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. Завідувач кафедри
_____ Олександр Коваль
«___»_____202_ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

спеціальності 121 Інженерія програмного забезпечення
на тему: «Інструментальні засоби формування рекомендацій енергоефективного використання автокондиціонера»

Виконав:
студент IV курсу, групи ТІ-92
Яринич Віталій Павлович _____
(прізвище, ім'я, по батькові) (підпис)

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

Рецензент:
Доцент, к.т.н., Сірий О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ — 2023

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 і веб-технологій»

ЗАТВЕРДЖУЮ
в.о. завідувач кафедри
_____ Олександр Коваль
(підпис)
«___»_____202_ р.

ЗАВДАННЯ
на дипломну роботу студенту

Яринич Віталій Павлович
(прізвище, ім’я, по батькові)

1. Тема роботи
Інструментальні засоби формування рекомендацій енергоефективного використання автокондиціонера
керівник роботи Гусєва Ірина Ігорівна, к.е.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від «29»травня 2023 р. №2039-с

2. Строк подання студентом роботи 12.06.2023
3. Вихідні дані до роботи інструментальні засоби формування рекомендацій енергоефективного використання автокондиціонера, що складаються із мобільного застосунку, розробленого у середовищі Android Studio мовою програмування Dart з фреймворком Flutter, сервера та тестового сервісу, що розроблених у середовищі IntelliJ IDEA мовою програмування Java з використанням фрейморку Spring
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити мобільний застосунок та сервер для формування рекомендацій енергоефективного використання автокондиціонера; розробити тестовий сервіс, що надасть змогу протестувати роботу застосунку.

5. Перелік ілюстративного матеріалу Архітектура системи, Концептуальна модель роботи системи, Алгоритм роботи застосунку, Взаємодія даних, Модуль сторінки із інформацією про машину

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «30» вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	30.09.2022	Виконано
2.	Дослідження предметної області	02.10.2023 – 16.04.2023	Виконано
3.	Розробка архітектури системи	17.04.2023 – 23.04.2023	Виконано
4.	Програмна реалізація системи	24.04.2023 – 14.05.2023	Виконано
5.	Тестування та внесення корективів	15.05.2023 – 21.05.2023	Виконано
6.	Захист програмного продукту	15.05.2023 – 19.05.2023	Виконано
7.	Оформлення пояснювальної записки	22.05.2023 – 04.06.2023	Виконано
8.	Передзахист	05.06.2023 – 11.06.2023	Виконано
9.	Захист	19.06.2023 – 23.06.2023	Виконано

Студент _____ Яринич В.П.
(підпис) (прізвище та ініціали,)

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

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 54 сторінки, 28 рисунків, 1 таблиці, 2 додатків та 25 посилань.

Метою роботи є реалізація мобільного застосунку для відстеження значень температури і вологості всередині та ззовні автомобіля, та формування рекомендацій, які допоможуть енергоефективно налаштувати та використовувати автокондиціонер відповідно до наявних умов та бажаної температури у салоні.

Для досягнення поставленої мети виконано такі завдання:

1. зібрано інформацію про автокондиціонери та їх режими роботи;
2. досліджено умови для використання режимів роботи автокондиціонера;
3. розроблено алгоритм формування рекомендацій для енергоефективного використання автокондиціонера.

Розроблено мобільний застосунок для збору потрібної інформації та виведення сформованих рекомендацій, з урахуванням умов, при яких застосунком будуть користуватися, сервер для проведення необхідного аналізу, та сервіс імітації стану автомобіля для тестування застосунку.

Практичне значення одержаних результатів полягає у створенні програмного застосунку, що допоможе водіям отримувати рекомендації щодо енергоефективного використання автокондиціонера у режимі реального часу.

Ключові слова: енергоефективність, рекомендації, мобільний застосунок, автокондиціонер, режим роботи, погодні умови, автомобіль.

ABSTRACT

The structure and scope of the diploma thesis. The thesis consists of 54 pages, 28 figures, 1 tables, 2 appendices, and 25 references.

The purpose of the thesis is to implement a mobile application for monitoring the temperature and humidity values inside and outside the vehicle, and generating recommendations that will help optimize the car air conditioning efficiently according to the prevailing conditions and desired temperature in the cabin.

To achieve these goals, the following tasks were completed:

1. collected information about car air conditioning systems and their operating modes;
2. ivestigated the conditions for utilizing the operating modes of the car's air conditioning system;
3. developed an algorithm for generating recommendations for energy-efficient use of the car's air conditioning system.

A mobile application has been developed to collect the necessary information and output the formed recommendations, taking into account the conditions under which the application will be used, a server for providing the necessary analysis, and a vehicle simulation service for testing the application.

The practical significance of the obtained results lies in the creation of a software application that assists drivers in receiving real-time recommendations for energy-efficient usage of the car's air conditioning system.

Keywords: energy efficiency, recommendations, mobile application, car air conditioning, operating mode, weather conditions, car.

ЗМІСТ

ВСТУП

Автомобільний кондиціонер - це необхідний елемент сучасного автомобіля, який дозволяє комфортно перебувати в машині, коригуючи температуру в салоні відносно побажань пасажирів. Однак, використання кондиціонера може значно збільшити споживання палива автомобілем (у середньому на 10-20%), що не тільки негативно впливає на середовище, але й може стати великою фінансовою витратою для власника транспортного засобу. Тому, енергоефективне використання кондиціонера стає все більш актуальною темою, особливо з погляду екологічної та економічної ситуації зараз.

Підтримка кондиціонера у хорошому технічному стані – це, безумовно, необхідний крок для досягнення задачі про енергоефективне використання кондиціонера. Використання навіть повністю справного автокондиціонера може бути дуже економічно та енергетично затратним. Вся справа у його налаштуваннях. Не всі водії знають, при яких умовах потрібно вмикати той чи інший режим автокондиціонера, а якщо і знають, то не завжди цим користуються, адже самому аналізувати температуру, вологість та їх відношення ззовні та всередині салону, особливо під час водіння, майже неможливо.

Виникає питання розробки інструментальних засобів, які б могли автоматизовано зчитувати необхідну інформацію з сенсорів автомобіля, та на їх основі формувати рекомендації для оптимальних налаштувань автокондиціонера, потребуючи мінімального залучення водія у роботу, але при цьому підвищуючи комфорт при перебуванні у салоні автомобіля.

Таким чином постає необхідність розробки саме мобільного застосунку для вирішення даної проблеми, адже для сучасного користувача є більш звичним використовувати програмне забезпечення саме через мобільні телефони. Окрім того, практика використання смартфонів під час поїздки є дуже поширеною серед водіїв, і є найбільш зручним варіантом для використання будь-якого програмного забезпечення під час процесу водіння.

Тому об’єктом дослідження даної роботи є шляхи формування рекомендацій, спрямованих на стимулювання енергоефективного використання систем кондиціонування повітря автомобіля та їх імплементація у мобільний застосунок. Дослідження передбачає вивчення існуючих технологій і методів, що використовуються в автокондиціонерах, для визначення потенційних областей для їх вдосконалення та можливого енергозбереження. Методи збору та аналізу даних, використовуються в режимі реального часу.

Дослідження також розглядає екологічні наслідки енергоефективного кондиціонування повітря в автомобілях, спрямованих на зменшення викидів вуглецю та сприяння екологічному транспорту. Увагу також було приділено на виявленні та аналізі різних компонентів і функцій автомобільних систем кондиціонування повітря, щоб зрозуміти ідеальні для них умови використання та їхній вплив на енергоефективність.

1 ЗАДАЧА ФОРМУВАННЯ РЕКОМЕНДАЦІЙ ЕНЕРГОЕФЕКТИВНОГО ВИКОРИСТАННЯ АВТОКОНДИЦІОНЕРА

Метою роботи є розробка застосунку, для енергоефективного використання кондиціонера.

Для успішної реалізації необхідно розв’язати наступні задачі:

1) дослідити предметну область та визначити основні особливості, розробки системи;
2) розробити алгоритм формування рекомендацій з енергоефективного використання автокондиціонера;
3) розробити архітектуру системи;
4) реалізувати функцію реєстрації та авторизації користувача;
5) розробити тестовий сервіс, який допоможе протестувати застосунок у наближених до реальності умовах, та виконати тестування;

Мобільний застосунок має наступні вимоги:

1) можливість зчитувати та виводити інформацію у режимі реального часу;
2) забезпечувати зручну та швидку взаємодію з користувачем, для можливості використання застосунку під час водіння;
3) можливість переглянути інформацію про кондиціонер, на основі якої проводиться аналіз та формування рекомендацій;
4) можливість оновити рекомендації за потреби користувача.

На етапі проектуванні системи ключовим фактором є визначити вихідні та вхідні дані системи, тобто необхідно визначити, який результат роботи системи повинен бути отриманий у кінці розробки, та яку інформацію потрібно буде передати у застосунок для цього.

Вхідними даними для даної системи, є стан кондиціонера автомобіля, зчитана інформація із сенсорів автомобіля, а саме значення температури та вологості, та

дані про сам автомобіль, а саме: класифікація автомобілю, модель, марку, витрату палива.

Вихідними даними будуть рекомендації для користувача, що допоможуть енергоефективно використовувати автокондиціонер, розрахунок витрати палива і часу для охолодження салону до відповідних значень, та при рекомендованих налаштуваннях.

Користувачами системи є водії або пасажири автомобілів.

Висновки до розділу 1

У розділі було вказано мету для даної роботи. Були наведені завдання, які потрібно виконати для успішної реалізації системи. Були виокремлені вимоги до розроблюваного мобільного застосунку. Також були визначені вхідні дані, які необхідні для повноцінного функціонування системи, вихідні дані, для розуміння результату, та користувачів даної системи. Було коротко описано предметну область, що досліджується в роботі.

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ АНАЛОГІЧНИХ СИСТЕМ

2.1 Принцип функціонування автокондиціонеру

Автокондиціонер - це система, яка встановлюється в автомобілі для регулювання температури, вологості та циркуляції повітря в салоні. Вона забезпечує комфортні умови для пасажирів, охолоджуючи або нагріваючи повітря в залежності від потреб. Автокондиціонери також можуть фільтрувати повітря, щоб зменшити концентрацію пилу, алергенів та інших забруднень у салоні автомобіля.

Кондиціонер складається з компресора, конденсатора, евапоратора та різних клапанів та датчиків. Компресор стискає холодоагент (зазвичай фреон), який потім перетікає через конденсатор та знижує температуру, перетворюючись з газоподібного стану у рідкий. Рідкий холодоагент потім проходить через евапоратор, де він знову перетворюється на газоподібний стан, збільшуючи при цьому об'єм і знижуючи температуру, і забезпечує охолодження повітря, яке потім надходить до салону.

Найчастіше автокондиціонери мають кілька стандартних функцій [1]:

1) кондиціонування;
2) вентиляція;
3) обігрів;
4) демістинг;
5) рециркуляція;
6) кондиціонування та вентиляція;
7) обігрів та вентиляція.

При використанні кожного з даних режимів витрата палива автомобіля збільшується. У таблиці 2.1 наведено середнє збільшення витрати палива при

кожному з наведених режимів.

Таблиця 2.1 – Режими кондиціонера та їх вплив на витрату палива

Режими кондиціонера	Витрата палива (відносно вимкненого кондиціонера)
Кондиціонування	На 15% більше
Вентиляція	На 5% більше
Обігрів	На 10% більше
Демістинг	На 12% більше
Рециркуляція	На 5% більше
Кондиціонування + вентиляція	На 20% більше
Обігрів + вентиляція	На 15% більше

Також важливими функціями, що можуть вплинути на енергоефективність автокондиціонера є встановлена температура роботи, до якої потрібно охолодити повітря в салоні та його потужність. Потрібно перевіряти чи увімкнений кондиціонер, що допоможе більш коректно надавати рекомендації.

Зібравши усю необхідну інформацію для створення основного алгоритму, що стосуються кондиціонера, потрібно визначити, які дані необхідно отримувати від машини та про машину, для майбутнього розрахунку часу охолодження повітря в салоні та енергоефективності використання тих чи інших налаштувань кондиціонера при даних погодних факторах та ситуації у салоні [2].

Для цього необхідно зчитувати інформацію з сенсорів автомобіля, що вимірюють температуру в салоні та на вулиці. Це мінімум даних, яких вистачить для формування рекомендацій.

Усю згадану вище інформацію можна збирати автоматизовано через мобільний застосунок. Зчитати інформацію можна через протокол OBD-II, що зможе передавати інформацію застосунку через WI-FI. Таким чином ми отримаємо інформацію про стан кондиціонера та дані з сенсорів.

Щоб мінімізувати навантаження на протокол, що не є дуже швидким, потрібно зменшити дані, що він передає. Виходом може бути введення користувачем певних даних про машину, які не будуть змінюватися з часом:

витрати палива, модель, класифікацію автомобіля, тип двигуна та розмір приблизний розмір авто для розрахунку кількості повітря у ньому. Паралельно з цими даними, для зручності, користувач буде мати змогу створити та користуватися власним акаунтом, щоб прив'язати до нього свої машини, щоб не вводити дані кожного разу. Для ще більшої зручності, акаунт у застосунку можна буде замінити на акаунт Google.

2.2 Аналіз аналогічних систем

FuelEconomy.gov - це застосунок, розроблений для надання інформації та порад щодо енергоефективного використання різних систем автомобіля для економії палива, у тому числі і для автокондиціонера. Це зручний та безкоштовний застосунок, призначений для допомоги водіям у виборі енергоефективних автомобілів та оптимізації використання палива [3].

Користувачі можуть знайти докладну інформацію про споживання палива різних автомобілів, включаючи середнє споживання у місті та на трасі, а також комбіноване споживання. Інтерфейс для пошуку інформації та різних моделей та марок машин зображено на рисунку 2.1.

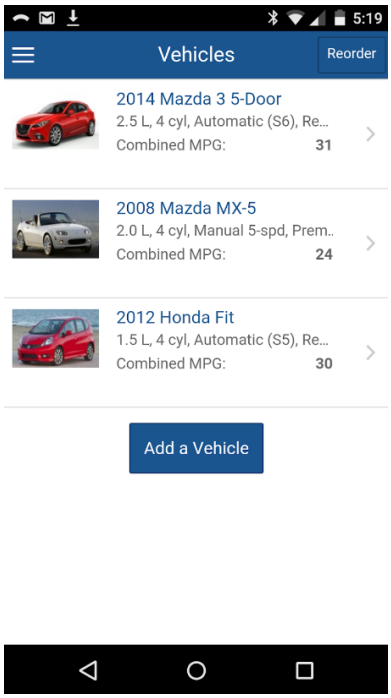

Рисунок 2.1 – Список машин у застосунку FuelEconomy.gov

Якщо не вдається знайти потрібну марку та модель автомобіля, застосунок має функціонал для того, щоб користувач зміг додати її у базу даних. Даний функціоналі зображено на рисунку 2.2.

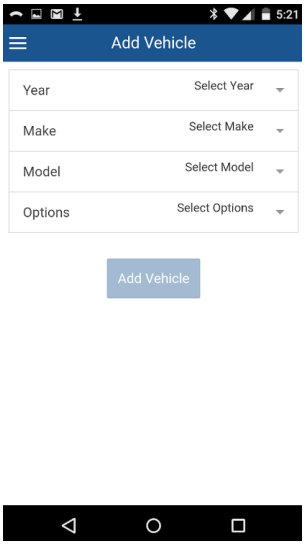

Рисунок 2.2 – Сторінка додавання автомобіля у базу

Застосунок дозволяє порівнювати різні моделі автомобілів за показниками енергоефективності, що допомагає зробити обґрунтований вибір при покупці нового транспортного засобу. Сторінку інформації про певну модель та марку автомобіля зображено на рисунку 2.3.

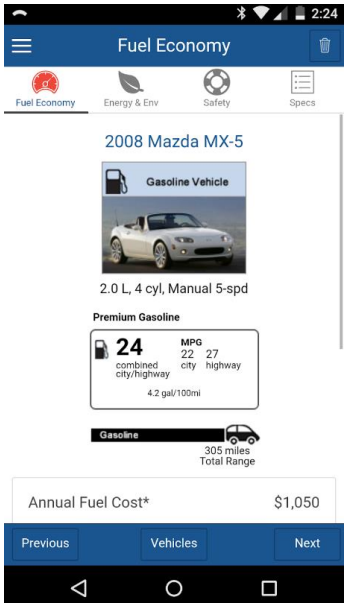

Рисунок 2.3 – Сторінка інформації про автомобіль

Застосунок надає корисні поради та рекомендації щодо ефективного використання палива, такі як правильне зберігання автомобіля, планування маршрутів та розумне використання гальм та педалі акселератора. FuelEconomy.gov має функцію визначення витрат палива на підставі особистих умов водіння користувача, включаючи тип дороги, середню швидкість та використання кондиціонера. Користувач може переглядати та порівнювати енергоефективність різних моделей автомобільних шин та знаходити оптимальний варіант для свого автомобіля.

Застосунок надає детальну інформацію про вартість палива та розраховує прогнозовані витрати на паливо для автомобіля. Користувач може вести журнал споживання палива, щоб відстежувати його ефективність і виявляти потенційні проблеми з автомобілем або стилем водіння. Сторінку розрахунків для витрат на паливо зображено на рисунку 2.4.

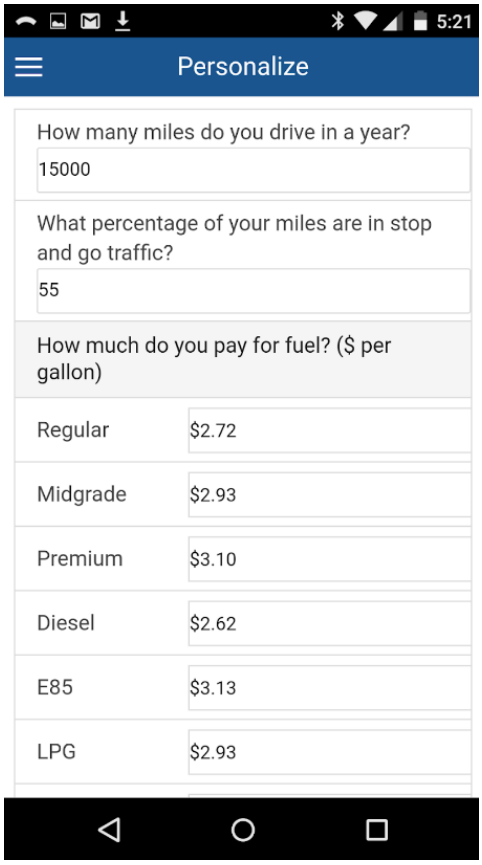

Рисунок 2.4 – Сторінка аналізу та розрахунку витрат

Застосунок має розширені можливості пошуку заправних станцій, включаючи ціни на паливо та відгуки користувачів. FuelEconomy.gov надає доступ до інформації про державні програми США та знижки на енергоефективні автомобілі, які можуть допомогти вам зекономити гроші при покупці нового авто. Застосунок також містить корисні поради щодо технік водіння, що сприяють економному споживанню палива, такі як уникання різких прискорень і гальмувань.

Застосунок має інтуїтивно зрозумілий інтерфейс, який дозволяє швидко знаходити необхідну інформацію та використовувати функції застосунку. FuelEconomy.gov є офіційним джерелом інформації від Агентства з охорони довкілля США, що забезпечує надійність та достовірність даних.

Одним з недоліків застосунку є обмежене охоплення моделей автомобілів, особливо якщо шукати інформацію про старіші або менш популярні моделі [4].

Застосунок надає рекомендації для енергоефективного використання автокондиціонеру, але вони є достатньо загальними:

1) застосунок рекомендує відкривати вікна на низькій швидкості, особливо під час поїздки містом.
2) застосунок нагадує користувачеві, що використання автокондиціонера при низькій швидкості або під час зупинки автомобіля може збільшити споживання палива.
3) застосунок надає поради щодо регулярного обслуговування автокондиціонера, включаючи перевірку рівня холодильного середовища та очищення фільтрів;
4) застосунок надає поради про вимкнення автокондиціонера перед тим як заглушити двигун.

Недоліком є те, що застосунок не надає поради для кожної моделі автомобілів у реальному часі.

Meu Volkswagen - це застосунок, розроблений компанією Volkswagen для підтримки енергоефективного використання автомобіля та оптимізації його

екологічних показників. Застосунок надає користувачам інформацію про енергоефективність їхнього автомобіля та способи її покращення [5].

Застосунок пропонує персоналізовані поради щодо оптимального використання автокондиціонера з урахуванням зовнішньої температури, вологості повітря та інших факторів.

Застосунок відображає інформацію про витрати палива в реальному часі, дозволяючи відстежувати вплив використання автокондиціонера на споживання палива. Сторінка аналізу поїздки, після підключення до машини зображено на рисунку 2.5.

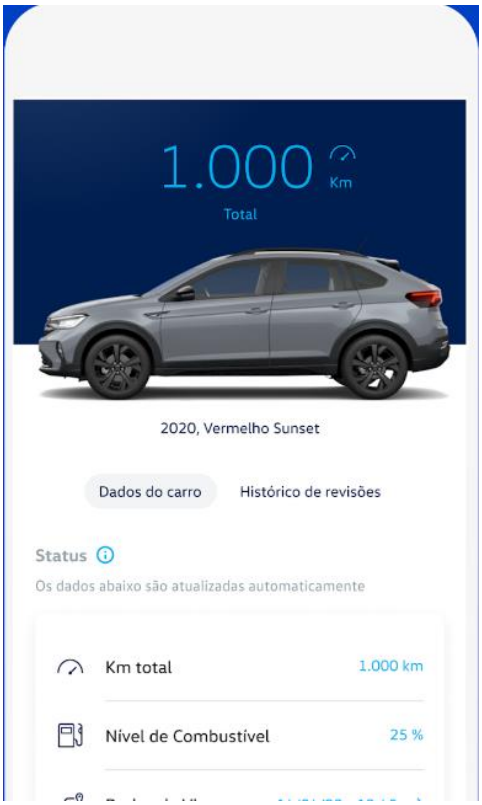

Рисунок 2.5 – Сторінка аналізу поїздки

Meu Volkswagen надає можливість створення й зберігання журналу подорожей зі споживанням палива, включаючи дані про використання автокондиціонера. Користувач може отримати персоналізовані поради щодо енергоефективного водіння та енергоефективного використання автокондиціонера, що сприяє зменшенню споживання палива та зниженню викидів шкідливих

речовин.

Застосунок надає інформацію про мережу зарядних станцій для електромобілів та допомагає знаходити найближчу доступну станцію для зарядки. Meu Volkswagen має вбудовану систему нагадувань про регулярне обслуговування автомобіля, включаючи перевірку системи кондиціонування повітря.

Застосунок дозволяє переглядати статистику поїздки, включаючи споживання палива та викиди CO2, що допомагає оцінити екологічні досягнення. Сторінка аналізу викидів зображено на рисунку 2.6.

Рисунок 2.6 – Сторінка аналізу шкідливих викидів

Meu Volkswagen надає доступ до актуальної інформації про нові моделі автомобілів Volkswagen.

Застосунок зберігає персональні налаштування та дані, що дозволяє користувачеві зручно використовувати його на різних пристроях, підтримуючи

мультиплатформенну синхронізацію

Meu Volkswagen сприяє зростанню свідомості про енергоефективне використання автокондиціонера та його вплив на довкілля.

Недоліком застосунку є обмеженість використання, адже застосунок підходить лише для автомобілів Volkswagen, що обмежує доступність використання застосунку для власників автомобілів інших марок, а також використання іспанської як основної мови інтерфейсу, що є менш поширеною ніж англійська, і, відповідно, скорочує кількість потенційних користувачів, і є найпоширенішою причиною поганих відгуків про застосунок.

Як показує низький рейтинг застосунку в Google Play та коментарі користувачів, застосунок також має безліч помилок під час роботи та є недостатньо відлагодженим [6].

Недоліком є те, що застосунок не надає поради для кожної моделі автомобілів у реальному часі.

Drivvo - це застосунок, призначений для ведення журналу автомобіля та керування витратами на паливо. Він дозволяє користувачам вести детальний журнал подорожей, включаючи інформацію про використання автокондиціонера [7].

Користувач можете налаштувати застосунок для нагадування про вимкнення автокондиціонера перед вимиканням двигуна, що допомагає знизити споживання палива та екологічний вплив.

Користувачі можуть налаштувати Drivvo для відображення інформації про споживання палива та викиди CO2 для роботи кожного модуля, у тому числі пов'язані з використанням автокондиціонера.

Застосунок надає можливість внесення даних про витрати палива та викиди CO2, формує звіти з наданої інформації, що дозволяє користувачам контролювати споживання палива та, відповідно власний екологічний вплив при водінні. Приклад звіту зображено на рисунку 2.7.

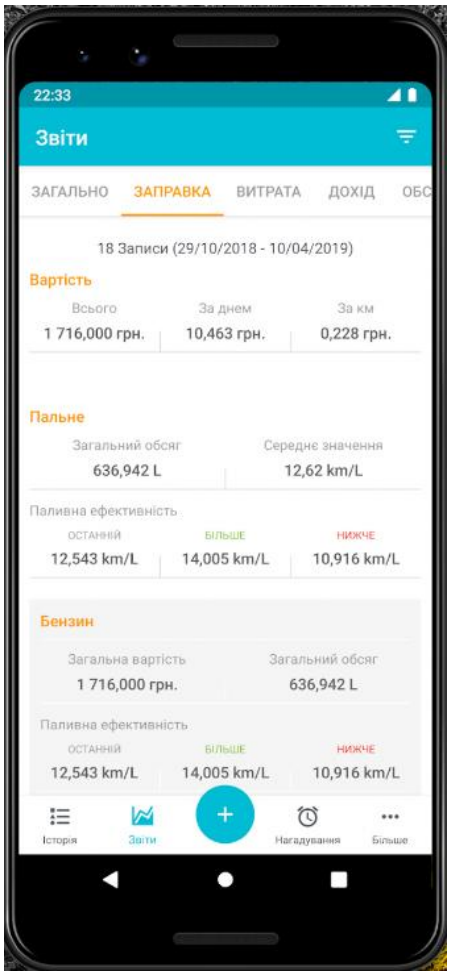

Рисунок 2.7 – Звіт витрат за заправку

Drivvo пропонує можливість встановлення цілей економії палива та надає рекомендації щодо оптимального використання модулів автомобіля, у тому числі і автокондиціонера, для досягнення цих цілей.

Застосунок має можливість імпорту та експорту даних, що дозволяє зберігати журнали подорожей та витрат палива на зовнішніх пристроях. На основі отриманих даних, Drivvo надає можливість розрахунку вартості подорожей, включаючи витрати на паливо та сервісне обслуговування автомобіля.

Застосунок надає можливість додавання нотаток та фотографій до записів про подорожі, включаючи коментарі про використання автокондиціонера.

Drivvo надає статистику та графіки щодо споживання палива, допомагаючи аналізувати свої витрати та ефективність використання автокондиціонера. Приклад

графіку витрат зображено на рисунку 2.8.

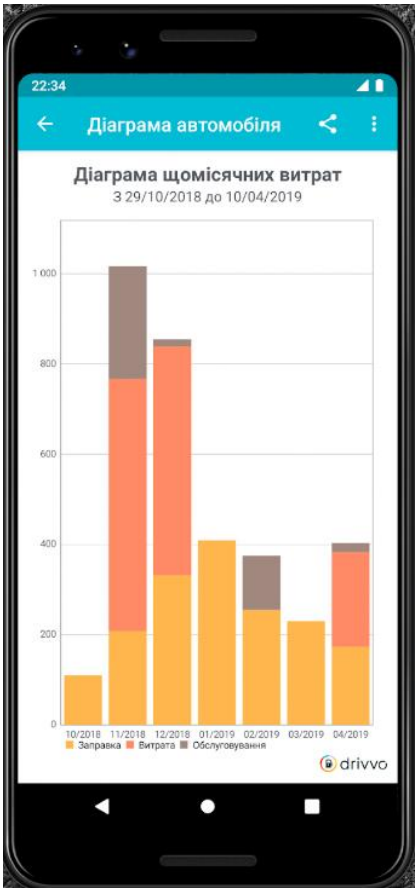

Рисунок 2.8 – Діаграма аналізу витрат за місяць

Застосунок має простий та інтуїтивно зрозумілий інтерфейс, що полегшує ведення журналу автомобіля та використання його функцій. Drivvo надає можливість резервного копіювання та відновлення даних, що допомагає зберегти записи користувача та налаштування безпечно. Застосунок підтримує мультиплатформенну синхронізацію, що дозволяє використовувати його на різних пристроях та зберігати дані синхронізованими [8].

Drivvo надає можливість відстежувати інші елементи обслуговування автомобіля, такі як заміна фільтрів та масла, що підтримує оптимальну роботу автокондиціонера. Застосунок зберігає історію усіх витрат на машину та усіх технічних обслуговувань, що вводилися користувачем, і таким чином дозволяє стежити за їх графіком, включаючи перевірку та очищення системи кондиціонування повітря. Приклад такої історії витрат зображено на рисунку 2.9.

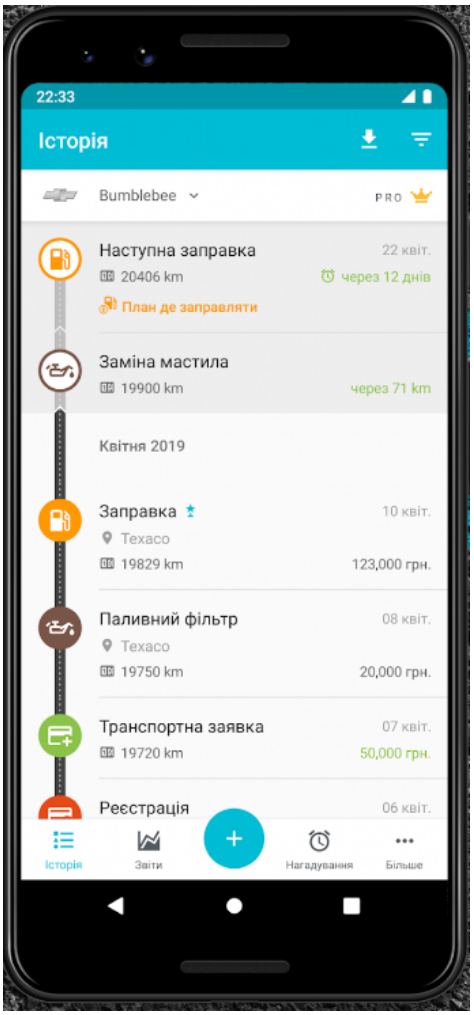

Рисунок 2.9 – Історія ТО та заправок

Перевагою застосунку є підтримка великої кількості мов таких як: українська; англійська, іспанська, португальська, французька, німецька, італійська, польська, корейська та ще 12 мов.

Корисним функціоналом застосунку є вбудований калькулятор витрат палива, що дозволяє зручно розраховувати споживання палива на певну відстань або тривалість подорожі. На обчислення даних витрат впливає і заправка, де здійснювалася купівля пального, адже застосунок має дані у базі про ціни про велику кількість заправних станцій у Європі та Америці. Сторінка калькулятора застосунку зображено на рисунку 2.10.

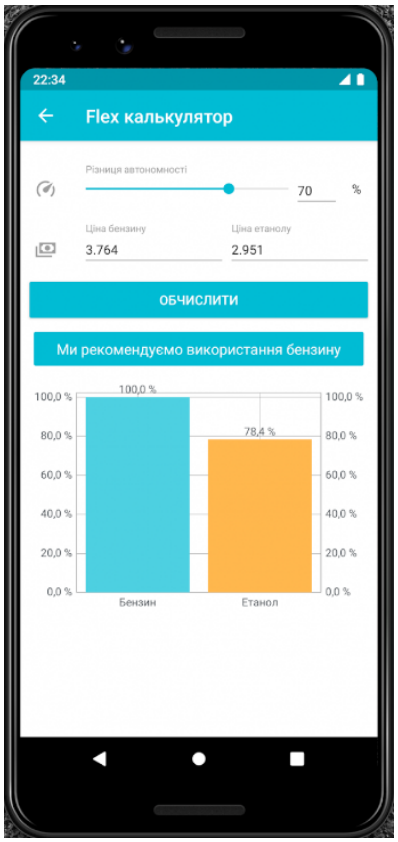

Рисунок 2.10 – Калькулятор витрат на паливо

Недоліком застосунку є обмежена підтримка певних марок автомобілів і відсутність функцій для енергоефективного використання автокондиціонера у реальному часі, а надаються лише загальні поради.

Висновки до розділу 2

Вищерозглянуті застосунки є корисними і надають велику кількість функціоналу для підвищення енергоефективності водіння. Окрім цього, вони мають і певну кількість недоліків, таких як: підтримка тільки певних марок автомобілів, обмежена кількість країн, можливе їх використання, та технологічні недоліки і помилки під час роботи. Жоден з наведених застосунків не має відповідного функціоналу для надання рекомендацій щодо енергоефективного використання автокондиціонера у реальному часі, що визначає необхідність розробки власного ПЗ.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Мова програмування Java та Spring Framework

Для реалізації серверної частини застосунку та тестового сервісу, щоб імітував стан машини, було обрано мову Java.

Java – потужна мова програмування, яка знайшла широке застосування у розробці серверних застосунків. Її основні переваги включають високу надійність, переносимість і масштабованість, що дозволяють створювати стабільні та ефективні серверні рішення. Java має велику кількість стандартних бібліотек та фреймворків, що спрощують розробку серверної частини програми і прискорюють процес розробки [9]. Принцип "write once, run anywhere" (пиши один раз, запускай скрізь), який означає, що програми, написані на Java, можуть бути виконані на будь-якому пристрої, який підтримує віртуальну машину Java (JVM). Дана кросплатформеність застосунків написаних на Java дозволяє розробникам створювати програми, які можуть працювати на різних операційних системах, таких як Windows, macOS та Linux, зменшуючи необхідність переписувати код для кожної платформи окремо. Невід’ємною перевагою Java є те, що мова має широкий спектр інструментів для управління пам'яттю, що дозволяє автоматично виконувати збірку сміття (garbage collection) і зменшує ризик витоку пам'яті [10].

Окрім того, Java підтримує багатопотоковість, що дозволяє розробникам ефективно використовувати паралельні обчислення та забезпечувати більшу продуктивність програм. Необхідно відмітити, що мова надає високий рівень безпеки завдяки системі управління доступом та механізмам обробки винятків, що допомагають уникнути критичних помилок та забезпечити безпеку даних.

На додачу до всього, Java має широке співтовариство розробників, що підтримується активними форумами, бібліотеками та документацією, що сприяє зручному обміну знаннями та підтримці у розробці програм на даній мові. Це все

дуже спрощує розробку, підтримку та розгортання програм на різних пристроях та забезпечує більшу універсальність.

Дуже підсилює дану мову програмування створений для неї Spring Framework. Дана утиліта – це потужний фреймворк для розробки серверних застосунків у мові програмування Java, який надає розширені можливості та забезпечує ефективну розробку великих проектів [11]. Одна з переваг Spring Framework полягає в його інверсії управління, що дозволяє зосередитися на розробці бізнес-логіки, а фреймворк бере на себе управління об'єктами та залежностями.

Spring Framework також має велику кількість модулів, які допомагають у вирішенні різних задач, таких як керування транзакціями, інтеграція з базами даних, веб-розробка та багато іншого. Крім того, Spring Framework пропонує підхід "контейнеру IoC", що полегшує тестування та розширення застосунків, а також підтримує архітектурні шаблони, такі як MVC, для створення добре організованих та підтримуваних серверних застосунків.

Spring Framework є проектом, що активно розвивається з великою спільнотою розробників, що надає високу підтримку та доступ до багатьох ресурсів і документації для полегшення роботи з фреймворком. Багато компаній та організацій обирають Spring для своїх проектів, оскільки він вважається стандартом для розробки Java-застосунків. Spring надає механізми для розширення функціональності за допомогою власних модулів і розширень, що дозволяє забезпечити високу гнучкість і адаптованість до потреб проекту.

Фреймворк також пропонує вбудований механізм обробки помилок та винятків, спрощуючи розробку безпечних та надійних застосунків. Spring Framework є ключовим компонентом для створення сучасних мікросервісних архітектур, де він забезпечує управління залежностями, масштабованість та інтеграцію різних сервісів у єдину систему[12].

Присутні інструменти для підключення та подальшої роботи системи із базами даних.

3.2 Мова програмування Dart та Flutter Framework

Для реалізації мобільного застосунку, було обрано мову програмування Dart.

Dart - це мова програмування, розроблена компанією Google, яка використовується для розробки мобільних застосунків та фронтенду для web-сервісів [13]. Одна з головних переваг полягає в його ефективному та швидкому виконанні, що дозволяє створювати високоякісні застосунки з плавною реакцією користувача. Dart також має зручний синтаксис і сучасні функціональні можливості, такі як асинхронне програмування, що полегшують розробку складних застосунків.

Dart підтримує гарний набір бібліотек та фреймворків, які дозволяють створювати крос-платформені мобільні застосунки для iOS та Android. Крім того, Dart має вбудовану систему типів, що допомагає виявляти помилки на етапі компіляції та забезпечує більшу надійність та стабільність програм. Dart також активно розвивається та підтримується спільнотою розробників, що забезпечує доступ до нових функцій та постійні вдосконалення[14].

Dart також має вбудовану підтримку гарячої перекомпіляції, що дозволяє швидко вносити зміни в код та бачити їх результат без необхідності повного перекомпілювання програми. Мова має можливості збірки сміття, що полегшує керування пам'яттю та дозволяє уникнути проблем з її витоками. Dart надає розширені можливості для роботи анонімними класами, що спрощує створення лаконічного коду. Dart є основною мовою розробки для фреймворку Flutter, який дозволяє створювати швидкі крос-платформені мобільні та веб-застосунки з однієї кодової бази.

Flutter - це відкритий фреймворк розробки інтерфейсу користувача, розроблений компанією Google, який дозволяє створювати високоякісні крос-платформені мобільні застосунки. Одна з головних переваг Flutter полягає в його швидкодії та продуктивності, що забезпечує плавну анімацію та швидку реакцію користувача. Flutter має вбудовану набір віджетів, які дозволяють створювати

зручний інтерфейс зі змінними ефектами та стилізацією, що спрощує розробку застосунків [15].

Фреймворк підтримує розробку на різних платформах, включаючи iOS, Android, та навіть настільні платформи, що забезпечує широку аудиторію для розповсюдження застосунків. Flutter пропонує можливість написання одноразового коду для інтерфейсу користувача та бізнес-логіки, що дозволяє покращити швидкість розробки та підтримку проекту. Фреймворк має вбудовану підтримку анімацій, що дозволяє створювати ефекти для взаємодії з користувачем. Flutter також надає доступ до багатьох сторонніх бібліотек та пакетів, що допомагають розширити функціональність та можливості розробки[16].

Firebase – це потужна платформа для розробки мобільних та веб-сервісів, що надає безліч інструментів та сервісів для зручного управління збереженням даних, аутентифікацією, хостингом, аналітикою та багатьма іншими функціями[17]. Завдяки Firebase, розробники можуть швидко налаштовувати інфраструктуру проекту без необхідності розгортання та керування власним сервером. Його використання дає змогу легко налаштувати безпечний процес автентифікації та реєстрації юзера, з можливістю використовувати акаунт Google для цього, не витрачаючи час на реалізацію та налаштування безпеки для подібного кастомного функціоналу.

Firebase також надає можливість використовувати реальний час (real-time) синхронізовану базу даних, яка дозволяє взаємодіяти з даними в режимі реального часу без необхідності оновлення сторінки або запиту до сервера. Це особливо корисно для розробки певних застосунків та чатів.

Firebase забезпечує легку інтеграцію з різними сервісами Google, такими як Google Analytics, Google Cloud Messaging (FCM) і Google AdMob. Це дає можливість використовувати потужні аналітичні і рекламні інструменти для покращення продуктивності та монетизації застосунків.

Платформа Firebase також має SDK для різних платформ, включаючи Android, iOS, веб і Unity. Це дозволяє розробникам використовувати у проектах на

різних платформах, забезпечуючи єдиною точкою керування даними та функціями.

3.3 Середовища розробки IntelliJ IDEA та Android Studio

Не менш важливим, ніж вибір інструментів розробки, є її середовище. Для мов програмування Java є безліч платформ, що забезпечать комфортну розробку. Достатня їх кількість, хоча і явно менша, є і у мови програмування Dart. Чудовим варіантом для вирішення поставлених задач є – IntelliJ IDEA та Android Studio.

IntelliJ IDEA - це потужне інтегроване середовище розробки (IDE) для мов програмування Java та інших, яке надає розробникам широкий набір функцій та інструментів для зручного та ефективного програмування [18]. Одна з головних переваг даної платформи полягає в її інтелектуальних можливостях, таких як автоматичне завершення коду, рефакторинг, аналіз коду та підказки, що допомагають збільшити продуктивність розробника. IDE також має інтеграцію з різними системами контролю версій, такими як Git, що спрощує роботу зі спільним кодом та відстежування змін, що дуже корисно, адже розробка у більшості випадків виконується декількома людьми.

IntelliJ IDEA також надає підтримку для різних фреймворків і технологій, що спрощує розробку різноманітних типів проектів. Вона має вбудовану підтримку для веб-розробки, що дозволяє створювати динамічні веб-сайти та взаємодіяти з різними серверними технологіями, такими як Java Servlets, JSP, Spring та багатьма іншими. Вона також надає можливість виконувати автоматичне тестування програмного коду, що допомагає забезпечити якість та стабільність розроблюваного програмного продукту. Крім того, IntelliJ IDEA підтримує інтеграцію з різними базами даних, що дозволяє розробникам легко працювати з даними та виконувати запити до бази даних прямо з інтерфейсу IDE [19].

Android Studio - це інтегроване середовище розробки (IDE) для платформи Android, яке надає зручні інструменти та ресурси для створення високоякісних мобільних застосунків [20]. Одна з головних переваг Android Studio полягає в його

потужних можливостях розробки, включаючи швидкий компілятор, ефективний дизайнер інтерфейсу користувача та засоби для розробки та тестування застосунків на різних пристроях і емуляторах. Android Studio надає засоби для розробки на різних мовах програмування, що дозволяє розробникам вибрати найбільш зручний інструмент для їх потреб. Наявна інтеграція з системами контролю версій, такими як Git.

Android Studio також надає багато корисних інструментів для оптимізації та покращення продуктивності розробки. Це включає в себе можливості профілювання, які дозволяють виявити й усунути проблеми з продуктивністю та споживанням ресурсів в мобільних застосунках. Крім того, Android Studio має вбудовану підтримку тестування, що дозволяє розробникам виконувати автоматичне тестування функціональності та перевірку взаємодії зовнішніх компонентів. Інтегроване середовище також надає можливість розгортання та публікації застосунків на Google Play Store безпосередньо з IDE, що спрощує процес розповсюдження мобільних застосунків [21].

3.4 Інструмент збірки проектів Gradle

Gradle є потужним інструментом для автоматизації збирання, тестування і випуску програмного забезпечення [22]. Він надає зручний DSL (Domain Specific Language) для конфігурації проектів, що дозволяє налаштовувати і керувати їхнім поведінкою. Gradle використовує потужну систему залежностей, що дозволяє керувати ними між модулями та бібліотеками. Крім того, він підтримує багато інтегрованих плагінів і має активну спільноту, що допомагає вирішувати проблеми і забезпечує широкі можливості розширення.

Налаштування для збору проекту, як і для застосунку так і для серверу, знаходяться у файлі build.gradle. Даний файл складається з ключових слів, що позначають те чи інше налаштування проекту, або позначають блок скрипту, який відповідає за певний процес у побудові проекту. Так як IDE створює даний файл

автоматизовано, у нього потрібно вводити мінімум коректівів, таких змінна версії або додавання потрібних залежностей.

Для серверної частини важливими складовими у побудові проекту є блоки: plugins (де оголошуються плагіни, що використовуються у програмі), repositories (де вводиться назва репозиторію, звідки потрібно брати залежності), та tasks (що позначає виконуваний блок для запуску певних завдань).

Дуже важливим є блок, що позначається ключовим словом dependencies. У ньому за допомогою ключів implementation та compileOnly позначаються залежності, що по суті є віддаленими бібліотеками, що знаходяться у maven репозиторії в інтернеті. За допомогою даного блоку, програма має змогу підтягувати для побудови певні бібліотеки, написані вже кимось раніше, автоматично при збірці [23]. Це полегшує розробку, адже не потрібно вручну скачувати та додавати у проект дані модулі.

Варто відмітити і ключові слова version та sourceCompatibility, що відповідно дають змогу вводити версію програми при розробці і внесенні змін у код, та позначати якою версією мови Java даний код написаний, що є дуже важливим і спрощує розробку. Збір проекту через Gradle використовується і у тестовому модулі, що імітує поведінку машини. Для внесення певних змін у проект, та додавання необхідних конфігурацій, при написанні серверу за допомогою мови Java та фреймворку Spring, використовується файл application.properties або application.yaml.

3.5 Патерни MVC та MVVM

Окрім затвердження архітектури системи, необхідно створити архітектуру кожного модуля, а саме клієнта та сервера. У цьому можуть допомогти уже наявні патерни, які підходять під вибрані інструменти та зроблять зрозумілою розробку.

MVC (Model-View-Controller) - це патерн проектування, що використовується для реалізації сервера. У MVC модель представляє дані та бізнес-

логіку застосунку, представлення відповідає за відображення даних користувачу, а контролер відповідає за обробку взаємодії користувача та керування моделлю та представленням. Схематично роботу патерна зображено на рисунку 3.1.

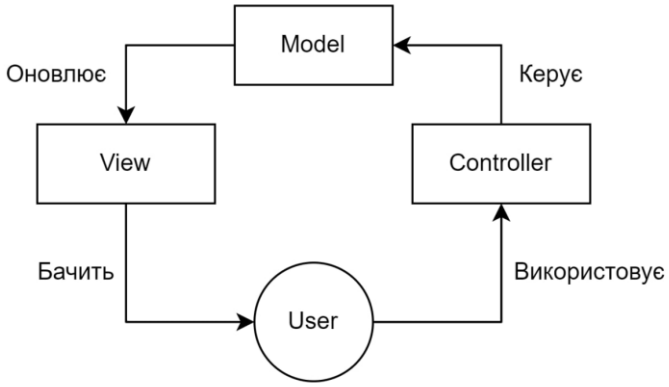

Рисунок 3.1 – Патерн MVC

Переваги використання патерна MVC (Model-View-Controller):

1. Розділення обов'язків: MVC розподіляє відповідальності між моделлю (Model), представленням (View) і контролером (Controller). Це дозволяє покращити структуру коду і розділити логіку застосунку на окремі компоненти;
2. Підтримка повторного використання: завдяки розділенню логіки застосунку на компоненти, можна повторно використовувати моделі, представлення і контролери в інших частинах програми. Це сприяє ефективному використанню коду і скороченню часу розробки;
3. Легке тестування: оскільки MVC розділяє логіку застосунку на окремі компоненти, це полегшує написання модульних тестів для кожного компонента. Тестування моделі, представлення і контролера окремо дозволяє ефективно перевіряти правильність їх роботи.
4. Гнучкість і масштабованість: MVC надає гнучкість при розробці, оскільки окремі компоненти можуть бути змінені або розширені без впливу на решту системи. Це дозволяє легко вносити зміни в застосунок і масштабувати його в майбутньому.

5. Стандартизація: використання патерна MVC допомагає встановити стандартизовану архітектуру для розробки застосунків.

MVVM (Model-View-ViewModel) - це патерн проектування, що схожий за принципом роботи до MVC, але який використовується для розробки клієнтських застосунків з графічним інтерфейсом [24]. У MVVM модель представляє дані та бізнес-логіку застосунку, представлення відповідає за відображення даних користувачу, а Viev-Model служить посередником між моделлю та представленням, надаючи методи та команди для взаємодії з моделлю. Цей патерн дозволяє відокремити логіку взаємодії з даними від логіки відображення, що полегшує тестування та збереження стану інтерфейсу користувача [25]. Схематично роботу патерна зображено на рисунку 3.2.

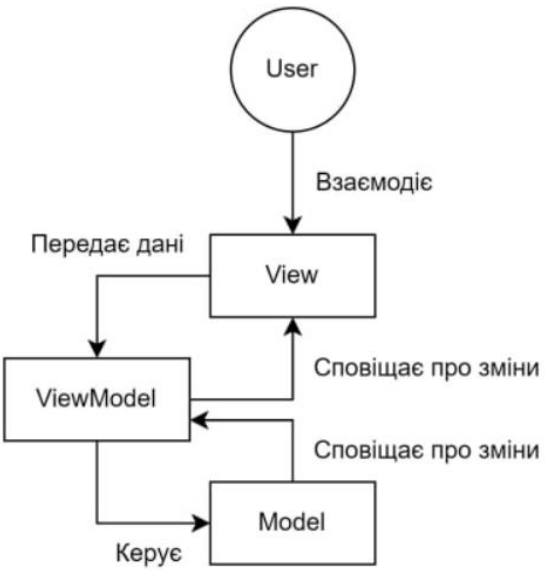

Рисунок 3.2 – Патерн MVVM

Переваги використання патерна MVVM (Model-View-ViewModel):

1. Забезпечення реактивного підходу: MVVM може використовувати реактивне програмування, де зміни в моделі автоматично оновлюють представлення через механізми спостереження (наприклад, використання Observable або Reactive Extensions). Це дозволяє створювати динамічні інтерфейси користувача з автоматичним оновленням даних.

2. Легке управління станом застосунку: ViewModel виконує роль зберігання та управління станом застосунку. Це дозволяє ефективно керувати станом елементів інтерфейсу користувача, що спрощує реалізацію складних логічних взаємозв'язків.
3. Масштабованість та повторне використання: MVVM дозволяє розділити логіку, представлення і дані на окремі компоненти. Це сприяє масштабованості та повторному використанню коду. ViewModel може бути повторно використана в різних представленнях, що дозволяє зменшити зусилля при розробці багатоплатформенних застосунків.

Висновки до розділу 3

У розділі були описані засоби розробки, які використовуються для створення даної системи. Було обґрунтовано їх вибір для розробки проекту та наведено переваги їх використання. Для реалізації мобільного застосунку було обрано мову програмування Dart з фрейворком Flutter та патерн MVVM. Для реалізації серверної частини, та тестового сервісу було обрано мову програмування Java з фреймворком Spring та патерн MVC. Середовищами розробки було обрано Android Studio та IntelliJ IDEA.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура системи та алгоритм роботи застосунку

Архітектура системи, що формує рекомендації для енергоефективного використання автокондиціонера являє собою клієнт-сервер, що спілкуються між собою через HTTP запити. Архітектуру системи із виконуваними функціями зображено на рисунку 4.1.

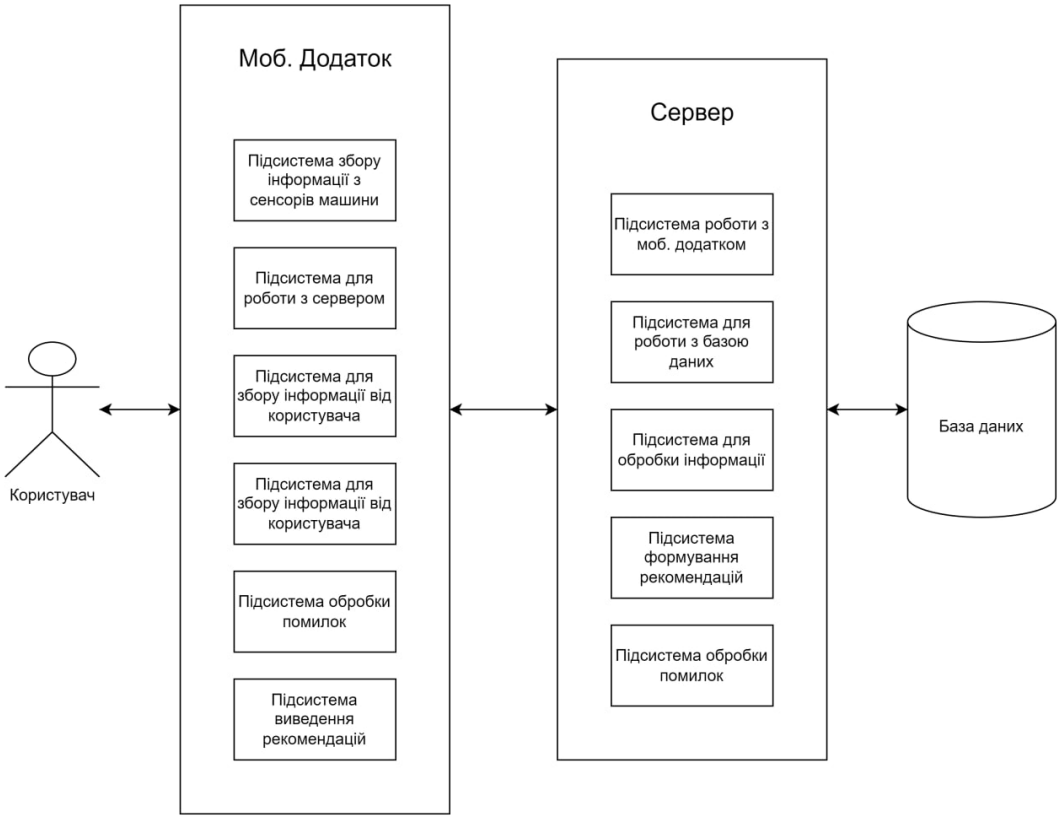

Рисунок 4.1 – Архітектура системи

Мобільний застосунок, що відіграє роль клієнта, виконує наступні функції:

1) збирає інформацію від користувача та зчитує дані з машини;
2) відповідає за автентифікацію користувача;
3) відповідає за передачу зібраних даних до серверної частини;
4) відповідає за отримання від сервера рекомендацій та виведення їх користувачу;

Серверна частина застосунку, що, відповідно, відіграє роль серверу в архітектурі системі, виконує наступні функції:

1) отримує дані від застосунку;
2) аналізує отриману інформацію, та формує рекомендації для користувача;
3) зв’язок та обмін даними з базою даних;
4) надсилає потрібну інформацію та рекомендації до мобільного застосунку.

Концептуальна модель роботи системи зображено на рисунку 4.2.

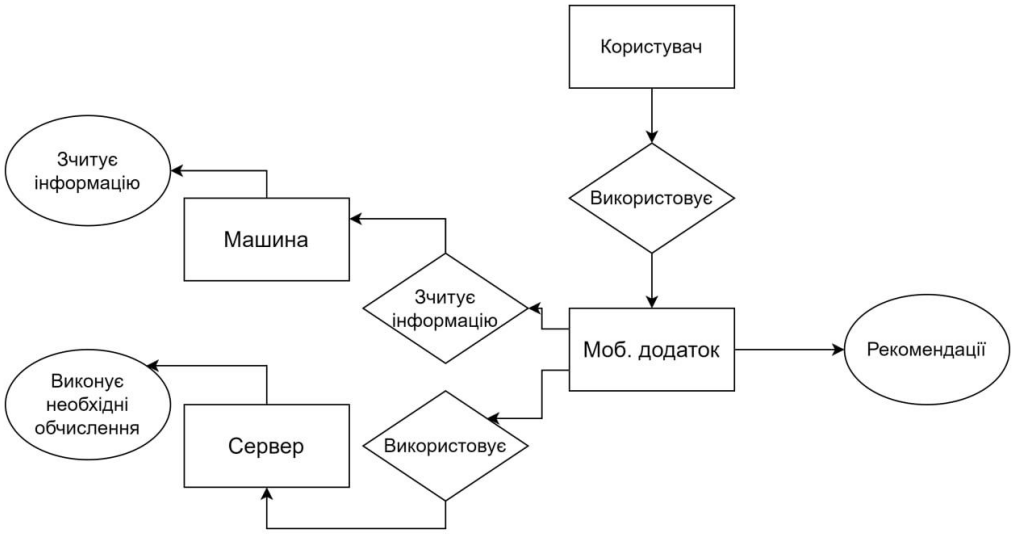

Рисунок 4.2 – Концептуальна модель роботи системи

Алгоритм роботи системи можна описати наступними пунктами:

1) користувач відкриває застосунок;
2) користувач автентифікується;
3) мобільний застосунок підключається до машини автоматично після автентифікації, та зчитує потрібну інформацію;
4) мобільний застосунок, за вимогою користувача, формує запит, передаючи дані до серверу для отримання рекомендацій або необхідної інформації;
5) сервер оброблює запит від застосунку, та звертається до бд за інформацією або передає дані до алгоритму, щоб сформувати рекомендації та зробити необхідні обчислення;
6) сервер надсилає відповідь до запиту застосунку;

7) мобільний застосунок отримує рекомендації або потрібну інформацію від сервера;
8) застосунок виводить необхідні рекомендації на екран;
9) користувач налаштовує кондиціонер відповідно до отриманих рекомендацій, та, за потреби, може їх оновити у будь-який момент.

Схематично алгоритм роботи зображено на рисунку 4.3.

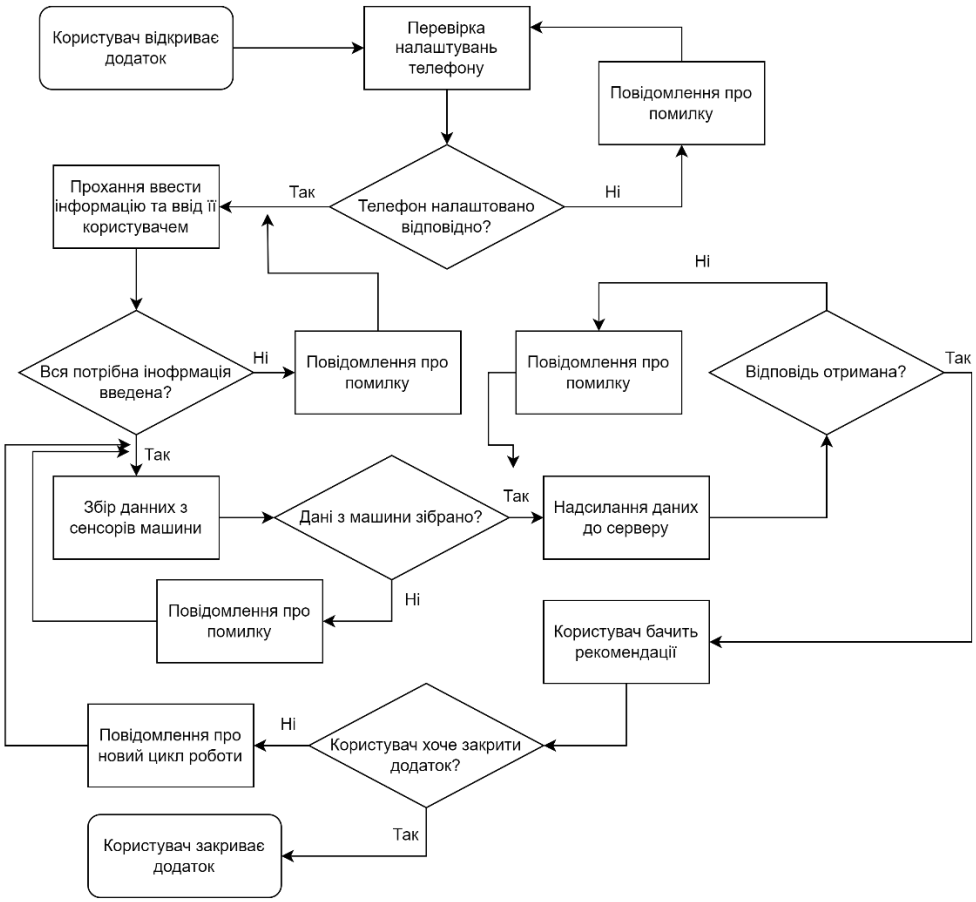

Рисунок 4.3 – Алгоритм роботи застосунку

Розроблена архітектура дає змогу створити застосунок, який зручний для використання та дозволяє без зайвих труднощів вносити новий функціонал, або вирішувати наявні проблеми під час відловлювання помилок у роботі, адже система масштабована і поділена на окремі модулі, які відповідають за окремий специфічний функціонал, що полегшує роботу з ним.

4.2 Серверна частина застосунку

Було створено серверну частину системи, яка відповідає за аналіз даних у системі та його основі формує відповідні рекомендації по використанню автокондиціонера для користувача застосунку.

Було створено спеціальні пакети, що будуть репрезентувати шари програми. У даному випадку папки: configuration, controller, entity, service, exception.

У шарі конфігурації був створений один файл: BeanGarden.java. Даний клас був позначений анотацією @Configuration, що допоможе Spring застосувати принцип IoC, та автоматизовано використовувати у коді об’єкти, що створюються через функції, що знаходяться у даному класі і позначені анотацією @Bean. У даному випадку створюється метод localApiClient(), що повертає об’єкт класу WebClient, що слугує для зв’язку сервера з іншими модулями системи.

Шар entity – це папка із класами, що являють собою певні об’єкти із реального світу, і які зберігають інформацію, що використовується під час роботи програми.

Поля класів описують, необхідні для роботи, їх особливості. Основними класами у даній роботі є: Car.java, CarExpanded.java, User.java, Conditioner.java, Sensor.java, LoginResponse.java та енам State.java. Останні два – репрезентують, відповідно, відповідь на запити з автентифікації по принципу DTO (Data Transfer Object), та enum станів кондиціонера: CONDITIONING, CONDITIONING_VENTILATION, VENTILATION, HEATING, HEATING_VENTILATION, RECIRCULATION, DEMISTING. CarExpanded.java, User.java – це класи, що являють собою об’єкт у базі даних, де кожне поле це відповідний рядок у таблиці. Car.java, Conditioner.java, Sensor.java – класи, що представляють собою стан кондиціонера та сенсорів машини, та її саму. При запитах для аналізу стану автомобіля, мобільний застосунок надсилає на відповідний endpoint об’єкт класу Car.java, з полів якого будуть витягуватися об’єкти Conditioner.java та Sensor.java, що зберігають у собі необхідну інформацію.

У патерні MVC, даний шар є моделлю. Схожою за логікою є папка exception, у якій зберігаються описані винятки, що зустрічаються у програмі.

У папці service містяться класи, що відповідають за бізнес-логіку, та які позначені анотацією @Service, що дають змогу Spring застосувати принцип IoC до них. CarService.java та CarServiceImpl.java реалізують функціонал для збереження та отримання інформації про машину в та з бази даних, і її передачі до контролера. У подальшому додасться функції для додавання зв'язування доданого автомобіля із акаунтом користувача. AuthService.java та AuthServiceImpl.java повторюють функціонал з машиною, але вже для користувача, розширюючи його процесом реєстрації та входу в акаунт.

AviceService.java та AdviceServiceImpl.java – це місце, де описана основна логіка програми, а саме формування рекомендацій. У даному класі аналізується інформація, що передається від машини через контролер. За допомогою конструкцій if-else йде порівняння отриманих даних та формування як рекомендованих, так і оптимальних для енергоефективності рекомендацій. Також тут вираховується витрата пального для роботи автокондиціонера.

З усього цього складається об'єкт колекції Map, де за ключем «advice» знаходиться List, у якому зберігаються рекомендації. Даний об'єкт повертається до мобільного застосунку, через контролер, і там уже виводиться для користувача.

Шар controller – є, очевидно, шаром контролера у патерні MVC. Він є зв'язною ланкою проекту, що отримує запити, та повертає на них відповіді. Класи, що знаходяться у даному пакеті: AuthController.java та ResourceController.java, позначаються анотацією @Controller або @RestController (використовується другий варіант), для того, щоб показати Spring, що саме вони оброблюють запити, що надходять до сервера. Їх відмінність полягає у тому, яку вони відповідь надсилають: представлення для користувача (посилання на View шар), чи Json.

Змогу обробити запити надають анотації @GetMapping та @PostMapping, що позначають адресу, через яку можна звернутися з до сервера для отримання відповідних даних. У відповідь вони передають об'єкт класу, що записується у її

тіло. Методи, що позначені даними анотаціями, також можуть обробляти інформацію, що знаходиться у тілі, або заголовках запиту, через анотації @RequestHeader та @RequestBody.

У даному випадку шар View патерна MVC було замінено мобільним застосунком.

Потрібно також відмітити, що Spring автоматично після ініціалізації проекту створює головний клас проекту, та позначає його анотацією @SpringBootApplication, що позначає клас, де він створює метод main(), що є вхідною точкою проекту.

4.3 Мобільний застосунок

За допомогою обраного патерна MVVM, концепція застосунку виглядає наступним чином: увесь застосунок поділяється на окремі сторінки (page), у яких розташовані відповідні віджети, що являються елементами кожної сторінки, і з якими буде взаємодіяти користувач.

Окремо створюються події (events), що описують дії користувача на тій чи іншій сторінці, та прив’язуються до активних елементів віджетів. Коли користувач під час певної дії з застосунком, запускає у роботу певні події, вони перехоплюються елементом системи під назвою блок (bloc). Вони мають змогу обробити відповідні events та створити і надіслати відповідний запит (request) до сервера або до даних, щоб отримати, потрібну, для виконання запущеної події, інформацію, що зберігається у відповіді (response), котра надсилається назад у застосунок.

Отримавши її, блок трасформує інформацію у об’єкт стан (state), та надсилає назад до сторінки, де його у правильному вигляді починає бачити юзер, або відбувається завершення виконання відповідного event та перехід на іншу сторінку. Схематично дану взаємодію зображено на рисунку 4.4.

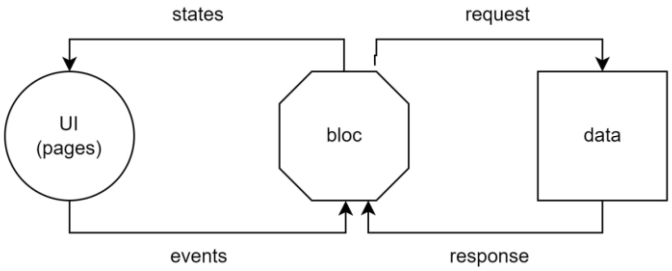

Рисунок 4.4 – Взаємодія даних

Для даного застосунку, було виділено точну кількість сторінок, і для кожної з них розписати відповідні елементи: events, bloc, state. Дану процедуру було проведено для сторінок, що є Statefull, тобто є тими, з якими користувач може взаємодіяти, і які зберігають у собі певну логіку. Протилежним до даного стану, є сторінки Stateless, що є просто інформаційними, та не відслідковують ніякої взаємодії з користувачем.

У даній роботі, під другу класифікацію підпадали такі сторінки як: splash_page.dart та no_internet_connection_page.dart, що відповідають за сторінки, що бачить користувач при завантаженні застосунку, та при вимкненого Інтернету на мобільному пристрої.

Усі інші сторінки, що підпадають під першу класифікацію, в основному знаходяться у папці page, де мають свій пакет з іншими елементами, а саме: block, event та state необхідними для роботи. Приклад даної реалізації для сторінки з інформацією про машину зображено на рисунку 4.5.

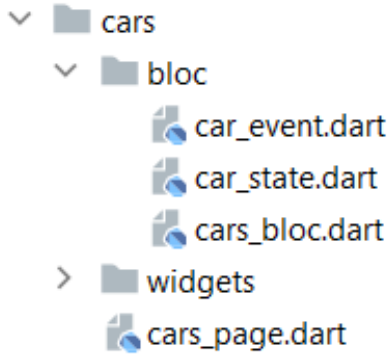

Рисунок 4.5 – Модуль сторінки із інформацією про машину

Кожен згаданий раніше елемент наслідує певні абстрактні класи, які мають певні початкові налаштування, що спрощує та пришвидшує розробку застосунку. Умовний шаблон, що використовується у даному випадку, можна імпортувати як залежність з відповідного репозиторію в Інтернеті, у даному випадку вони знаходяться у папці base. Таким чином, для кожної Statefull сторінки було прописано елемент page, в якому наявні активні віджети. Для кожного з них прописати event, який запускався при взаємодії з певним елементом на сторінці, та відповідний bloc та state, щоб відповідав за обробку події, та інформації, що поверталася у відповідь. Дана концепція застосована при розробці сторінок: sign_in_page.dart, sign_up_page.dart, cars_page.dart, recommendations_page.dart, resource_info_page.dart, app.dart. Самі віджети, що використовуються у програмі, знаходяться у папці widgets.

Основними початковими об’єктами, що було створено, це класи, що відповідають за модель у патерні MVVM. Вони знаходяться у папці models, і являють собою об’єкти, які вже використовувалися при розробці сервера: кондиціонер, сенсор, машину, користувача та інші, що не використовувалися: рекомендації. Вони містять у собі поля, що зберігають у собі, потрібну для роботи, інформацію, та методи, такі як перетворення їх з Json та на Json.

У даній роботі конфігурація Firebase прописується у файлі firebase_auth_service.dart, де реалізуються функції для реєстрації та входу в акаунт.

Для підключення локальної бази даних, було використано Hive, що є легкою та швидкою базою даних у вигляді ключ-значення, написана на Dart, і яка дозволяє зберігати та синхронізувати дані застосунку офлайн. Усі необхідні методи та налаштування для даної бази даних знаходяться у файлах hive_contract.dart, hive_provider.dart та hive_repository.dart.

Створено елементи, за допомогою яких можна звертатися до машини та сервера. Вони знаходяться у папці network. Там знаходиться кілька елементів: пакет api, що формує зв’язки для з’єднання із сервером та машиною; client – що

являє собою клієнта для відправлення та обробки відповідних запитів, та інші елементи, які є самими запитами або відповідями на них.

Також реалізовано різні утиліти, що допомагають при розробці спрощуючи процес. Це елементи, що знаходяться у папці utils, а саме: extensions, що описує певні загальні функції, які потім використовуються у багатьох місцях у програмі; mixins, що вміщає у себе певні функції, щоб не повторювати їх на усіх сторінках програми; validators – для перевірки правильності вводу даних у поля.

Головний функціонал, що збирає воєдино увесь описаний вище функціонал знаходиться у пакеті applications, що виконує функцію основи застосунку, на якій усе будується.

Файли constants.dart та main.dart – відповідно, зберігають у собі константи програми, та є основним класом у програмі з головним методом main().

4.4 Сервіс для тестування застосунку

За шар контролеру проекту відповідають класи, що знаходяться у папці controller. Клас ResourceController.java – відповідає за надсилання застосунку інформації про стан автомобіля. Клас StateController.java – відповідає за використання шару бізнес-логіки, що налаштовує стан автомобіля.

Шар моделі патерна – це пакет entity, де знаходяться класи, що зберігають інформацію, яка використовується при роботі застосунку.

Папка service – шар бізнес-логіки проекту, де знаходяться функції, що налаштовують стан автомобіля.

Для імплементації шару View були додані HTML сторінки з використанням CSS. Вони надають змогу ввести тестові дані машини, використовуючи зрозумілий інтерфейс на сторінці в Інтернеті. Для того, щоб мати змогу передати або отримати дані сервісу, було підключено бібліотеку Thymeleaf, що за допомогою ключових слів, що вставляються у теги розмітки HTML, і допомагають зв’язати front і back частини.

Щоб працювати із сервісом, необхідно просто відкрити відповідне посилання у браузері, та ввести у поля той стан машини, рекомендації для якого цікавлять користувача. Їх автоматично зчитає застосунок, після початку роботи.

Висновки до розділу 4

У даному розділі було розглянуто архітектуру системи, що розроблялася. Наведено схеми, що надають змогу краще зрозуміти, як влаштований додаток та які підходи в його проектуванні використовувались. Було наведено пояснення загальної архітектури системи у цілому та розписано реалізацію кожного модуля.

5 ВЗАЄМОДІЯ КОРИСТУВАЧА З СИСТЕМОЮ

5.1 Системні вимоги

Розроблений застосунок призначений для користувачів, що мають телефон з операційною системою Android. Пристрій повинен мати як мінімум 50 МБ вільної внутрішньої пам'яті та мінімум 2 ГБ оперативної пам’яті. Клієнт повинен мати доступ до інтернету. Для запуску застосунку на комп'ютері, користувач повинен мати 200 МБ вільної внутрішньої пам’яті та мінімум 16 ГБ оперативної пам’яті.

5.2 Встановлення мобільного застосунку

Після перевірки пристрою на відповідність системним вимогам, для встановлення застосунку на телефон з ОС Android потрібно завантажити файл AC_Recommendation.apk, запустити даний файл, та натиснути «Завантажити», після чого відкриється інформаційне вікно про початок встановлення застосунку, яке зображено на рисунку 5.1.

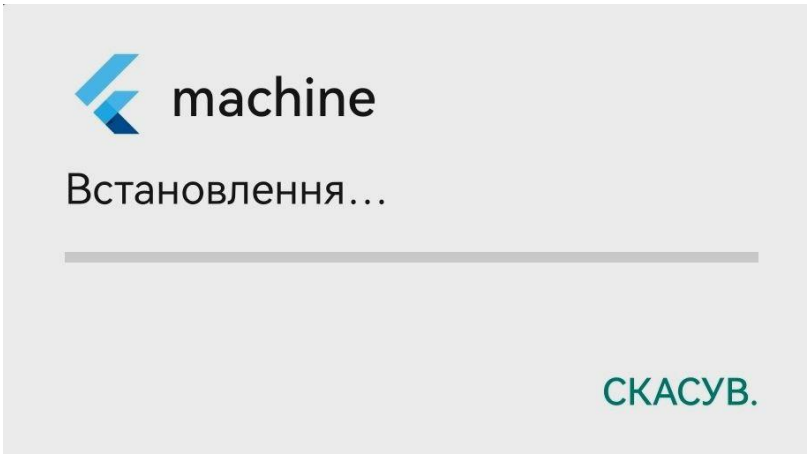

Рис. 1 – Вікно встановлення застосунку

Під час даного процесу можуть з’явитися діалогові вікна про надання додаткових дозволів для встановлення програми, для продовження потрібно підтвердити надання цих дозволів.

Після успішного встановлення, на телефоні з’явиться іконка даного

застосунку, що зображено на рисунку 5.2.

Рисунок 5.2 – Іконка застосунку

Для початку роботи із застосунком потрібно натиснути на дану іконку і додаток автоматично запуститься.

5.3 Робота користувача з системою

При запуску застосунку, якщо на телефоні не увімкнений WI-FI, то користувачеві буде показано інформаційний віджет, що не зникне, поки підключення до Інтернету не з’явиться, як показано на рисунку 5.3.

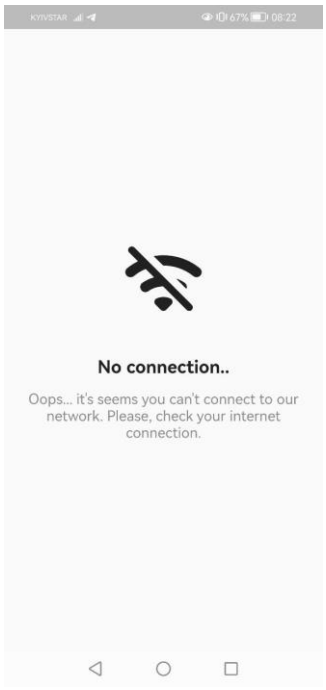

Рисунок 5.3 – Інформаційний віджет, про відсутність увімкеного WI-FI

Коли підключення з’явиться, то у користувача, якщо він до того не увійшов у свій акаунт, на телефоні відкриється віджет для авторизації. На даній сторінці він також може зареєструватися, або використати акаунт Google для входу. Для кожного поля, з яким може взаємодіяти користувач, реалізована валідація, для того, щоб не були введені, та збережені явно некоректні дані. Перевірка введених даних реалізовано зрозумілим для користувача чином, щоб він міг легко дотриматися встановлених правил. Даний функціонал авторизації та реєстрації показано на рисунках 5.4 – 5.6.

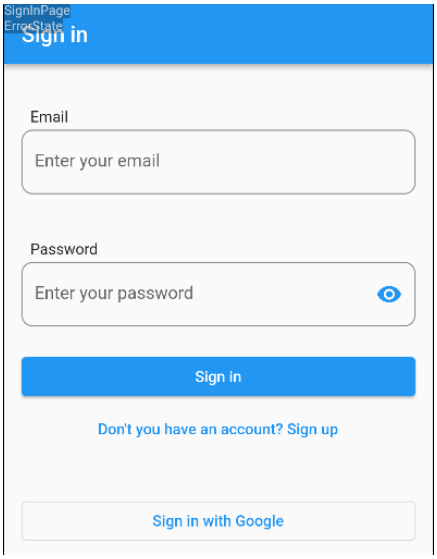

Рисунок 5.4 – Сторінка входу в акаунт

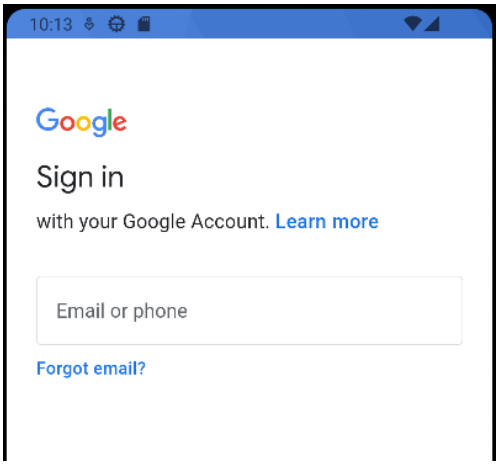

Рисунок 5.5 – Сторінка входу в акаунт через Google

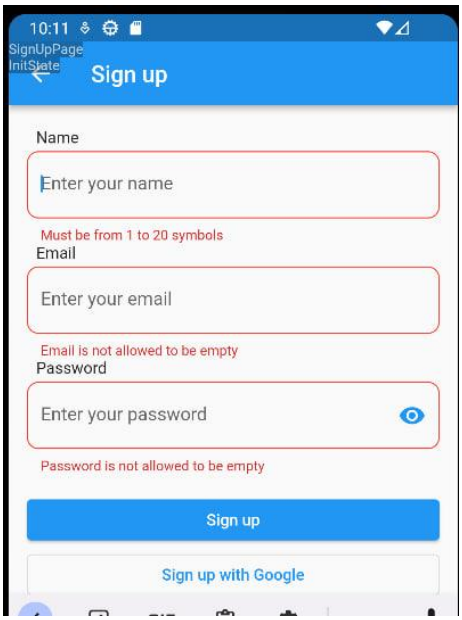

Рисунок 5.6 – Сторінка реєстрації з валідацією

Після входу в акаунт або реєстрації, застосунок автоматично підключається до машини, та відразу зчитує її налаштування, що потрібно для роботи застосунку. Зчитані дані відразу виводиться на екран, і користувач може їх переглянути. Це зображено на рисунку 5.7.

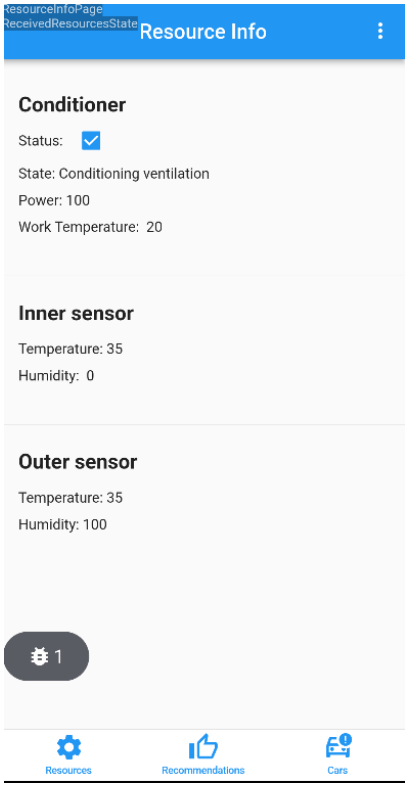

Рисунок 5.7 – Сторінка з інформацією від машини

Користувач, для перегляду сформованих рекомендацій, знизу має натиснути кнопку Recommendations. Сформовані рекомендації на сторінці можна переглянути на рисунку 5.8.

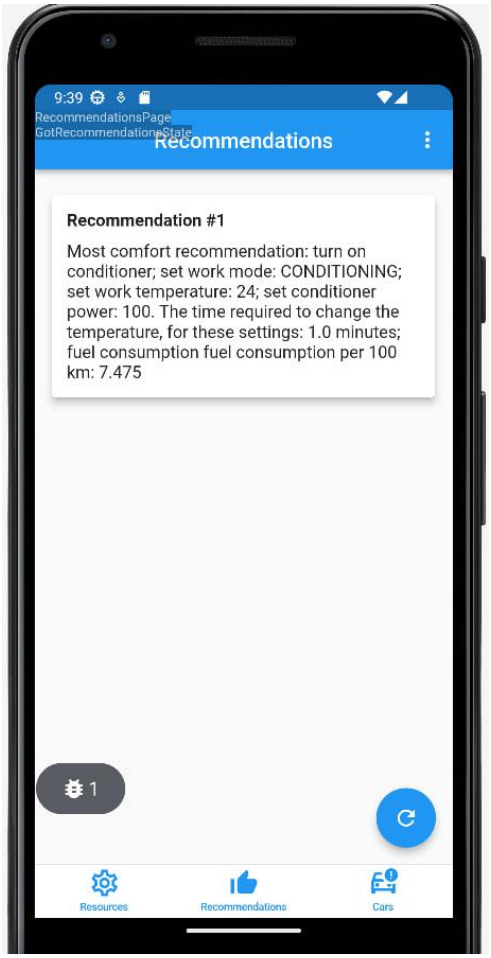

Рисунок 5.8 – Сторінка рекомендацій

Водій може легко переглядати сформовані рекомендації, а якщо захоче їх оновити, він може скористатися спеціально розробленим для цього функціоналом, натиснувши на кнопку оновлення, яка знаходиться у нижньому правому кутку екрану, і згідно нових зчитаних даних із сенсорів машини, рекомендації оновляться на ті, які краще підходять до нових вхідних даних.

Для перегляду інформації про свою машину, користувачеві потрібно натиснути на нижню праву кнопку Cars, що знаходиться у навігаційному меню застосунку внизу екрану, і це перенесе його на відповідну сторінку. Дану сторінку

зображено на рисунку 5.9.

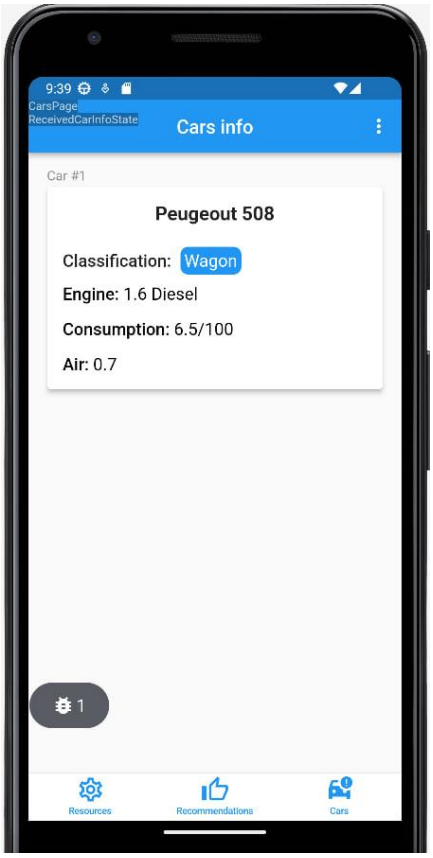

Рисунок 5.9 – Сторінка інформації про машину

Щоб змінити стан машини, для тестування формування рекомендацій при різних погодних умовах, користувач використає створений тестовий сервіс, що імітує стан автомобіля. Налаштування відповідних модулів вводиться через поля, одне з яких зображене на рисунку 5.10.

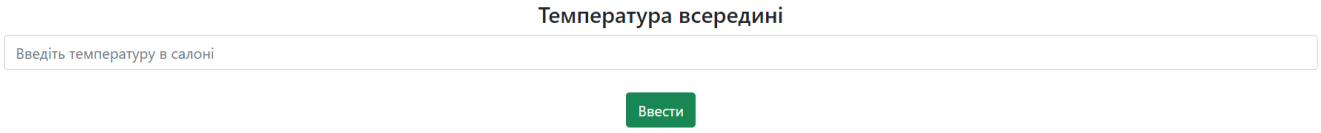

Рисунок 5.10 – Поле налаштування стану автомобіля

Користувач, після введення значень, для налаштування стану машини, може одразу їх переглянути у даному сервісі.

Стан машини, що може переглянути на тестовому сервісі користувач, зображено на рисунку 5.11.

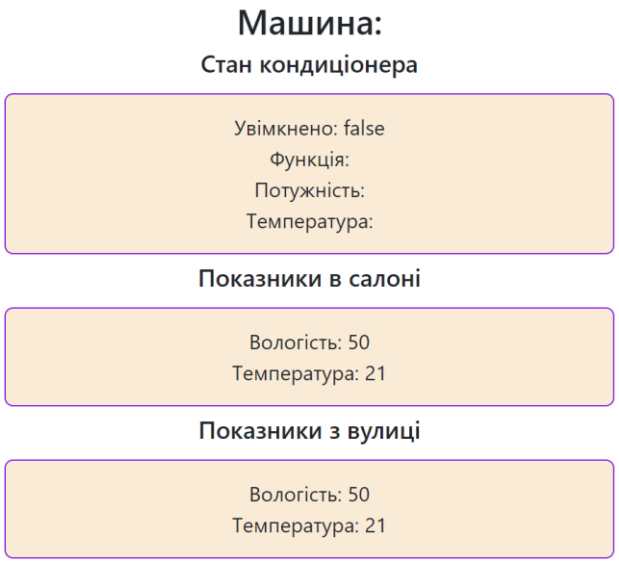

Рисунок 5.11 – Блок інформації про стан автомобіля

Даний сервіс надає можливість легкого тестування застосунку під час його розробки, та дозволяє не використовувати автомобіль протягом даного процесу.

Висновки до розділу 5

У даному розділі було продемонстровано процес використання користувачем застосунку, його функціонал та сервіс для його тестування.

Даний застосунок має можливості до подальшого розвитку та вдосконалення шляхом розширення його функціоналу, та додавання передових технологій таких як штучний інтелект.

ВИСНОВКИ

У ході роботи було досліджено те, як влаштований автокондиціонер, його режими роботи та умови у яких їх використовувати найкраще. На основі отриманої під час дослідження інформації, була розроблено алгоритм та імплементовано його у програму, що формує рекомендації для енергоефективного використання автокондиціонера.

На основі поставлених у роботі завдань, для їх вирішення, було виконано наступні дії:

1. проаналізовано аналоги додатків для підвищення енергоефективності водіння, виявлено, що жоден не має функціоналу для надання рекомендацій енергоефективного використання автокондиціонера у режимі реального часу;
2. спроектовано архітектуру системи, яка складалася з мобільного застосунку та серверної частини;
3. розроблено серверну частину програми, яка була написана за допомогою мови програмування Java та фреймворку Spring, використовуючи патерн MVC. Даний модуль містить функції, що відповідають за прийом та обробку даних, що надходили від мобільного застосунку. На основі отриманих даних, проводився аналіз усіх факторів, що тим чи іншим чином впливають на енергозатратність системи кондиціонування, та формувалися і відправлялися необхідні рекомендації до мобільного застосунку;
4. реалізовано web-сервіс, що імітував собою стан машини. Він дає змогу протестувати застосунок для рекомендацій, адже у режимі реального часу зв’язується з застосунком, і відповідно до його запитів надсилає необхідну інформацію;

5. розроблено мобільний застосунок, архітектура якого будувалася на основі патерна MVVM. Для написання коду в редакторі Android Studio, використовувалася мова програмування Dart та фрейморк Flutter;
6. протестовано систему на коректність роботи.

Необхідно відмітити, що окрім забезпечення енергоефективного використання автокондиціонера та зменшення витрат водія, за рахунок меншої витрати палива, застосунок таким чином забезпечує позитивний вплив на довкілля, адже використання пального у двигуні внутрішнього згорання прямо пропорційне до забруднення навколишнього середовища.

Застосунок має перспективу подальшого покращення, такі як впровадження штучного інтелекту і додавання функціоналу для обчислення шкідливого впливу викидів автомобіля у середовище.

Дипломна робота виконана в рамках проекту DRIVER`S BEHAVIOR COGNITION - Розпізнавання поведінки водія на основі сенсорів мобільного телефону (спільно з Політехнічним інститутом м. Томар, Португалія).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Different AC types. URL: https://www.swanswaygarages.com/blog/explained-air-conditioning-what-do-all-of-those-buttons-do/. (дата звернення: 29.05.2023).

2. The car air conditioner URL: https://www.aircowell.com/customers/advice/storytelling_detail_512.php. (дата звернення: 29.05.2023).

3. Fuel Economy Toolkit URL: https://www.fueleconomy.gov/feg/toolkit.shtml (дата звернення: 29.05.2023)

4. FuelEconomy.gov Web Services URL: https://www.fueleconomy.gov/feg/ws/. (дата звернення: 29.05.2023)

5. Meu Volkswagen URL: https://apkpure.com/meu-volkswagen/br.com.vw.connected.car. (дата звернення: 29.05.2023)

6. Meu Volkswagen App URL: https://play.google.com/store/apps/details?id=br.com.vw.connected.car&hl=ru&gl=US. (дата звернення: 29.05.2023)

7. 10 Professional fleet and vehicle management software URL: https://viindoo.com/blog/business-management-3/vehicle-management-software-622. (дата звернення: 29.05.2023)

8. Drivvo app review URL: https://steemit.com/travel/@raserrano/drivvo-app-review-pictures-are-mine. (дата звернення: 29.05.2023)

9. Bates B., K. Sierra Head first Java: Your Brain on Java. 2003. 656 с.

10. Stack Memory and Heap Space in Java URL: https://www.baeldung.com/java-stack-heap. (дата звернення: 29.05.2023)

11. Spring Framework Documentation URL: https://docs.spring.io/spring-framework/-reference/. (дата звернення: 29.05.2023).

12. What Is Spring Framework and Its Advantages URL: https://www.simplilearn.com/tutorials/spring-boot-tutorial/what-is-spring-framework-and-its-advantages. (дата звернення: 29.05.2023)

13. Sande J. Dart Apprentice: Fundamentals. 2022. 630 с.
14. Will Flutter and Dart be the Next Big Thing in Application Development? URL: https://medium.com/bricksnbrackets/will-flutter-and-dart-be. (дата звернення: 29.05.2023)
15. Flutter Tutorials URL: https://docs.flutter.dev/reference/tutorials. (дата звернення: 29.05.2023).
16. 8 Flutter Advantages URL: https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/. (дата звернення: 29.05.2023)
17. Firebase Advantages and Disadvantages URL: https://blog.back4app.com/firebase-advantages-and-disadvantages/. (дата звернення: 29.05.2023)
18. IntelliJ IDEA Tutorial URL: https://www.tutorialspoint.com/intellij_idea/intellij_idea_tutorial.pdf. (дата звернення: 29.05.2023).
19. Manage data sources URL: https://www.jetbrains.com/help/idea/managing-data-sources.html. (дата звернення: 29.05.2023)
20. Android Studio Guide URL: https://www.firstinspires.org/sites/default/files/uploads/resource_library/ftc/android-studio-guide.pdf. (дата звернення: 29.05.2023).
21. Overview of Google Play services URL: https://developers.google.com/android/guides/overview. (дата звернення: 29.05.2023)
22. Muschko B. Gradle in Action. 2014. 480 с.
23. Declaring dependencies URL: https://docs.gradle.org/current/userguide/declaring_dependencies.html. (дата звернення: 29.05.2023)
24. Brown M. MVVM Unleashed. 2012. 500 с.
25. Basic Coding Principles for Mobile URL: https://freaksense.com/basic-coding-principles-for-mobile-every-programmer-must-follow/. (дата звернення: 29.05.2023).

ДОДАТОК А

Інструментальні засоби формування рекомендацій енергоефективного використання автокондиціонера

Програмний код

Аркушів 13

Київ — 2023

main.dart

```
Future<void> main() async {
  /// цей метод потрібен, щоб main метод був асинхроним
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  /// ініт бази данних
  await HiveProvider.instance.init();
  /// ініт дебаггера
  await _initLogger();
  /// загрузка додатку
  runApp(const App());
}
Future<void> _initLogger() async {
  final logger = CRLoggerInitializer.instance;
  await logger.init(
    printLogs: !kReleaseMode,
    useCrLoggerInReleaseBuild: !kReleaseMode,
  );
  logger.appInfo = {'Endpoint': serverBaseURL};
}
```

constants.dart

```
const machineBaseURL = 'http://192.168.1.104:8080/';
const serverBaseURL = 'http://192.168.1.104:8081/';
```

app.dart

```
final _navKey = GlobalKey<NavigatorState>();

NavigatorState? get nav => _navKey.currentState;

class App extends StatefulWidget {
  const App({Key? key}) : super(key: key);

  @override
  State<App> createState() => _AppState();
}

class _AppState extends State<App> {
  StreamSubscription? _connectionSub;
  final _appBloc = AppBloc();

  @override
  void initState() {
    super.initState();
```

```
  /// Підписуємося на зміни стану інтернету
  _connectionSub = Connectivity()
      .onConnectivityChanged
      .listen((ConnectivityResult result) {
    if (result == ConnectivityResult.none) {
      _appBloc.add(NoInternetConnectionEvent());
    } else {
      /// Якщо підключення є, то перевіряється чи юзера авторизовано
      _appBloc.add(CheckUserStateEvent());
    }
  });
  _appBloc.add(InitializeAppEvent());
}

/// Стейт всього додатку
@override
void dispose() {
  _appBloc.close();
  _connectionSub?.cancel();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return BlocProvider(
    create: (context) => AppBloc(),

    /// Слухаємо блок
    child: BlocListener(
      bloc: _appBloc,
      listener: _onStateChange,
      child: MaterialApp(
        navigatorKey: _navKey,
        debugShowCheckedModeBanner: false,

        /// початковий екран
        initialRoute: SplashPage.routeName,

        /// Всі можливі роути
        routes: {
          SplashPage.routeName: (context) => const SplashPage(),
          SignInPage.routeName: (context) => const SignInPage(),
          SignUpPage.routeName: (context) => const SignUpPage(),
          BottomNavigationPage.routeName: (context) =>
              const BottomNavigationPage(),
          NoInternetConnectionPage.routeName: (context) =>
              const NoInternetConnectionPage(),
        },
        title: 'Car app',
```

```
        ),
      ),
    );
  }

  Future<void> _onStateChange(BuildContext _, BaseState state) async {
    if (state is ErrorState) {
      showErrorDialog(context, state);
    } else if (state is AppInitializedState) {
      _appBloc.add(SplashEndedEvent());
    } else if (state is SplashEndedState) {
      //   _appBloc.add(CheckUserStateEvent());
    } else if (state is UserState) {
      await _onUserState(state);
    } else if (state is MoveToNoInternetPage) {
      await nav?.pushNamedAndRemoveAll(NoInternetConnectionPage.routeName);
    }
  }

  /// Якщо юзер авторизован, то перекидує на основна пейжду. Якщо ні, то на сайн ін пейждц
  Future _onUserState(UserState state) async {
    if (!state.isUserLogged) {
      return nav?.pushNamedAndRemoveAll(SignInPage.routeName);
    }

    return nav?.pushNamedAndRemoveAll(BottomNavigationPage.routeName);
  }
}
```

recommendation_event.dart

```
class GetRecommendationsEvent extends BaseEvent {}
```

recommendation_state.dart

```
class GotRecommendationsState extends BaseState {
  const GotRecommendationsState(this.recommendations);

  final Recommendations recommendations;
}
```

recommendation_bloc.dart

```
class RecommendationsBloc extends BaseBloc with LogoutBlocMixin {
  RecommendationsBloc() {
    on<GetRecommendationsEvent>(
```

```
    (event, emit) => emit.async(_getRecommendations()),
   );
  }

  final _recommendationsApi = RecommendationsApi();

  Stream<BaseState> _getRecommendations() {
   return doAsync(
    () => _recommendationsApi.getRecommendations(GetRecommendationsRequest()),
    onComplete: GotRecommendationsState.new,
   );
  }
}
```

recommendation_page.dart

```
class RecommendationsPage extends StatefulWidget {
  const RecommendationsPage({this.controller, Key? key}) : super(key: key);

  static const bottomNavIndex = 1;
  final RecommendationPageRefreshCtr? controller;

  @override
  State<RecommendationsPage> createState() => _RecommendationsPageState();
}

class _RecommendationsPageState
    extends BasePageState<RecommendationsPage, RecommendationsBloc> {
  Recommendations? _recommendations;

  @override
  RecommendationsBloc createBloc() => RecommendationsBloc();

  @override
  void initState() {
   super.initState();
   bloc.add(GetRecommendationsEvent());
   widget.controller?.addListener(_reloadRecommendations);
  }

  @override
  void dispose() {
   widget.controller?.removeListener(_reloadRecommendations);
   super.dispose();
  }

  @override
  void onRebuild(BuildContext context, BaseState state) {
   super.onRebuild(context, state);
```

```
  if (state is GotRecommendationsState) {
    _recommendations = state.recommendations;
  }
}

@override
Widget bodyWidget(BuildContext context) {
  final advises = _recommendations?.advise;

  return Scaffold(
      appBar: AppBar(
        title: const Text('Recommendations'),
        centerTitle: true,
        actions: [PopupMenu(onLogout: _logout)],
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _reloadRecommendations,
        child: const Icon(Icons.refresh),
      ),
      body: advises != null
          ? ListView.builder(
              padding: const EdgeInsets.all(16),
              itemCount: advises.length,
              itemBuilder: (context, index) {
                final advise = advises[index];

                return Stack(
                  children: [
                    Card(
                      elevation: 6,
                      margin: const EdgeInsets.symmetric(vertical: 8),
                      child: Padding(
                        padding: const EdgeInsets.all(14),
                        child: Column(
                          crossAxisAlignment: CrossAxisAlignment.start,
                          children: [
                            Text(
                              'Recommendation #${index + 1}    ',
                              style: const TextStyle(
                                fontSize: 16,
                                fontWeight: FontWeight.bold,
                              ),
                            ),
                            const SizedBox(height: 10),
                            Text(
                              advise,
                              style: const TextStyle(fontSize: 16),
                            ),
                          ],
                        ),
```

```
                    ),
                  ),
                ],
              );
            },
          )
        : const Center(
            child: Text(
              'No recommendations',
              style: TextStyle(
                fontSize: 20,
              ),
            ),
          ));
  }

  @override
  FutureOr<void> onAction(BuildContext context, ActionState state) {
    if (state is LogoutState) {
      nav?.pushNamedAndRemoveAll(SignInPage.routeName);
    }
    return super.onAction(context, state);
  }

  void _reloadRecommendations() => bloc.add(GetRecommendationsEvent());

  void _logout() => bloc.add(LogoutEvent());
}
```

BeanGarden.java

```
@Configuration
public class BeanGarden {

    @Bean
    public WebClient localApiClient() {
        return WebClient.create("http://localhost:8080");
    }
}
```

ResourceController.java

```
@RestController
public class ResourceController {

    @Autowired
    AdviceService adviceService;
```

```
    @Autowired
    CarService carService;


    @GetMapping("/recommendation")
    public Map<String, List<String>> createRecommendation(@RequestHeader(value = "Authorization") String authorizationHeader) {
        return adviceService.createRecommendation(authorizationHeader);
    }

    @GetMapping("/refresh")
    public Map<String, List<String>> refreshRecommendation(@RequestHeader(value = "Authorization") String authorizationHeader) {
        return adviceService.refreshRecommendation(authorizationHeader);
    }

    @GetMapping("/getCarInfo")
    public CarExpanded getCarInfo(@RequestHeader(value = "Authorization") String authorizationHeader) {
        return carService.getInfo(authorizationHeader);
    }
```

AuthController.java

```
@RestController
public class AuthController {

    @Autowired
    AuthService authService;

    @PostMapping("/login")
    public User login(@RequestHeader(value = "Authorization") String email) throws Exception {
        return authService.login(email);
    }

    @PostMapping("/registration")
    public User registration(@RequestBody User user) throws Exception {
        return authService.registration(user);
    }

    @GetMapping("/users")
    public Map<String, User> getUsers() {
        return authService.getUsers();
    }
}
```

Car.java

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Car {
    private Conditioner conditioner;
    private Sensor innerSensor;
    private Sensor outerSensor;
}
```

Sensor.java

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Sensor {
    private Integer temperature;
    private Integer humidity;
}
```

Conditioner.java

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Conditioner {
    private boolean status;
    private State state;
    private Integer power;
    private Integer workTemperature;
}
```

State.java

```
public enum State {
    CONDITIONING,
    CONDITIONING_VENTILATION,
    VENTILATION,
    HEATING,
    HEATING_VENTILATION,
    RECIRCULATION,
    DEMISTING}
```

BackServiceApplication.java

```
@SpringBootApplication
public class BackServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(BackServiceApplication.class, args);
    }

}
```

StateController.java

```
@Controller
public class StateController {

    @Autowired
    private CarStateService carStateService;

    @GetMapping("/")
    public String carPage(Model model) {
        model.addAttribute("title", "Машина");

        Car car = carStateService.getCache();
        model.addAttribute("conditioner", car.getConditioner());
        model.addAttribute("innerSensor", car.getInnerSensor());
        model.addAttribute("outerSensor", car.getOuterSensor());

        return "car";
    }

    @PostMapping("/setConditions")
    public void setConditions(@RequestParam Integer innerTemp) {

    }

    @PostMapping("/setInnerTemp")
    public String setInnerTemp(@RequestParam Integer innerTemp) {
        carStateService.setInnerSensorTemperature(innerTemp);
        return "redirect:/";
    }

    @PostMapping("/setInnerHumidity")
    public String setInnerHumidity(@RequestParam Integer innerHumidity) {
        carStateService.setInnerSensorHumidity(innerHumidity);
        return "redirect:/";
    }

    @PostMapping("/setOuterTemp")
```

```
    public String setOuterTemp(@RequestParam Integer outerTemp) {
        carStateService.setOuterSensorTemperature(outerTemp);
        return "redirect:/";
    }

    @PostMapping("/setOuterHumidity")
    public String setOuterHumidity(@RequestParam Integer outerHumidity) {
        carStateService.setOuterSensorHumidity(outerHumidity);
        return "redirect:/";
    }

    @PostMapping("/setOnOff")
    public String setStatus(@RequestParam(value = "on_off", required = false) String checkboxValue) {
        carStateService.setConditionerStatus(checkboxValue);
        return "redirect:/";
    }

    @PostMapping("/setFunction")
    public String setState(@RequestParam(value = "functions", required = false) String checkboxValue)
{
        carStateService.setConditionerFunctions(checkboxValue);
        return "redirect:/";
    }

    @PostMapping("/setPower")
    public String setPower(@RequestParam Integer power) {
        carStateService.setConditionerPower(power);
        return "redirect:/";
    }

    @PostMapping("/setTemperature")
    public String setTemperature(@RequestParam Integer temperature) {
        carStateService.setConditionerTemperature(temperature);
        return "redirect:/";
    }
}
```

ResourceController.java (Car service)

```
@RestController
public class ResourceController {

    @Autowired
    AdviceService adviceService;

    @Autowired
    CarService carService;
```

```
    @GetMapping("/refresh")
    public Map<String, List<String>> refreshRecommendation(@RequestHeader(value = "Authorization") String authorizationHeader) {
        return adviceService.refreshRecommendation(authorizationHeader);
    }

    @GetMapping("/getCarInfo")
    public CarExpanded getCarInfo(@RequestHeader(value = "Authorization") String authorizationHeader) {
        return carService.getInfo(authorizationHeader);
    }
```

CarStateService.java

```
@Service
public class CarStateService {
    private final Integer BEST_HUMIDITY = 50;
    private final Integer BEST_TEMPERATURE_MIN = 18;
    private final Integer BEST_TEMPERATURE_MAX = 20;

    private Cache carState;
    private Sensor innerSensor;
    private Sensor outerSensor;
    private Conditioner conditioner;

    @PostConstruct
    public void setDefaultValues() {
        Car car = new Car();
        setInnerSensor();
        car.setInnerSensor(innerSensor);
        setOuterSensor();
        car.setOuterSensor(outerSensor);
        setConditioner();
        car.setConditioner(conditioner);
        Cache cache = new Cache();
        cache.setCar(car);
        this.carState = cache;
    }

    public Car getCache() {
        return this.carState.getCar();
    }

    public void setConditioner() {
        Conditioner conditioner = new Conditioner();
        conditioner.setStatus(Boolean.FALSE);
        this.conditioner = conditioner;
    }
```

```
    public void setConditioner(boolean onoff, State state, Integer power, Integer workTemperature) {
        conditioner.setStatus(onoff);
        conditioner.setState(state);
        conditioner.setPower(power);
        conditioner.setWorkTemperature(workTemperature);
    }

    public void setInnerSensor() {
        Sensor innerSensor = new Sensor();
        innerSensor.setHumidity(BEST_HUMIDITY);
        innerSensor.setTemperature((BEST_TEMPERATURE_MAX +
BEST_TEMPERATURE_MIN)/2);
        this.innerSensor = innerSensor;
    }

    public void setInnerSensorTemperature(Integer temperature) {
        innerSensor.setTemperature(temperature);
    }

    public void setInnerSensorHumidity(Integer humidity) {
        innerSensor.setHumidity(humidity);
    }

    public void setOuterSensor() {
        Sensor outerSensor = new Sensor();
        outerSensor.setHumidity(BEST_HUMIDITY);
        outerSensor.setTemperature((BEST_TEMPERATURE_MAX +
BEST_TEMPERATURE_MIN)/2);
        this.outerSensor = outerSensor;
    }

    public void setOuterSensorTemperature(Integer temperature) {
        outerSensor.setTemperature(temperature);
    }

    public void setOuterSensorHumidity(Integer humidity) {
        outerSensor.setHumidity(humidity);
    }

    public void setConditionerStatus(String value) {
        if("on".equals(value)) {
            conditioner.setStatus(Boolean.TRUE);
        } else {
            conditioner.setStatus(Boolean.FALSE);
        }
    }

    public void setConditionerFunctions(String value) {
        conditioner.setState(State.valueOf(value));
    }
```

```
    public void setConditionerPower(Integer power) {
        conditioner.setPower(power);
    }

    public void setConditionerTemperature(Integer temperature) {
        conditioner.setWorkTemperature(temperature);
    }

    public Car getCarState() {
        return carState.getCar();
    }

    public void readStateInfo(Conditioner conditioner, Sensor innerSensor, Sensor outerSensor) {
        Car car = new Car(conditioner, innerSensor, outerSensor);
        this.carState.setCar(car);
    }
}
```

CarServiceApplication.java

```
@SpringBootApplication
public class CarServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(CarServiceApplication.class, args);
    }

}
```

ДОДАТОК Б

Інструментальні засоби формування рекомендацій енергоефективного використання автокондиціонера

Апробація результатів роботи

Аркушів 3

Київ — 2023

УДК 004.75

Студент 4 курсу, гр. ТІ-92 Яричич В.П.
Доц., к.е.н. Гусєва І.І.

ІНСТРУМЕНТАЛЬНІ ЗАСОБИ АВТОМАТИЧНОГО КОНТРОЛЮ КОНДИЦІОНЕРА ДЛЯ МАКСИМАЛЬНОЇ ЕНЕРГОЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ПАЛИВА

Кондиціонер у машині (або автокондиціонер) - це система, що призначена для забезпечення комфортної температури та вологості в салоні автомобіля. Він працює за принципом циклічного охолодження та очищення повітря, яке потрапляє до салону.

Кондиціонер складається з компресора, конденсатора, евапоратора та різних клапанів та датчиків. Компресор стискає холодоагент (зазвичай фреон), який потім перетікає через конденсатор та знижує температуру, перетворюючись з газоподібного стану у рідкий. Рідкий холодоагент потім проходить через евапоратор, де він знову перетворюється на газоподібний стан, збільшуючи при цьому об'єм і знижуючи температуру, і забезпечує охолодження повітря, яке потім надходить до салону [1].

Використання автокондиціонера може підвищити витрати палива автомобіля. Це пов'язано з тим, що він працює за рахунок двигуна, і його робота збільшує опір в повітрі, що вимагає більшої потужності, щоб забезпечити необхідний рівень комфорту в салоні.

Конкретна витрата палива залежить від декількох факторів, таких як тип та вік автомобіля, обсяг двигуна, температура зовнішнього повітря, швидкість руху, стан системи кондиціонування тощо. Зазвичай використання кондиціонера збільшує витрати палива на 5-20%, в залежності від умов експлуатації [2].

Існує помилкове враження, що функція «cruisecontrol», що наявна у багатьох автомобілів, допомагає достатньо економити паливо для того, щоб не брати до уваги інші рекомендації для енергоефективності водіння. Дана функція дуже корисна з ряду причин:

- функція дозволяє підтримувати стабільну швидкість, що зменшує кількість різких прискорень та гальмувань, що в свою чергу зменшує споживання палива;
- утримання стабільної швидкості без необхідності постійного натискання на педаль газу може допомогти зменшити втому водія та поліпшити комфорт під час дальніх поїздок.

Та дана функція не є універсальним рішенням економії, адже вона не підходить для усіх можливих типів доріг, через що існує висока імовірність частого її увімкнення та вимкнення, що разом із використанням автокондиціонера може доволі негативно відобразитися на енергоефективності.

Наразі більшість водіїв, які користуються кондиціонером, не дуже задумуються про те, як це впливає на енергоефективність їх поїздки. Це може бути пов'язано з тим, що вони вважають кондиціонер невід'ємною частиною комфортної поїздки, і не беруть до уваги його вплив на витрату палива.

Однак, деякі водії стають уважнішими до ефективного використання кондиціонера, особливо у тих регіонах, де температура дуже висока і він використовується протягом тривалого часу. Вони можуть звернутися до різноманітних джерел, таких як інтернет-форуми або блоги автомобілістів, для отримання рекомендацій щодо ефективного використання автокондиціонера [4]. Але даний підхід не дуже гнучкий, адже не може надавати рекомендації у режимі реального часу і не бере до уваги наявні, у момент використання кондиціонера, погодні умови. Дані недоліки методу роблять його певною мірою не ефективним.

Тому розробка додатку, щоб зміг у режимі реального часу слідкувати за тим, щоб використання автокондиціонеру було максимально енергоефективним є дуже корисною та актуальною.

Дана програма може дати корисні рекомендації щодо ефективного використання кондиціонера, які дозволять знизити витрати палива. Наприклад, програма може надавати рекомендації щодо того, коли варто використовувати автокондиціонер, а коли можна обійтися без нього, як правильно налаштувати температуру в салоні, які швидкості обертання вентилятора найбільш економічні тощо.

Також програма може слідкувати за витратами палива під час використання кондиціонера та аналізувати їх для надання рекомендацій щодо економного використання. У результаті, водії зможуть знизити витрати палива та використовувати можливості для комфортної поїздки ефективніше.

Отже, програма може допомогти зменшити витрати палива та зберегти кошти на заправках, а також знизити вплив автомобілів на довкілля шляхом скорочення викидів вуглекислого газу та інших шкідливих речовин.

Перелік посилань:

1. How Automotive Air Conditioning Works [Електронний ресурс] – 2021. – Режим доступу до ресурсу: https://auto.howstuffworks.com/automotive-air-conditioning.htm?srch_tag=wl3um5ohbhorgmyj6tgtvf3a4hxsgmuw.

2. The impact of heating ventilation and air conditioning systems on energy consumption and CO2 emissions [Електронний ресурс] – 2023. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0360544223005492.

3. HowCruiseControlSystemsWork [Електронний ресурс] – 2021. – Режим доступу до ресурсу: https://auto.howstuffworks.com/cruise-control.htm?srch_tag=rxslp6hr6ewjip7p4uaw2h264fyvnjo6.

Tipstoimprovefuelefficiency [Електронний ресурс] – Режим доступу до ресурсу: https://www.fueleconomy.gov/feg/driveHabits.jsp.