration.between(it, Instant.now()).seconds } ?: 0L) < 120) {
            delay(1000)
        }

        while (tripStarted) {
            try {
                if (_tripDataPoints.isNotEmpty()) {
                    val response = apiService.realTime(_tripDataPoints)
                    realTimePrediction = response
                }
            } catch (e: Exception) {
                Log.e("PredictionUpdate", "Error sending prediction request", e)
            }
            delay(5000)
        }
    }
}
```

```
fun clearPrediction() {
    realTimePrediction = null
}


fun startTrip() {
    viewModelScope.launch {
        Log.d("TripDataToSend", "Trip started")
        val authHeader = "Bearer $token"
        val trip = apiService.startTrip(authHeader)
        tripId = trip.id
        tripStarted = true
        tripStartTime = Instant.now()

        tripPreferences.saveTripState(trip.id, true, tripStartTime!!)

        PhoneSensorManager.register()
        startSensorCollection()
        startPredictionUpdates()

    }
}

fun finishTrip() {
    viewModelScope.launch {
        Log.d("beforesend", "Final trip data: $tripDataPoints")
        sendTripDataToBackend()

        sensorJob?.cancel()
        sensorJob = null
        PhoneSensorManager.unregister()

        tripPreferences.clearTripState()
        tripId = null
        tripStarted = false
        tripStartTime = null
        _tripDataPoints.clear()
        predictionJob?.cancel()
        predictionJob = null

    }
}

fun deleteTrip() {
    viewModelScope.launch {
        try {
            tripId?.let { id ->
                apiService.deleteTrip(id)
            }
        } catch (e: Exception) {
            e.printStackTrace()
        } finally {
            sensorJob?.cancel()
```

```
            sensorJob = null
            PhoneSensorManager.unregister()
            tripPreferences.clearTripState()
            tripId = null
            tripStarted = false
            tripStartTime = null
            _tripDataPoints.clear()
            predictionJob?.cancel()
            predictionJob = null

        }
    }
}

private fun startSensorCollection() {
    val formatter = DateTimeFormatter.ISO_INSTANT

    sensorJob = viewModelScope.launch {
        Log.d("SensorCollection", "Sensor collection started")

        while (tripStarted) {
            val now = Instant.now()
            val sensorData = getSensorData()
            val phoneTime = formatter.format(now)

            _tripDataPoints.add(
                TripDataPoint(
                    phone_time = phoneTime,
                    phone_latitude = sensorData.latitude,
                    phone_longitude = sensorData.longitude,
                    phone_elevation = sensorData.elevation,
                    phone_speed = sensorData.speed,
                    phone_x_acceleration = sensorData.xAcc,
                    phone_y_acceleration = sensorData.yAcc,
                    phone_z_acceleration = sensorData.zAcc,
                    phone_x_angl_acceleration = sensorData.xAngAcc,
                    phone_y_angl_acceleration = sensorData.yAngAcc,
                    phone_z_angl_acceleration = sensorData.zAngAcc
                )
            )

            Log.d("SensorCollection", "Data point count: ${_tripDataPoints.size}")
            delay(1000)
        }
    }
}

private suspend fun getSensorData(): PhoneSensorData {
    val data = PhoneSensorManager.getSensorData()
    Log.d("SensorData", "Collected sensor data: $data")
    return data
}
```

```
    private suspend fun sendTripDataToBackend() {
        try {
            tripId?.let { id ->
                Log.d("TripDataToSend", "Final trip data: $tripDataPoints")
                apiService.finishTrip(id, tripDataPoints)
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
```

ДОДАТОК Б

Програмне забезпечення прогнозування споживання пального

Презентація

Аркушів 7

Київ — 2025

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

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

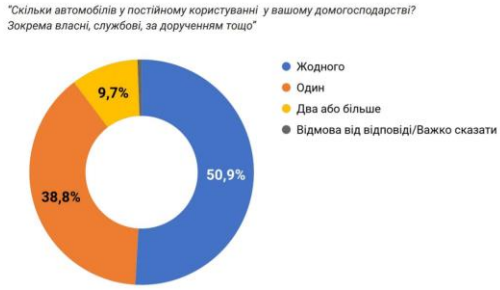
Програмне забезпечення прогнозування споживання пального

Гундяк Валерія Русланівна, ТВ-11
Доцент, к.е.н., Гусєва І. І.

Київ – 2025

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

Актуальність теми



Станом на 19 березня 2024 року, за даними Єдиного державного реєстру транспортних засобів, на обліку в Україні перебувало 14,16 млн транспортних засобів, з яких 9,8 млн — легкових.

В умовах зростання цін на пальне, а також потреби у зменшенні шкідливих викидів в атмосферу, особливої вагомості набуває точний моніторинг і прогнозування споживання палива. Технології штучного інтелекту мають можливість зробити вагомий внесок у підвищення енергоефективності та скорочення витрат у транспортній галузі, а також покращення безпосередньо досвіду водія завдяки можливості більш точно моніторити дані щодо поїздок.

Постановка завдання

Метою роботи є створення програмного забезпечення для прогнозування споживання пального з використанням моделей машинного навчання на основі даних сенсорів смартфона.

Завдання, що потребують вирішення для досягнення мети:

1. провести аналіз існуючих методів прогнозування споживання пального за допомогою моделей машинного навчання;
2. дослідити та підготувати відповідні дані для навчання моделей;
3. розробити архітектуру системи;
4. розробити базу даних системи;
5. розробити мобільний застосунок зі збору та обробки даних з сенсорів смартфона;
6. розробити алгоритми перетворення даних та прогнозування споживання;
7. реалізувати та протестувати декілька моделей машинного навчання для прогнозування витрат пального;
8. створити інтерфейс для візуалізації прогнозів та статистики поїздок;
9. забезпечити функціонал для збереження, перегляду та видалення поїздок.

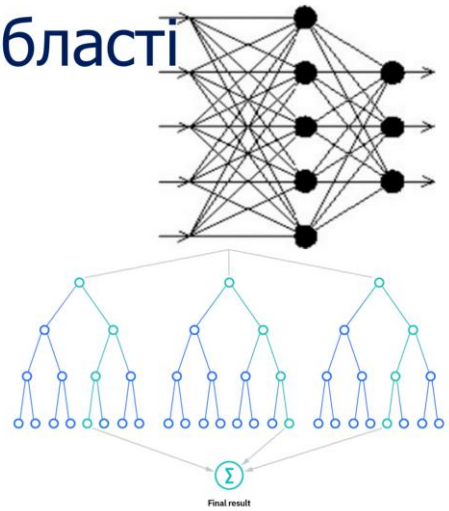
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Аналіз предметної області

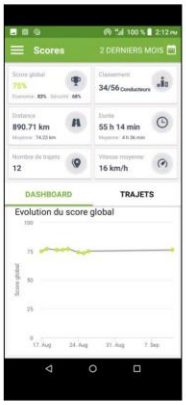
Предметна область охоплює аналіз і прогнозування споживання пального автомобіля на основі персоналізованих даних користувача, що збираються його смартфоном під час поїздки без збереження для прогнозування в реальному часі та після завершення в повному обсязі передаються на сервер для внесення фінальних даних та прогнозів у базу даних.

У сфері інтелектуального транспорту застосовуються різні моделі прогнозування, як класичні статистичні методи так й складніші моделі машинного навчання (дерева рішень, нейронні мережі тощо). Ключовими критеріями ефективності прогнозної моделі є точність, стійкість до шуму у даних та здатність адаптуватися до індивідуального стилю водіння.

Використання прогнозування споживання пального дозволяє покращити економність водіння та покращити досвід водія.



Опис аналогічних систем

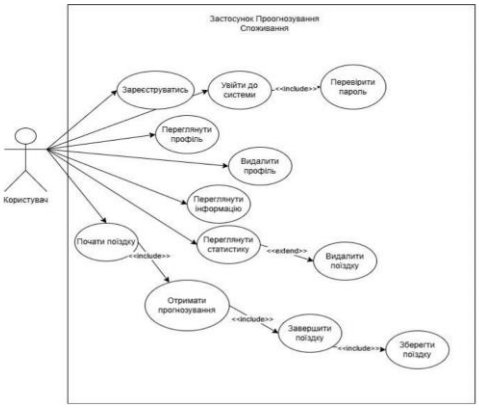


EcoDrive ACTIA

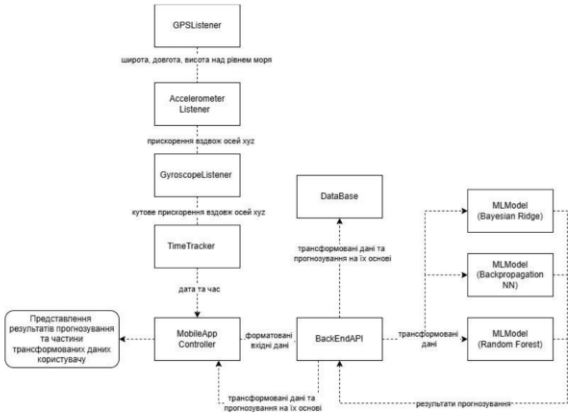
EcoDrive Ruptela

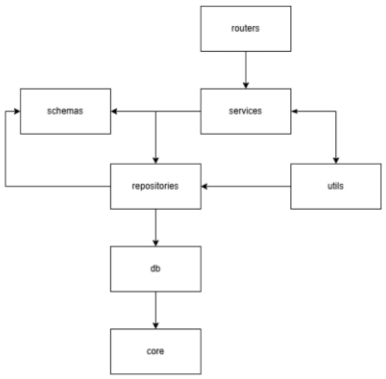
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Діаграма прецедентів



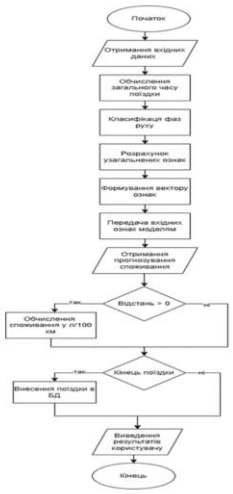
Архітектура системи





База даних

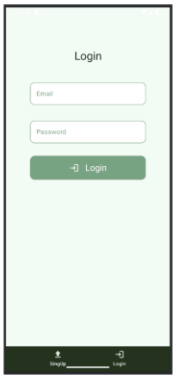
Алгоритм прогнозування споживання пального

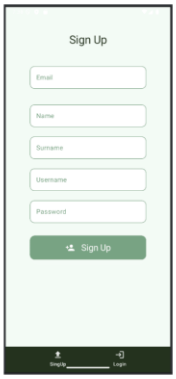


НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

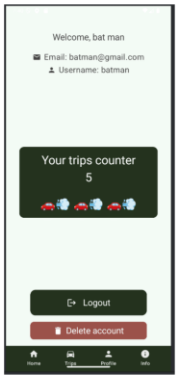
Засоби розробки

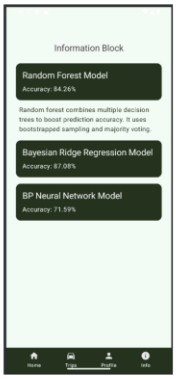
Інсталяція та налаштування





Сторінки авторизації

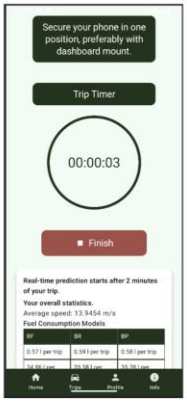




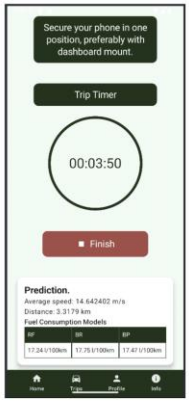
Сторінки доступні після авторизації

Сторінка поїздки

До початку поїздки



Перші дві хвилини поїздки

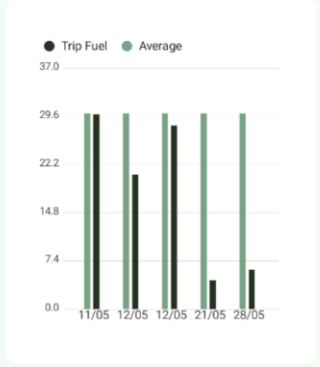


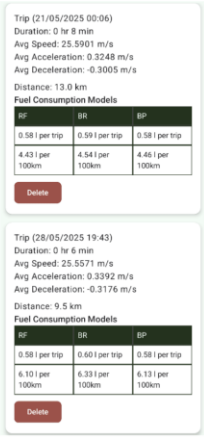
Після двох хвилин поїздки

Тестування


```
{
  "RF": 0.8426,
  "BR": 0.8708,
  "BP": 0.7159
}
```


Точність моделей





Результати тестування прогнозування

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

Висновки

У межах даної роботи було розроблено програмне забезпечення, яке дозволяє прогнозувати витрати пального на основі даних сенсорів смартфона за допомогою моделей машинного навчання. Отриманий застосунок надає можливість користувачам візуалізувати прогнози та порівнювати ефективність різних моделей прогнозування.

У процесі розробки було виконано всі основні завдання. Проведено аналіз існуючих методів прогнозування витрат пального з використанням моделей машинного навчання. Підготовлено дані для навчання моделей – створено функцію перетворення сирих даних (GPS, акселерометр, гіроскоп), а також реалізовано генератор штучних поїздок для формування навчального датасету у випадку відсутності реальних даних. Реалізовано архітектуру сервер-клієнтної системи. Розроблено серверну частину, що здійснює обробку, зберігання та прогнозування Створено мобільний застосунок для Android на базі Kotlin та Jetpack Compose, що забезпечує збір сенсорних даних, взаємодію з сервером і зручну візуалізацію результатів. Реалізовано та протестовано три моделі машинного навчання: випадкові ліси, баєсова гребенева регресія, нейромережа зворотного поширення похибки. Створено інтерфейс користувача, що відображає прогнози споживання пального, графіки порівняння поїздок та загальну статистику.

Забезпечено функціонал збереження, перегляду та видалення поїздок – реалізовано з використанням бази даних PostgreSQL та асинхронної обробки запитів у серверній частині.

Таким чином, проєкт демонструє можливість практичного застосування моделей машинного навчання в транспортній галузі з метою підвищення енергоефективності та контролю витрат.