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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів»

Виконала:
студентка IV курсу, групи КП-93
Білоус Анна Юріївна _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Рибачок Наталія Антонівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ЕІ, к.т.н., доцент,
Вунтесмері Юрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2023 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2022 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Білоус Анні Юріївні

1. Тема проєкту «Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів», керівник проєкту Рибачок Наталія Антонівна, к.т.н., доцент, затверджені наказом по університету від «31» травня 2023 р. № 2107-с

2. Термін подання студентом проєкту «16» червня 2023 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування, технологій розроблення мобільних застосунків;
   - структурна організація розроблених програмних засобів;
   - аналіз реалізації програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   - функціональність мобільного застосунка (креслення);
   - структура бази даних (креслення);
   - структурна схема мобільного застосунка (плакат);
   - структура сторінок інтерфейсу (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2022	
2.	Розроблення та узгодження технічного завдання	25.11.2022	
3.	Розроблення структури застосунка	15.12.2022	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2022	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2023	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2023	
7.	Програмна реалізація застосунка	10.03.2023	
8.	Тестування застосунка	17.03.2023	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2023	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	12.04.2023	
11.	Підготовка графічної частини дипломного проєкту	21.04.2023	
12.	Оформлення документації дипломного проєкту	26.05.2023	

Студент Анна БІЛОУС

Керівник проєкту Наталія РИБАЧОК

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню мобільного застосунка для систематизації інформації про проходження автомобілем технічних оглядів.

Розроблене програмне забезпечення являє собою Android-застосунок, який покликаний допомогти користувачам підтримувати задовільний технічний стан свого автомобіля. Функціональність мобільного застосунка забезпечує організацію роботи з даними автомобілів та записів про проходження технічних оглядів, надання статистики витрат та знеособленої аналітики, надсилання сповіщень про майбутні технічні огляди тощо.

Інформаційна безпека мобільного застосунка реалізована за допомогою розподілу прав доступу: для роботи із застосунком користувач має зареєструватися та авторизуватися. Зареєстровані користувачі також розділяються за ролями, що вирізняються набором функцій, які користувачі певної ролі мають право виконувати, зокрема керування доступом інших користувачів до даних автомобіля та налаштування сповіщень.

У даному дипломному проєкті розроблено: архітектуру мобільного застосунка, алгоритм розмежування доступу користувачів, процедуру сповіщення користувачів, а також графічний інтерфейс користувача.

ABSTRACT

This diploma project is dedicated to the creation of a mobile application for the systematization of information about passing technical inspections by car.

The developed software is an Android application designed to help users maintain the satisfactory technical condition of their car. The functionality of the mobile application supports operations with car data and car's records of passing technical inspections, provision of expense statistics and depersonalized analytics, sending notifications about upcoming technical inspections, etc.

The information security of the mobile application is implemented using the distribution of access rights: to work with the application, the user must register and authorize. Registered users are also divided by roles, which are distinguished by a set of functions that users of a certain role have the right to perform, including managing access of other users to car data and setting up notifications.

The following structures and algorithms are developed in this project: the architecture of the mobile application, the user access restriction algorithm, the procedure of user notification, as well as graphical user interface.

ДП.045440-01-90 Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Мобільний застосунок для	5	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Технічне		
	завдання		
ДП.045440-03-81	Мобільний застосунок для	73	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Пояснювальна		
	записка		
ДП.045440-04-51	Мобільний застосунок для	4	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Програма та		
	методика тестування		
ДП.045440-05-34	Мобільний застосунок для	24	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Мобільний застосунок для	1	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Функціональність		
	мобільного застосунка.		
	UML-діаграма		
	прецедентів		
ДП.045440-07-99	Мобільний застосунок для	1	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Структура бази		
	даних. ER-діаграма		
ДП.045440-08-98	Мобільний застосунок для	1	
	систематизації інформації		
	про проходження		
	автомобілем технічних		
	оглядів. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2022 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ СИСТЕМАТИЗАЦІЇ ІНФОРМАЦІЇ ПРО ПРОХОДЖЕННЯ АВТОМОБІЛЕМ ТЕХНІЧНИХ ОГЛЯДІВ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна БІЛОУС

2022

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для збереження інформації про проходження автомобілем технічних оглядів з метою спрощення процесу контролю користувача за станом свого автомобіля.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Мобільний застосунок має забезпечувати такі основні функції:

1. Підтримка наступних ролей: Власник, Редактор, Переглядач.
2. Керування власними автомобілями (створення, перегляд, редагування, видалення) для користувачів типу Власник.
3. Збереження та керування записами про проведені ремонтні роботи та замінені деталі власних автомобілів (створення, фільтрація та пошук, перегляд, редагування, видалення) для користувачів типу Власник або Редактор.
4. Підтримка роботи з одними даними кількох типів користувачів із забезпеченням різних прав доступу (Власник, Редактор, Переглядач).

5. Представлення статистики витрат для обраних автомобілів для всіх типів користувачів.
6. Сповіщення для всіх типів користувачів про майбутні обстеження автомобілів на основі інформації про попередні проведені обстеження.
7. Представлення для всіх типів користувачів знеособленої аналітики поширених проблем, пов’язаних із певним типом автомобіля.
8. Представлення для всіх типів користувачів переліку робіт, які найчастіше проводить певна СТО, з оцінками виконання цих робіт іншими користувачами.

Системні вимоги:

1. Розробку виконати на мові програмування Kotlin.
2. Мобільний застосунок має працювати на мобільних пристроях з операційною системою Android версії 5.0. Lollipop або вище.
3. Мобільний застосунок має працювати на мобільних пристроях, які мають щонайменше 1 ГБ оперативної пам’яті та 200 МБ вільного місця на диску.

Додаткові вимоги:

1. Налаштування та персоналізація сповіщень про наступні обстеження для всіх типів користувачів.
2. Додавання фотографій та коментарів до записів про проведені обстеження для користувачів типу Власник або Редактор.
3. Дизайн інтерфейсу з використанням у якості базових білого та ціанового кольорів.
4. Підтримка темного режиму інтерфейсу із використанням у якості базових чорного та синього кольорів.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1. Пояснювальна записка.
2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
   - «Функціональність мобільного застосунка. UML-діаграма прецедентів».
   - «Структура бази даних. ER-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи..... 16.11.2022
Розроблення та узгодження технічного завдання 27.11.2022
Розроблення структури мобільного застосунка..... 15.12.2022
Розроблення дизайну сторінок та графічних елементів 03.02.2023
Програмна реалізація мобільного застосунка..... 15.03.2023
Тестування мобільного застосунка 07.04.2023
Підготовка матеріалів текстової частини проєкту 28.04.2023
Підготовка матеріалів графічної частини проєкту 12.05.2023
Оформлення технічної документації проєкту..... 25.05.2023

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ СИСТЕМАТИЗАЦІЇ ІНФОРМАЦІЇ ПРО ПРОХОДЖЕННЯ АВТОМОБІЛЕМ ТЕХНІЧНИХ ОГЛЯДІВ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна БІЛОУС

2023

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс); один зі способів взаємодії з програмним забезпеченням.

DAO – Data Access Object; об’єкт, що надає абстрактний інтерфейс до даних та реалізує певні операції без розкриття деталей бази даних.

DTO – Data Transfer Object; об’єкт, що використовується для передачі даних між підсистемами та не містить жодної бізнес-логіки.

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення); основні чотири функції керування даними.

HTTP – протокол передачі даних, що використовується в комп’ютерних мережах.

JVM – Java Virtual Machine (віртуальна машина Java); віртуальна машина для виконання байт-коду Java – скомпільованої програми, написаної мовою Java, яка містить інструкції для виконання віртуальною машиною та додаткову інформацію.

JSON – JavaScript Object Notation (нотація об’єктів JavaScript); текстовий формат представлення та обміну даними між програмними модулями, де дані представлені у вигляді пар ключ-значення.

MVC – Model-View-Controller (Модель-Вигляд-Контролер); шаблон проєктування архітектури застосунка, за яким архітектура поділяється на три взаємопов’язані частини: модель даних відповідає за керування даними та бізнес-логіку застосунка, вигляд відповідає за інтерфейс користувача, контролер перетворює вхідні дані на команди для моделі або вигляду.

MVVM – Model-View-ViewModel (Модель-Вигляд-Модель вигляду); шаблон проєктування архітектури застосунка, схожий на MVC, але замість частини контролера наявна модель вигляду, що двобічно пов’язує дані з вигляду та моделі даних, спрощуючи пряму взаємодію цих частин та дозволяючи динамічно реагувати на зміни моделі чи ввід користувача.

RESTful API – інтерфейс для взаємодії з програмним забезпеченням за допомогою протоколу HTTP та методів, зокрема GET, POST, PUT, DELETE.

UI – User Interface; користувацький інтерфейс.

XML – Extensible Markup Language; розширювана мова розмітки для текстового представлення ієрархічно структурованих даних та обміну цими даними між різними застосунками, зокрема, через Інтернет.

Мобільний застосунок – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях, що може бути завантажене з онлайн-магазинів мобільних застосунків.

ООП – об’єктно-орієнтоване програмування.

ОС – операційна система.

ПЗ – програмне забезпечення.

СТО – станція технічного обслуговування; підприємство, що надає послуги з технічного обслуговування і ремонту автомобільного транспорту.

СКБД – система керування базами даних.

Технічне обслуговування – комплекс операцій щодо підтримки працездатності або справності виробу під час використання за призначенням, зберігання та транспортування.

Техогляд – технічний огляд; процес визначення відповідності транспортного засобу встановленим до конструкції і технічного стану вимогам.

Токен авторизації – спеціально згенерований секретний ключ, призначений для ідентифікації власника токена.

Фреймворк – англ. Framework, каркас, платформа; програмне середовище, що полегшує та прискорює розробку складних систем.

ВСТУП

На сьогоднішній час українське суспільство продовжує нерівну боротьбу із окупантом, який прийшов на наші землі із загарбницькими намірами. Ця боротьба негативно вплинула на багато галузей та сфер діяльності нашої держави, зокрема на автомобільний ринок. Але попри це, попит на легкові автомобілі поступово відновився, ринок стабілізувався та не зазнав критичних втрат: відповідно до відкритих даних Міністерства внутрішніх справ України, автомобільний ринок легкових автомобілів 2022 року становить 72,6% від ринку 2021 року [1].

Легковий автомобіль став важливим надбанням щонайменше кожного п’ятого українця: станом на 2021 рік рівень автомобілізації населення України становив 245 автомобілів на 1000 осіб [2]. Для підтримки задовільного стану своїх транспортних засобів власники автомобілів зобов’язані регулярно проводити технічне обслуговування своїх авто. На жаль, не всі власники автомобілів відстежують або пам’ятають, коли їх автомобіль потребує наступного візиту до СТО, бо відсутні механізми планування та нагадування про майбутні техогляди. Ускладненим також є перегляд інформації про попередні проведені ремонтні роботи, адже власникам автомобілів необхідно шукати потрібну інформацію у паперових документах, які іноді можуть бути частково або повністю відсутні.

Не менш актуальною проблемою для власників автомобілів є оцінка витрат на проходження наступного техогляду, що також є складною задачею через відсутність єдиної бази даних усіх попередніх витрат та інструментів аналізу цих даних.

Створення програмного продукту, який дозволить власникам автомобілів у одному місці переглядати інформацію про проходження автомобілем техоглядів, оцінювати свої витрати та отримувати сповіщення про майбутні техогляди, є актуальною задачею. Таким чином дипломний проєкт присвячений створенню мобільного застосунка для систематизації інформації про проходження автомобілем техоглядів, який вирішуватиме

вищенаведені проблеми власників автомобілів та спрощуватиме процес контролю за станом автомобіля.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд проблеми, яка вирішується програмним забезпеченням

Для підтримки задовільного технічного стану свого автомобіля власники автомобілів вимушені виконувати велику кількість ручної роботи. Після кожного пройденого техогляду власники автомобілів отримують на руки акти виконаних робіт, де зазначається, які деталі було замінено та які ремонтні роботи було виконано у процесі технічного обслуговування авто. Власники авто окремо зберігають ці акти, адже у майбутньому на основі актів про виконані роботи власники автомобілів прораховують, через який термін вони мають наступного разу відвідати СТО для повторного проведення певних регулярних робіт.

Якщо змоделювати процес планування власником автомобіля майбутніх візитів до СТО для проходження автомобілем техогляду, коли власник автомобіля не використовує жодного автоматизованого рішення, то відразу виникає декілька проблем.

1. Проблеми несвоєчасного проходження техогляду:
    1.1. Існує ризик забути про необхідність проходження наступного техогляду у найближчий час.
    1.2. Якщо використовувати фізичні нотатки для нагадування про заздалегідь запланований візит до СТО – є шанс загубити нотатки та пропустити запланований візит.
    1.3. Потрібно власноруч розраховувати період між попереднім та наступним регулярним техоглядом.
    1.4. Документацію необхідно класифікувати по періодах та автомобілях.
    1.5. Необхідно зберігати та переглядати паперову документацію про попередні техогляди.
    1.6. Є ненульова імовірність того, що частина документації може бути зіпсованою або втраченою.

2. Проблеми оцінки витрат на технічне обслуговування автомобіля:
    2.1. Ручний перегляд документації про попередні техогляди та пошук у них даних про витрати, що цікавлять.
    2.2. Потрібно проводити велику кількість додаткових математичних обчислень (різниця витрат між певними періодами, сума загальних витрат на певну послугу тощо).
    2.3. При створенні окремої паперової бази витрат необхідно регулярно проводити її оновлення.
    2.4. Для прогнозування потенційних проблем для певного типу автомобіля та оцінки додаткових витрат на їх вирішення власник має проводити власне дослідження на тему того, чи є встановлені проблеми поширеними для цього типу автомобіля і наскільки висока ймовірність їх настання.
    2.5. Для розрахування вартості майбутнього техогляду власнику необхідно самостійно шукати актуальну інформацію про перелік, ціни та якість виконання ремонтних робіт на певній СТО.
    2.6. Оцінка витрат кількох автомобілів потребує повторного виконання вищенаведених дій для даних кожного з них.

Існуючі програмні засоби для систематизації інформації про проходження автомобілем технічних оглядів не надають достатньо функцій для повного вирішення наведених проблем. Часто такі рішення більше фокусуються на веденні історії та відстеженні рівня пального в автомобілі, а для збереження історії техогляду пропонують лише роботу із запрограмованими базовими сервісами (заміна масла, резини, гальмівних колодок тощо). Відповідно, для збереження повної історії проведених робіт користувач вимушений додатково окремим чином зберігати ті дані, що неможливо ввести у програму. Тому проблема залишається незмінною: для підтримки задовільного технічного стану свого автомобіля користувач такого програмного рішення вимушений виконувати багато ручної роботи.

1.2. Аналіз існуючих програмних рішень

Ринок надання послуг контролю та ведення статистики технічного стану автомобіля не новий, сформований декілька років тому, а тому більшість ринку вже давно захоплена декількома найбільшими постачальниками. З метою підтвердження унікальності мобільного застосунка, що розробляється у рамках дипломного проєкту, було проведено пошук найбільш популярних існуючих програмних рішень серед мобільних застосунків, розміщених у одному з найбільших магазинів застосунків для мобільних пристроїв із ОС Android – Google Play. На жаль, не усі мобільні застосунки, що підходять для аналізу, є доступними для завантаження на території України, тому було відібрано аналоги лише з тих застосунків, що можливо завантажити в Україні без обмежень.

Було обрано три програмні рішення, кожне з яких нижче детально розглянуто та проаналізовано, виділено їх особливості, переваги та недоліки. На наступному етапі було висвітлено ступінь вирішення проаналізованими програмними рішеннями проблем споживачів, а також було здійснено порівняння основних функцій цих рішень та власного програмного продукту. Аналіз популярних рішень, якими вже кілька років користуються сотні користувачів, дозволить врахувати існуючі переваги і недоліки та покращити програмну розробку мобільного застосунка.

1.2.1. *Simply Auto: Car Maintenance [3]*

Simply Auto – це мобільний застосунок, розроблений для ОС Android та iOS, що надає інструменти для керування даними автомобілів. Цей сервіс дозволяє зберігати дані не тільки про автомобілі та виконані ремонтні роботи, а й про окремі поїздки та інші категорії витрат (сплата штрафу, страхування, паркування, податку, послуг евакуації тощо). Також цей сервіс містить у собі журнал пального, що допомагає відстежувати візити до автозаправних станцій, рівень пального та динаміку його використання. Особливістю даного програмного рішення є інтеграція із сервісами Google,

такими як Google Account для авторизації, Google Drive для резервного копіювання даних користувача та Google Assistant для заповнення окремих полів даних (значення пробігу автомобіля на одометрі, назва СТО, де виконувалися ремонтні роботи, примітки до запису тощо) за допомогою голосового вводу. Користувачі застосунка, розробленого для ОС iOS, мають змогу авторизуватися за допомогою акаунту Apple.

Мобільний застосунок Simply Auto має модель розповсюдження freemium: користувачу доступні основні функції безкоштовно. Також доступна платна версія застосунка Gold Version за одноразову оплату, що надає користувачу вичерпний перелік додаткових функцій та збільшує ліміти для певних основних функцій. До того ж, користувач має можливість придбати на рік підписку Platinum, що дозволить йому отримати ще більше додаткових функцій та зняти встановлені обмеження із певних основних функцій. Користувачі, що придбали будь-яке платне розширення та авторизувалися за допомогою акаунту Google або Apple, також мають можливість працювати із власними даними за допомогою вебсайту. Для підприємств, що володіють автопарком, існує окрема версія застосунка Simply Fleet, що підтримує більшість функцій звичайної версії застосунка, але оптимізована для роботи із даними великої кількості автомобілів.

Наведені нижче переваги та недоліки сформульовані для версії застосунка, розробленого для ОС Android. Тестування версії для ОС iOS не відбувалося, адже у рамках дипломного проєкту розробляється нативний мобільний застосунок (тобто лише для ОС Android), тому дослідження застосунків, розроблених під ОС iOS, є поза рамками поточного аналізу.

Переваги Simply Auto:

1. Окреме вікно зі статистикою із можливістю фільтрування по періодах та налаштування даних, які відображати на екрані.
2. Створення резервної копії у форматі CSV та відновлення даних із файлу резервної копії можливі як з локального файлу, так і з файлу, збереженого на прив’язаний диск Google Drive.

3. Наявність попередньо запрограмованого переліку виконаних робіт (заміна акумулятора, мастила, свічок запалювання тощо) та інших видів витрат, які можливо розширювати власними видами робіт та витрат.
4. Підтримка роботи із даними кількох автомобілів.
5. Зручне бічне меню, що дозволяє легко переходити між окремими журналами даних.
6. Наявність світлої та темної теми екрану.
7. Переклад щонайменше 20 мовами, зокрема українською.

Недоліки Simply Auto:

1. Якість перекладу низька: для більшості мов значна частка термінів відображається англійською, а перекладена частина не завжди перекладена правильно.
2. Уся статистика відображається у єдиному вікні, відсутня можливість переглянути статистику для окремих категорій (наприклад, лише витрати пального).
3. Безкоштовна версія дозволяє додавати максимально 10 різних видів виконаних робіт до одного запису про пройдений техогляд, аналогічне обмеження наявне для записів про інші види витрат.
4. Неможливо ввести дані про замінені деталі.
5. Нагадування про майбутні техогляди потрібно задавати вручну.
6. Підтримка роботи кількох користувачів із даними одного автомобіля доступна лише у платних версіях застосунка.

1.2.2. *AUTOsist Fleet Maintenance App [4]*

AUTOsist – це програмне забезпечення для обслуговування автопарку, що дозволяє ефективно керувати даними автомобілів автопарку. Користувач може користуватися можливостями AUTOsist за допомогою вебсайту або мобільного застосунка, що розроблений для ОС Android та iOS. Основними споживачами ПЗ AUTOsist є підприємства, що володіють

автопарком, але це ПЗ також підтримує режим роботи для власників приватних автомобілів. Окрім режиму роботи для власників приватних автомобілів, AUTOsist також підтримує розподілений доступ до спільних даних двох груп користувачів – працівників та власників підприємства, що володіють автопарком, – та має два окремих режими роботи для цих груп. Особливістю даного ПЗ є можливість авторизації у системі за допомогою акаунтів Facebook або Google+.

ПЗ AUTOsist надає обмежені можливості у безкоштовній версії свого продукту: користувачі можуть керувати даними лише одного автомобіля. Але головне обмеження безкоштовної версії – це кількість хмарного сховища, виділеного для даних автомобіля, яке при тривалому використанні ПЗ неодмінно заповниться даними, і користувач буде вимушений придбати платну підписку для продовження роботи. Після реєстрації користувачам доступний пробний період у 14 днів, під час якого вони мають доступ до повного набору функцій. По закінченню пробного періоду для продовження роботи із розширеним набором функцій необхідно обрати один з трьох доступних планів підписки. Вагомим недоліком пропонованої системи підписки є тарифікація не тільки за термін користування, а й за кількість автомобілів, що має користувач, тож для власників кількох приватних автомобілів ціна використання може бути зависокою.

Для цього ПЗ також відбувалося тестування лише версії для ОС Android, адже оцінка роботи вебсайту та мобільного застосунка для ОС iOS є поза рамками аналізу для даного дипломного проєкту.

Переваги AUTOsist Fleet Maintenance App:

1. Можливість систематизувати різноманітну інформацію для необмеженої кількості автомобілів: дані про деталі на складі, поїздки, записи про проходження техогляду, документація (страхування, посвідчення водія тощо), витрати пального тощо.
2. Створення власником автомобіля форм із переліком ремонтних робіт, які мають бути виконані на наступному техогляді, та

передача цих форм іншим користувачам, що мають доступ до даних автомобіля. Користувачі, що отримали ці форми, можуть заповнювати їх інформацією про те, чи були зазначені ремонтні роботи виконані у повному обсязі.

3. Автоматичне вирахування вартості пройденого техогляду як суми цін замінених деталей.
4. Автоматичне відстеження поїздок та заповнення даних про поїздки за допомогою технології GPS.
5. Цілодобова підтримка клієнтів телефоном та електронною поштою, що включена у вартість усіх платних підписок.

Недоліки AUTOsist Fleet Maintenance App:

1. Не найоптимальніше рішення для власника приватного автомобіля: перенасиченість різноманітними функціями, значна частка з яких направлені на підприємства, що володіють автопарком, та висока ціна користування.
2. Продукт доступний лише англійською мовою.
3. Нагадування про майбутні техогляди потрібно задавати вручну.
4. Форма із переліком ремонтних робіт, виконаних під час техогляду, містить лише попередньо запрограмований перелік робіт та не дозволяє додавати нові роботи.
5. При заповненні запису про пройдений техогляд замінені деталі можливо обрати лише з попередньо створеного переліку у розділі «Інвентар», і тільки якщо їх кількість у інвентарі достатня.

1.2.3. *Drivvo [5]*

Drivvo – це система управління окремими транспортними засобами та автопарками, що надає повний контроль над даними транспортних засобів. Доступ до системи Drivvo організований через вебсайт або мобільний застосунок, розроблений для ОС Android та iOS. Неавторизований користувач може використовувати обмежений функціонал, але для

необмеженого використання усіх функцій безкоштовної версії необхідна реєстрація у системі або авторизація із використанням акаунту Facebook або Google. Як і попередньо розглянуті програмні продукти, окрім записів про пройдені техогляди Drivvo дозволяє зберігати й інші дані, наприклад витрати на пальне, паркування та страхування.

Споживчий сегмент системи охоплює як власників приватних автомобілів, так і підприємства, що володіють автопарком. Для власників приватних автомобілів є лише один вид платної підписки Pro, а для підприємств, що володіють автопарком, передбачено чотири види підписок, що відрізняються кількістю автомобілів, кількістю користувачів, яким можливо надавати доступ до даних, та обсягом додаткових даних, які можна прикріплювати до записів, як-от фотографії актів виконаних робіт, квитанції на пальне тощо. Особливістю Drivvo є можливість придбати підписку за найпоширеніші криптовалюти, такі як Bitcoin та Ethereum.

Переваги Drivvo:

1. Детальні та різноманітні графіки, діаграми й таблиці, що можливо відображати для кожного автомобіля окремо. Наявна фільтрація даних для відображення та періоду, за який представляти результати.
2. Додавання доходів (наприклад, оплата за поїздку для таксистів).
3. Ігровізація процесу використання застосунка за допомогою розділу із нагородами й досягненнями.
4. Попереднє нагадування про заплановані події при досягненні певної умови (наприклад, за місяць до вказаної дати події).
5. Переклад щонайменше 60 мовами, зокрема українською.

Недоліки Drivvo:

1. Як і для застосунка Simply Auto, якість перекладів є низькою.
2. Багато функцій, що у аналогічних продуктах доступні у безкоштовній версії, є платними: резервне копіювання у хмарне

сховище, синхронізація між пристроями, додавання вкладених файлів та фотографій до записів.

3. Безкоштовна версія не дозволяє надавати доступ до даних автомобілів іншим користувачам, а версія Pro підтримує лише одного додаткового користувача.
4. Будь-які нагадування у системі потрібно задавати вручну.
5. Записи про техогляд хоч і дозволяють додавати дані про проведені ремонтні роботи та замінені деталі, але не розділяють ці два види даних як окремі категорії.

1.3. Результати аналізу

Для більш якісного аналізу розглянутих вище програмних продуктів та відображення ступені вирішення ними проблем користувачів, сформульованих у даному дипломному проєкті, пропонується скласти порівняльну таблицю. Критерії порівняння було сформовано на основі вимог до програмного продукту дипломного проєкту. Для деяких критеріїв, які притаманні усім розглянутим програмним продуктам, було прийнято рішення накласти певні додаткові обмеження. Це допоможе у власному програмному продукті визначити та встановити на аналогічні функції оптимальні ліміти, які позитивно сприйме користувач.

Наведемо пояснення щодо обраних критеріїв порівняння:

1. *Україномовна версія.* Система передбачає зміну мови інтерфейсу на українську.
2. *Безоплатна організація роботи з кількома автомобілями.* Система безоплатно надає можливість працювати з даними більш ніж одного автомобіля (додаткове обмеження накладене, так як усі розглянуті системи в цілому підтримують роботу з даними щонайменше двох різних автомобілів).
3. *Необмежений розподілений доступ до даних користувача.* Система дозволяє користувачу керувати доступом необмеженої кількості

інших користувачів до даних своїх автомобілів (додаткове обмеження накладене, так як усі розглянуті системи в цілому надають користувачеві можливість надавати доступ до даних щонайменше одному користувачеві).

4. *Розділення інформації про виконані ремонтні роботи та замінені деталі.* Система дозволяє у записах про проведені техогляди вказати додаткову інформацію окремо для переліку виконаних ремонтних робіт та для переліку замінених деталей.
5. *Безоплатна персоналізація записів про техогляд.* Для будь-яких записів про техогляд система безоплатно дозволяє користувачу додавати будь-яку кількість власних варіантів виконаних ремонтних робіт та замінених деталей (додаткове обмеження накладене, так як усі розглянуті системи в цілому дозволяють додавати власні види ремонтних робіт або замінених деталей).
6. *Виділення складових витрат.* Система дозволяє вводити як загальну вартість пройденого техогляду, так і ціну кожної ремонтної роботи та заміненої деталі окремо.

Результат порівняння описаних вище програмних рішень за вказаними критеріями представлено у табл. 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих рішень

Критерій	Аналоги		
	Simply Auto	AUTOsist	Drivvo
1	2	3	4
Україномовна версія	+	–	+
Безоплатна організація роботи з кількома автомобілями	+	–	+
Необмежений розподілений доступ до даних користувача	+	+	–

Продовження табл. 1.1

1	2	3	4
Розділення інформації про виконані ремонтні роботи та замінені деталі	–	+	–
Безоплатна персоналізація записів про техогляд	–	+	+
Виділення складових витрат	–	+	–

З порівняльної таблиці можна зробити висновок, що мобільний застосунок Simply Auto надає україномовним користувачам найбільше переваг для розподіленої роботи групи користувачів з різноманітними даними кількох автомобілів. Це єдине з порівнюваних ПЗ, яке у безкоштовній версії дозволяє користувачу мати необмежену кількість власних автомобілів та дозволяти необмеженій кількості інших користувачів працювати з даними цих автомобілів. Головним недоліком Simply Auto є відсутність поділу інформації про виконані ремонтні роботи та замінені деталі. Власнику автомобіля доведеться записувати усі значення підряд або створювати декілька записів про техогляд для штучної структуризації даних.

Мобільний застосунок AUTOsist пропонує найбільші можливості для систематизації інформації про проведені техогляди. Головна його перевага – це окреме виділення даних про виконані ремонтні роботи та замінені деталі. Однак персоналізація цих записів хоч і доступна, але є обмеженою та залежить від інших даних (вище при аналізі цього застосунка було вказано, що замінені деталі можливо обрати лише з розділу «Інвентар», якщо вони там є у потрібній кількості). Також це єдина система з розглянутих, що не має української версії інтерфейсу, бо розроблена виключно англійською.

Мобільний застосунок Drivvo представляє собою зручний інструмент для систематизації різноманітної інформації про автомобілі користувача. Через спробу об’єднати усі види записів у одному застосунку Drivvo надає мало спеціалізованих під записи про техогляд функцій. Тим не менш, ця система єдина дозволяє користувачу вільно налаштовувати вміст даних про виконані ремонтні роботи та замінені деталі, незважаючи на те, що вона, як і Simply Auto, не поділяє їх на окремі категорії.

Враховуючи результати аналізу існуючих рішень проблеми систематизації інформації про проходження автомобілем технічних оглядів, можемо зробити висновок, що жодне з цих рішень повністю не вирішує поставлену проблему та цілком не відповідає поставленим до дипломного проєкту вимогам. Отже, було прийнято рішення про розроблення мобільного застосунка, що враховуватиме переваги та недоліки вищеописаних застосунків-аналогів та реалізовуватиме поставлені вимоги.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1. Переваги нативного мобільного застосунка над іншими типами мобільних застосунків

За даними сервісу вебаналітики StatCounter [6], впродовж останніх 10 років найпопулярнішою ОС для мобільних пристроїв є ОС Android (рис. 1). Станом на березень 2023 року цю ОС мають 70,88% мобільних пристроїв на ринку. Друге місце за популярністю посідає ОС iOS, яку на березень 2023 року мають 28,42% мобільних пристроїв.

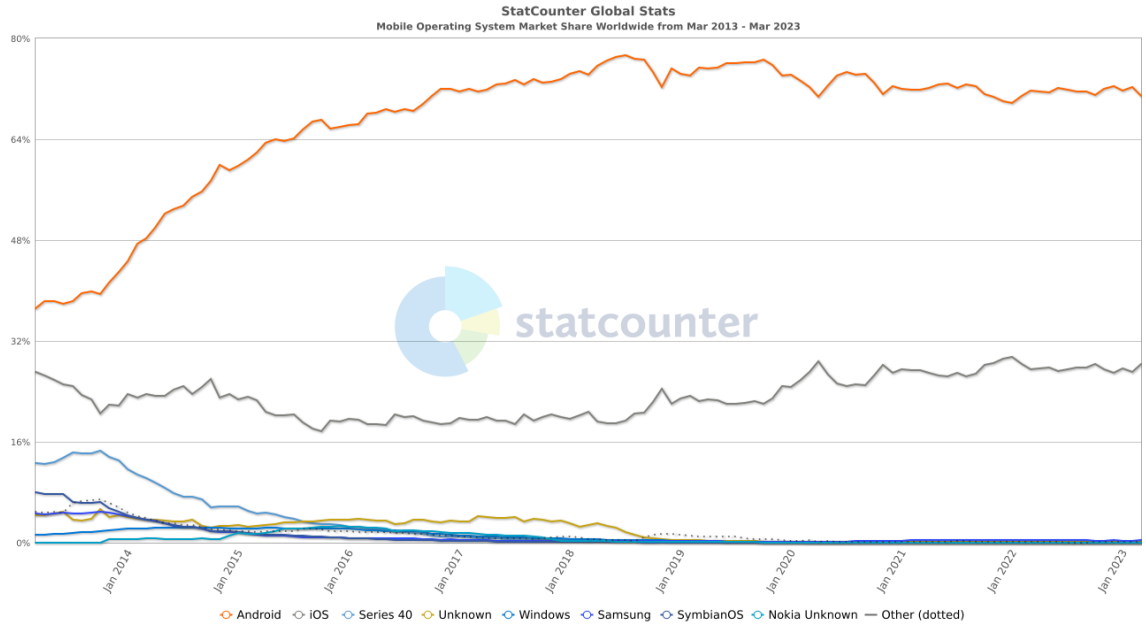


Рис. 1. Графік популярності мобільних пристроїв залежно від операційної системи за період березень 2013 – березень 2023

З графіку очевидно, що для першого виходу продукту на ринок з метою формування користувацької бази більша доля ринку буде охоплена, якщо орієнтуватися на користувачів мобільних пристроїв з ОС Android. Після успішного становлення на ринку для розширення кількості користувачів можлива розробка версії застосунка для мобільних пристроїв з ОС iOS. Класичним підходом до розробки програмних застосунків є

створення окремої версії застосунка під певну операційну систему, але залежно від вимог до програмного продукту розробку можливо зробити частково або повністю універсальною для декількох платформ. Пропонується окремо розглянути переваги та недоліки трьох основних типів мобільних застосунків: нативні застосунки, мобільні вебзастосунки, гібридні застосунки.

2.1.1. *Нативний застосунок*

Нативний застосунок – це мобільний застосунок, який написаний «рідною» для певної платформи мовою програмування та який встановлюється на пристрій за допомогою магазинів застосунків.

Переваги нативних застосунків:

1. *Вільне використання вбудованих інструментів.* За допомогою системних викликів розробник у повній мірі може використовувати будь-які ресурси пристрою: гіроскоп, системні та власні жести, модулі GPS та NFC, Bluetooth, списки контактів, сканери відбитків пальців, акселерометр тощо.
2. *Висока надійність та швидкість роботи.* Використання ресурсів пристрою є максимально оптимізованим.
3. *Використання вбудованої системи сповіщення.*
4. *Зручна система встановлення та оновлення.* Поширення нативних застосунків відбувається за допомогою спеціальних магазинів застосунків, таких як Google Play та Apple App Store, а оновлення встановлених застосунків є автоматизованим та зручним: здебільшого від користувача потребується лише натиснути на одну кнопку для швидкого оновлення застосунка.
5. *Покращений користувацький досвід.* Нативні застосунки використовують вбудовані інструменти для створення UI, що забезпечує більш оптимізовану взаємодію з користувачем.

6. *Розвинене середовище розробки.* Розробникам нативних застосунків доступний широкий вибір інструментів для налагодження програми, оцінки використання ресурсів пристрою та оптимізації програмного коду. До того ж, розробники мають широку підтримку від власників платформи та інших розробників, а також їм доступний великий обсяг спеціалізованої документації.

Недоліки нативних застосунків:

1. *Необхідність повторної розробки для іншої платформи.* Для розроблення однієї версії застосунка для кількох різних платформ необхідно з нуля повністю створювати окрему версію застосунка для кожної цільової платформи.
2. *Вартість розробки та підтримки.* Розробка та подальша підтримка нативних застосунків коштує значно дорожче, ніж для інших видів застосунків, особливо для кількох різних платформ.
3. *Систематичне оновлення.* Для використання останньої версії застосунка користувачі повинні завантажити та перевстановити версію застосунка, що містить оновлення.

2.1.2. *Мобільний вебзастосунок*

Мобільний вебзастосунок – це будь-який веб-орієнтований застосунок, адаптований для перегляду та роботи на мобільних пристроях за допомогою сумісного веббраузера.

Переваги мобільних вебзастосунків:

1. *Вільний доступ з різних пристроїв.* Доступ до застосунка можливий з будь-якого мобільного пристрою: для роботи потрібні лише веббраузер, який зазвичай є вбудованим для більшості мобільних пристроїв, та доступ до мережі Інтернет.
2. *Незалежність від ОС пристрою.* Використання веббраузерів для роботи застосунка виключає необхідність розробнику підлаштовувати систему під конкретну ОС.

3. *Не займають додаткового місця на жорсткому диску.* Так як доступ забезпечується за допомогою веббраузера, то застосунок не потребує окремого встановлення для використання.
4. *Не потребують ручного оновлення користувачами.* Оновлення застосунка відбувається на стороні сервера та автоматично відображається всім користувачам застосунка.
5. *Швидка та недорога розробка.* Не потрібно витрачати додаткові час та кошти на розробку кількох версій застосунка для різних платформ, адже одна версія вебзастосунка є універсальною.

Недоліки мобільних вебзастосунків:

1. *Сумісність з браузером.* Деякі браузери та версії браузерів можуть бути несумісними для роботи з мобільним вебзастосунком.
2. *Обмеженість роботи без мережі Інтернет.* Без підключення до мережі Інтернет більшість мобільних вебзастосунків надають користувачу урізаний набір функцій, а деякі вебзастосунки без підключення до мережі Інтернет не здатні працювати взагалі.
3. *Безпека даних.* Зазвичай дані користувачів знаходяться у хмарному сховищі, а не локально, і користувачі жодним чином не можуть контролювати методи захисту своїх даних у хмарі.
4. *Обмеженість використання ресурсів пристрою.* Для таких ресурсів пристрою, як-от модуль NFC чи гіроскоп, методи взаємодії залежать від конкретної ОС пристрою.

2.1.3. *Гібридний застосунок*

Гібридний застосунок – це мобільний застосунок, що являє собою поєднання нативних та мобільних вебзастосунків.

Переваги гібридного застосунка:

1. *Поєднання основних переваг мобільних вебзастосунків та нативних застосунків.* Гібридні застосунки підтримують доступ до ресурсів

пристрою, як це дозволяють нативні застосунки, та є кросплатформними, як мобільні вебзастосунки.

2. *Швидка та недорога розробка.* Розробка однієї кросплатформної версії застосунка із використанням можливостей нативного застосунка залишається швидшою та дешевшою за повну розробку аналогічного нативного застосунка навіть для однієї ОС.
3. *Ширша спільнота розробників.* Через простоту використання популярних вебфреймворків для створення гібридних застосунків поріг входу для розробників є низьким.

Недоліки гібридного застосунка:

1. *Поєднання основних недоліків мобільних вебзастосунків та нативних застосунків.* Як і нативні застосунки, гібридні також потребують встановлення та періодичного оновлення зі спеціальних магазинів застосунків. На додачу гібридні застосунки, так само як мобільні вебзастосунки, мають обмежену роботу без підключення до мережі Інтернет.
2. *Нижча швидкість роботи, ніж у нативних застосунків.*

2.1.4. *Результати аналізу розглянутих типів застосунків*

У результаті аналізу розглянутих вище переваг та недоліків трьох типів мобільних застосунків було прийнято рішення обрати для розробки дипломного проєкту мобільний застосунок нативного типу. Головна причина такого вибору є можливість легко керувати системними сповіщеннями, що є однією з основних вимог до програмного продукту. Також на рішення вплинули такі переваги нативного застосунка, як можливість створення кращого користувацького досвіду, розвинене середовище розробки із широкою підтримкою спільнотою і найкращі показники надійності та швидкості роботи з-поміж усіх розглянутих типів застосунків. Ці переваги допоможуть створити мобільний застосунок, що

оптимально використовує системні ресурси, надає користувачу максимальний захист даних та легку взаємодію із застосунком.

Додатково на вибір вплинула популярність ОС Android на мобільних пристроях, а особливо можливість інтеграції мобільного пристрою із стереосистемами певних моделей автомобілів за допомогою Android Auto [7]. Кількість систем та автомобілів, що підтримують Android Auto, поступово збільшується, тож модифікація застосунка для підтримки роботи з дисплея автомобіля є одним із перспективних напрямів розвитку продукту.

2.2. Вибір мови програмування та технологій для розроблення серверної частини

Для забезпечення захисту даних користувача від стороннього впливу та розподіленого доступу кількох користувачів до певних даних необхідно, аби кожен з користувачів працював із однаковою актуальною версією даних, що знаходиться у просторі, захищеному від впливу неавторизованих користувачів. Для цього дані мають зберігатися не на пристрої користувача, а в загальній базі даних у захищеному просторі. Для контролю доступу до бази даних має бути створений окремий програмний модуль, що отримуватиме запити від клієнтів, оброблюватиме їх та виконуватиме взаємодію із базою даних. Описана архітектура застосунка відповідає класичній клієнт-серверній архітектурі.

Основні задачі серверної частини мобільного застосунка, як вже було наведено вище, полягають у обробці запитів від клієнтської частини, роботі з базою даних та виконанні бізнес-логіки. Для розробки серверної частини мобільних застосунків існує багато архітектурних підходів на різних мовах програмування. Пропонується проаналізувати найбільш популярні мови програмування для розроблення серверної частини мобільних застосунків: Java, Kotlin, JavaScript та Python.

2.2.1. *Java [8]*

Java є однією з найпопулярніших мов програмування для написання серверної частини різних типів застосунків. Java є строго-типізованою мовою програмування та базується на принципах ООП. Водночас Java являє собою і платформу для розробки різноманітних застосунків та систем. Найпопулярнішою технологією для розробки серверної частини клієнт-серверних застосунків на Java є Spring projects – набір фреймворків для вирішення різноманітних задач, такі як Spring Data для роботи із базами даних та Spring Boot для створення мікросервісів і вебзастосунків.

Мова програмування Java має наступні властивості:

1. *Об'єктно-орієнтованість.* Основний підхід до представлення структурних елементів у Java є екземпляри класів.
2. *Висока продуктивність.* Програма, написана на Java, компілюється у бінарний байт-код, який під час виконання транслюється у машинний код.
3. *Кросплатформність.* Програма, написана на Java та скомпільована у байт-код, однаково виконається на будь-якому пристрої, на якому встановлена JVM.
4. *Надійність.* Java самостійно виділяє пам'ять для ресурсів програми, а також перевіряє час компіляції та виконання.
5. *Багатопотоковість.* Java підтримує паралельні обчислення (одночасне виконання кількох процесів на різних потоках), що дозволяє одночасно оброблювати запити декількох користувачів.
6. *Динамічність.* Java динамічно завантажує будь-які необхідні для роботи бібліотеки, що дозволяє легко адаптувати програму до мінливого середовища.
7. *Безпечність.* Java має вбудований захист від вірусів та несанкціонованого втручання.

2.2.2. *Kotlin [9]*

Kotlin, як і Java, також є статично типізованою мовою програмування, що працює поверх JVM та є сумісною з мовами Java та JavaScript. Kotlin є повністю сумісною із будь-якими фреймворками на основі Java, зокрема Spring projects, але також існують фреймворки, оптимізовані саме під Kotlin. Наприклад, для написання серверної частини на Kotlin найпопулярнішим фреймворком є Ktor, розроблений авторами мови Kotlin.

Напочатку мова програмування Kotlin задумувалася як «вдосконалення» та «розширення» для мови Java. Дійсно, Kotlin виправляє низку основних проблем Java, а саме:

1. Для запобігання виникненню помилки доступу за нульовим посиланням (NullPointerException) було додано типи, що допускають значення NULL (nullable type).
2. Було виключено «сирі типи» (raw type): для створення екземпляру узагальненого класу необхідно вказати конкретний тип даних, що зберігаються у цьому класі.
3. За замовчуванням колекції даних, такі як масив Array або список List, є незмінними.
4. Синтаксис мови було спрощено, що дозволяє уникнути значної кількість шаблонного та повторюваного коду.
5. Було додано можливість створювати методи та процедури на глобальному рівні поза межами класів.

До того ж, Kotlin розширює можливості Java власними функціями, які допомагають розробити ефективну серверну частину застосунка:

1. Наявність співпроцедур (coroutines), що створюють асинхронні програми без використання потоків, дозволяє робити сервери, що легко масштабуються до величезної кількості клієнтів із помірними вимогами до обладнання.

2. Kotlin може автоматично перетворювати вже написаний код Java на код Kotlin, що дозволить поступово мігрувати великі проєкти та зменшити кількість коду, що потребує підтримки.
3. Підтримка функціонального програмування у вигляді можливості створення лямбда-виразів.
4. Kotlin також підтримує створення функцій-розширення для будь-яких класів та інтерфейсів – як власних, так і системних.
5. Автоматичний неявний вивід типів значень у виразі на момент компіляції, якщо тип не вказаний явно.
6. Можливість перевантаження операторів.
7. Підтримка дата класів, що дозволяють створювати класи виключно для зберігання набору даних.

2.2.3. *JavaScript [10]*

JavaScript (JS) найчастіше використовується для повної розробки вебзастосунків, але також є популярним рішенням для створення серверів будь-яких інших типів застосунків. Мова JavaScript реалізує стандарт ECMAScript, є динамічною та багатопарадигмовою мовою: вона підтримує прототипний та імперативний стилі програмування, а також частково підтримує функціональний стиль. Хоча ця мова здебільшого використовується для виконання коду у браузері на стороні клієнта, за допомогою популярного середовища Node.js можна створювати серверну частину застосунків за межами браузера. Поверх Node.js розроблений фреймворк Express.js, що широко використовується для управління серверами та створення API.

Основні переваги мови програмування JavaScript як мови для написання серверної частини застосунка:

1. Замість багатопотоковості JavaScript використовує асинхронну модель виконання, яка дозволяє ефективно максимізувати використання ресурсів сервера.

2. Спільнотою програмістів розроблена велика кількість бібліотек та інструментів, що дозволять спростити процес розробки.
3. Відсутність строгих правил до написання архітектури дозволяє розробнику самостійно обирати, які шаблони програмування використати для вирішення поставленої задачі.
4. JavaScript є інтерпретованою мовою програмування та може виконуватися будь-де, де встановлено інтерпретатор мови.
5. Динамічна типізація дозволяє розробнику взагалі не вказувати типи змінних при ініціалізації.
6. Необхідність виконання коду на JavaScript у браузері клієнта сприяла тому, що мова JavaScript є досить швидкою.

2.2.4. *Python [11]*

Python, як і JavaScript, є високорівневою інтерпретованою мовою програмування із виключно динамічною типізацією. Найбільшого поширення Python набула у галузях Data Science та Machine Learning завдяки розвинутим можливостям роботи із великими даними (Big Data) та широкому вибору науково-числових бібліотек. Але простий синтаксис, схожий на природню мову, та легкість опановування дозволили Python стати популярним рішенням для створення будь-яких продуктів: ігор, настільних застосунків, скриптів вебсайтів та вебзастосунків. Для створення серверної частини клієнт-серверної архітектури застосунка найбільш поширеними є фреймворки Django та Flask.

Особливості Python:

1. Python має велику стандартну бібліотеку, яка для програмування більшості задач є достатньою, та широкий вибір сторонніх бібліотек, які легко завантажуються та підключаються.
2. Як і у випадку JavaScript, код на Python може виконуватися на будь-якій системі, що має встановлений інтерпретатор мови.
3. Також, як і JavaScript, Python має динамічну типізацію.

4. Мові Python властиві модульність та масштабованість, що дозволить розробнику легко розбити код на окремі модулі та спростити їх подальшу підтримку.
5. Простота вивчення, інтуїтивно зрозумілий синтаксис та спеціальні фреймворки дозволили Python стати поширеним рішенням для написання автоматизованих тестів.
6. Python має широку активну спільноту розробників.

2.2.5. *Результати аналізу*

Після аналізу популярних мов розробки серверної частини мобільних застосунків для розробки серверу дипломного проєкту було обрано мову програмування Kotlin. Кожна з розглянутих мов має свої переваги у написанні серверної частини мобільних застосунків, але тільки мови Java та Kotlin підтримують явну типізацію, оптимізують використання ресурсів сервера та захищені від зовнішнього втручання.

У виборі між Java та Kotlin перевага була надана останній мові, тому що вона має більш читабельний код, який легше надалі підтримувати, потребує менше програмного коду для реалізації певної функції, розширює можливості мови Java та вирішує частину її основних проблем.

Як основна технологія для розроблення серверної частини мовою програмування Kotlin був обраний фреймворк Ktor [12], який дозволяє швидко створити і налаштувати доступний API, а також максимально просто запустити сервер для прослуховування запитів клієнта.

2.3. Вибір СКБД та технологій взаємодії з обраною мовою програмування

Аналіз вимог до мобільного застосунку дозволив встановити тип бази даних, яку необхідно обрати, та висунути перелік вимог до СКБД, що має використовуватися серверною частиною.

Основний елемент даних застосунка – записи про техогляд – не має чітко визначеної структури: звісно, є перелік полів, обов'язкових для заповнення, але так само записи можуть не мати значень для певних необов'язкових полів. До того ж, записи про техогляди мають вкладені дані, такі як інформація про проведені роботи та замінені деталі. У реляційній базі даних важко працювати з великою кількістю необов'язкових полів: схема бази даних є визначеною, тож прибрати порожні поля для частини записів неможливо, а збереження порожніх полів витрачає місце на диску. Так само складно реалізувати ефективні збереження та запити вкладених даних: дані запису, дані про проведені роботи та дані про замінені деталі – це три окремих таблиці, які потребуватимуть частого поєднання операцією JOIN, що є повільною та затратною для великих таблиць.

Отже, реляційна база даних не є оптимальним рішенням для реалізації дипломного проєкту. Натомість документо-орієнтована база даних має ряд переваг, що вирішують поставлену задачу:

1. Документи представлені як сукупність наборів ключ-значення, де значення може бути будь-яким, зокрема й іншим документом. Ця властивість вирішує проблему збереження вкладених даних.
2. При запиті до конкретного документа можливо отримати доступ як до його звичайних полів, так і окремо до кожного поля вкладених документів. Ця властивість вирішує проблему неефективних запитів до даних.
3. Документи не мають чіткої структури, тож за потреби до кожного документа можна додавати нові поля чи видаляти непотрібні. Ця властивість вирішує проблему неефективної роботи із необов'язковими полями.

Пропонується обрати СКБД із двох найбільш поширених документо-орієнтованих СКБД: MongoDB та Amazon DynamoDB.

2.3.1. *MongoDB [13]*

MongoDB – це документо-орієнтована СКБД, що зазвичай використовується для зберігання великих обсягів даних. Дані у MongoDB представлені у вигляді BSON (Binary JSON) – формат даних, що є розширенням формату JSON і додає бінарні типи даних та шифрування.

Основні переваги MongoDB:

1. Підтримка найбільшої кількості мов програмування: інструменти для роботи з MongoDB представлені для більш ніж 30 мов програмування, зокрема для Kotlin.
2. MongoDB має безкоштовні та платні рішення для розміщення як на локальній машині, так і у хмарних середовищах.
3. Висока доступність даних завдяки вбудованій реплікації та стійкості до відмови.
4. Використання індексів сприяє швидкому пошуку та додаванню документів, які можуть зберігати до 16 МБ даних кожен.
5. Легке горизонтальне масштабування за рахунок вбудованого шардування даних (збереження даних на декількох машинах).

СКБД MongoDB притаманні такі недоліки:

1. Розробник повинен власноруч встановлювати, налаштовувати та підтримувати базу даних.
2. За замовчуванням MongoDB вимикає процес автентифікації, що може призвести до проблем безпеки даних.
3. Некоректне налаштування та використання індексів призводить до повільного виконання запитів.
4. MongoDB не підтримує транзакції та операцію JOIN.

2.3.2. *Amazon DynamoDB [14]*

Amazon DynamoDB – це нереляційна база даних, яка пропонується Amazon.com як одна зі служб Amazon Web Services (AWS). Дані у DynamoDB представлені у вигляді колекцій документів, що містять

атрибути виду «ключ-значення», та зберігаються у логічних контейнерах – таблицях. Також DynamoDB підтримує збереження даних у форматі JSON.

Наведемо переваги Amazon DynamoDB:

1. Спрощений процес налаштування, який позбавляє розробника необхідності ручного керування інфраструктурою.
2. Будь-який доступ до даних має бути наданий явно за допомогою інших сервісів AWS, що робить базу даних DynamoDB безпечною за замовчуванням.
3. Розробники можуть не слідкувати за кількістю вільного місця на диску: DynamoDB не має обмежень на максимальний обсяг даних, що зберігаються у базі, та має вбудовану автоматичну масштабованість, що керується AWS.
4. Доступність бази даних забезпечується автоматичною реплікацією та збереженням щонайменше трьох реплік у дата-центрах по всьому світу.

Amazon DynamoDB також має такі слабкі сторони:

1. DynamoDB може використовуватися виключно у хмарному середовищі AWS.
2. Хоча AWS надає безкоштовний пробний період, після закінчення терміну його дії для продовження роботи з DynamoDB необхідно платити за обсяг даних, що зберігається, операції читання і запису, передачу даних та резервне копіювання.
3. DynamoDB має значне обмеження на розмір документів, яке для кожного становить 400 КБ.
4. DynamoDB не має фреймворку для роботи з Kotlin.

2.3.3. *Результати аналізу*

Зважаючи на результати проведеного аналізу та описані переваги, було прийнято рішення використовувати базу даних MongoDB. На це рішення вплинули такі фактори, як можливість інтеграції з раніше обраною

мовою програмування Kotlin та ширші можливості, що надаються безкоштовною версією, зокрема більший обсяг даних, збережених у одному документі.

Для налаштування взаємодії бізнес-логіки із базою даних MongoDB було обрано набір інструментів KMongo [15], який розроблюється та підтримується спільнотою програмістів і є написаний на основі офіційної бібліотеки MongoDB для мови програмування Java.

2.4. Вибір хмарного сервісу та технологій для розгортання серверної частини у хмарному середовищі

З причин відсутності відповідного локального обладнання та задля забезпечення цілодобової доступності сервера було прийнято рішення розгортати серверну частину у хмарному середовищі. На сьогодні розгортання частини інфраструктури застосунків у хмарі стало поширеним рішенням, що спричинило появу різних постачальників послуг. Пропонується порівняти три найбільших постачальника послуг хмарних технологій: Amazon Web Services, Microsoft Azure, Google Cloud Platform.

2.4.1. *Amazon Web Services (AWS) [16]*

AWS – це одна з найперший та найбільш поширених хмарних платформ, що на основі платної підписки пропонує більш ніж 200 повнофункціональних послуг для користувачів по всьому світу.

Розглянемо сильні сторони AWS:

1. Гнучкість AWS дає користувачу можливість вільно змінювати інфраструктуру відповідно своїм поточним потребам.
2. Платформа дозволяє користувачам використовувати вже знайомі технічні рішення та спрощує їх налаштування за допомогою зручного інтерфейсу.

3. Принцип найменших привілеїв забезпечує захист ресурсів користувачів за умовчанням.
4. Принцип pay-as-you-go дозволяє користувачам платити лише за ті ресурси, що вони використовують, без будь-яких передплат чи довгострокових зобов’язань.
5. AWS надає безкоштовний пробний період тривалістю 12 місяців для ознайомлення з найбільш популярними послугами.
6. Популярність платформи призвела до формування широкої спілки користувачів та великої кількості документації, навчальних курсів, керівництв користувачів тощо.

2.4.2. *Microsoft Azure [17]*

Microsoft Azure – це хмарна платформа, розроблена корпорацією Microsoft, що покликана спростити процес створення онлайн застосунків. Як і AWS, Azure пропонує понад 200 послуг на основі платної підписки.

Наведемо особливості Azure, які вирізняють її від AWS:

1. Azure також пропонує 12 місяців пробного періоду для певних сервісів, але на додачу у перший місяць новим клієнтам надається кредит Azure на суму 200 доларів США, які можливо використати впродовж цього терміну на будь-які сервіси.
2. У AWS тарифікація за використані ресурси відбувається погодинно, а у Azure щохвилинно.
3. Інтеграція з іншими інструментами Microsoft, такі як Office 365, Active Directory, Microsoft Teams, OneDrive тощо.
4. Azure дозволяє власникам ліцензій Windows Server та SQL Server, які були підтверджені за допомогою Software Assurance, платити за деякі ресурси за спеціальними зниженими тарифами.
5. У Azure представлені набори інструментів та служб, що налаштовують взаємодію локальних та хмарних ресурсів.

2.4.3. *Google Cloud Platform [18]*

Google Cloud Platform (GCP) – це набір хмарних служб, розроблених компанією Google, що виконуються на тій же інфраструктурі, яку Google використовує для власних продуктів, призначених для кінцевих споживачів.

Наведемо особливості GCP, які не притаманні іншим попередньо розглянутим хмарним платформам:

1. На відміну від AWS та Azure, де знижені тарифи діють лише при передоплаті за рік чи три роки, GCP автоматично нараховує знижки при тривалому використанні певних ресурсів упродовж значно меншого терміну (кілька тижнів або місяців).
2. Аналогічно до Azure та продуктів Microsoft, GCP інтегрується з деякими продуктами Google, зокрема Google Workplace, а також надає актуальний інструментарій, що розроблює та використовує сама компанія Google, як-от reCAPTCHA.
3. GCP має розширений інструментарій для DevOps, технологій штучного інтелекту та машинного навчання.
4. GCP – це єдина платформа, що підтримує мультихмарність: за допомогою деяких інструментів GCP користувачі можуть використовувати сервіси та методики GCP у кількох різних середовищах одночасно, зокрема у AWS та Azure.
5. За додаткову плату користувачі можуть використовувати глобальну високонадійну та високопродуктивну мережу компанії Google для передавання даних із мінімальною затримкою.

2.4.4. *Результати аналізу хмарних сервісів*

За результатами проведеного аналізу було вирішено обрати для хостингу серверної частини у хмарному середовищі платформу AWS. Вибір дійсно є складним, адже усі три розглянуті кандидати надають схожі послуги. Факторами, що вплинули на остаточний вибір AWS, є популярність платформи (саме AWS є піонером хмарних обчислень та

займає перше місце серед постачальників аналогічних послуг), широка колекція документації та навчальних матеріалів, простота освоєння платформи новачками, а також найбільша кількість інструментів для контролю доступу до ресурсів AWS.

2.4.5. *Вибір технологій для розгортання серверної частини у хмарному середовищі*

Платформа AWS має щонайменше 200 різноманітних інструментів для розробки у хмарному середовищі, тому не пропонує єдиної «правильної» архітектури серверів. Для розгортання серверної частини клієнт-серверної архітектури найбільш поширений базовий підхід полягає у використанні двох віртуальних машин Elastic Compute Cloud (EC2) для окремого хостингу коду сервера та бази даних.

Для того, щоб позбавити користувача необхідності вручну налаштовувати багато аспектів машин EC2 (наприклад, масштабування чи оновлення ОС), AWS пропонує сервіси для швидкого та простого підняття архітектури. Провайдери інших технологій також часто пропонують свої послуги у вигляді хмарних рішень та використовують для цього ресурси AWS. Наприклад, сервіс MongoDB Atlas [13] дозволяє розгорнути базу даних MongoDB у хмарному середовищі AWS, Azure чи GCP. Усі ці сервіси так само запускають нові машини EC2, але більшість процесів керування переймають на себе, тим самим звільнюючи користувача від необхідності самостійного налаштування. Звичайно, за таку зручність користувачу доводиться платити: більшість таких сервісів коштуватимуть дорожче, ніж використання машин EC2 із самостійним налаштовуванням.

У рамках дипломного проєкту було вирішено власноруч налаштувати один екземпляр віртуальної машини EC2 у AWS для хостингу сервера, адже у рамках безкоштовного пробного періоду передбачена безоплатна цілодобова робота одного екземпляра машини EC2 певних типів. Для хостингу бази даних було прийнято рішення використати ту саму машину

EC2, так як у рамках пробного періоду безкоштовно та цілодобово може працювати лише один екземпляр віртуальної машини, а на початкових стадіях життєвого циклу мобільного застосунка обсягів обчислювальних ресурсів машини EC2, які доступні у рамках пробного періоду, має бути достатньо для одночасної роботи на ній і бази даних, і коду бізнес-логіки. Альтернативним варіантом для хостингу бази даних був згаданий раніше сервіс MongoDB Atlas, але його безкоштовний тариф пропонує значно менші обчислювальні потужності, ніж безкоштовна машина EC2.

2.5. Вибір мови програмування та технологій для розроблення клієнтської частини

2.5.1. *Вибір мови програмування*

Вибір мови програмування для розроблення клієнтської частини клієнт-серверної архітектури у першу чергу залежить від типу мобільного застосунку. Для розробки нативних мобільних застосунків під ОС Android найпопулярнішими мовами є Java та Kotlin. У рамках дипломного проєкту було прийнято рішення для розробки клієнтської частини також обрати мову Kotlin. На це рішення вплинуло кілька факторів:

1. *Розширені можливості.* Під час аналізу мов програмування для розробки серверної частини було встановлено, що Kotlin має низку переваг, що роблять її кращим вибором за Java.
2. *Офіційне визнання авторитетною компанією.* Під час конференції Google I/O 2019 компанія Google оголосила, що Kotlin стала пріоритетною у розробці застосунків під ОС Android: компанія заявила, що «Якщо ви починаєте новий проєкт, вам слід написати його мовою Kotlin» [19].
3. *Офіційна підтримка нових сумісних технологій.* На цій же конференції [19] Google заявила, що більшість нових модулів та

функцій, створених ними для розробки під ОС Android, будуть розроблені у першу чергу під Kotlin.

2.5.2. *Вибір технології розроблення UI*

Традиційно для розроблення UI нативних мобільних застосунків використовується колекція файлів формату XML. У цих файлах описується перелік компонент, що використовуються для представлення елементів інтерфейсу, зокрема елементи взаємодії (кнопки, перемикачі, поля вводу тощо), рядкові літерали чи шістнадцяткові коди кольорів. До деяких компонент можливо додати певні обробники події, описані у програмному коді (наприклад, функція, що викликається при натисненні на кнопку).

Альтернативним варіантом створення UI є використання набору інструментів та бібліотек Android Jetpack Compose [20]. Саме цьому підходу була надана перевага у даному дипломному проєкті. Прийняттю цього рішення сприяв ряд наступних переваг:

1. *Рекомендований авторитетною компанією.* На сьогодні Google рекомендує використовувати інструментарій Jetpack Compose для розробки UI мобільних застосунків під ОС Android.
2. *Не потрібно створювати UI у XML файлах.* Увесь інтерфейс створюється програмно за допомогою коду мовою Kotlin.
3. *Зменшення повторюваного коду.* Jetpack Compose самостійно керує такими складними операціями, як фонові процеси, навігація та керування життєвим циклом.
4. *Декларативний API.* Замість того, щоб явною послідовністю команд описувати, як досягти результату, розробник описує лише те, що він хоче отримати у результаті.
5. *Адаптивність.* За допомогою збереження стану змінних та прямого доступу до компонент у коді розробник може швидко та легко оновлювати інтерфейс у відповідь на дії користувача.

6. *Анімації.* Jetpack Compose має окремий набір інструментів для створення складних анімацій.

3. СТРУКТУРНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Загальний опис мобільного застосунка

Розроблений мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів дозволяє зареєстрованим користувачам-водіям зберігати інформацію про свої автомобілі разом із записами про проходження цими автомобілями техоглядів, а також дозволяє користувачам переглядати статистику власних витрат і отримувати сповіщення про заплановані техогляди. Окрім цього застосунок надає користувачам знеособлену аналітику поширених проблем певного типу автомобіля та перелік робіт певної СТО з оцінками виконання цих робіт, що поставлені іншими користувачами.

У застосунку також передбачено багаторівневий доступ користувачів до даних, що захищатиме дані користувачів від неавторизованого впливу. За замовчуванням користувачі не можуть виконувати жодних операцій з даними інших користувачів. Для того, щоб певний зареєстрований користувач отримав доступ до даних іншого користувача, власник даних має явно вказати для іншого користувача його роль, що визначає вичерпний перелік дозволених операцій.

Всього передбачено три ролі авторизованих користувачів: Власник, Редактор, Переглядач. Функціональність відрізняється для кожної з цих ролей та являє собою окремий сценарій використання (див. Додаток 1. Функціональність мобільного застосунка). Пропонується окремо розглянути кожен з ключових сценарії використання застосунка.

3.1.1. *Функціональність для неавторизованого користувача*

Неавторизований користувач жодним чином не взаємодіє з даними інших користувачів та має дуже обмежені можливості:

- Увійти у свій обліковий запис.

- Зареєструвати новий обліковий запис.
- Відновити пароль від свого облікового запису.

3.1.2. *Функціональність для Переглядача*

Авторизований користувач типу Переглядач має наступні можливості:

- Переглядати дані облікового запису.
- Переглядати перелік та детальну інформацію доступних автомобілів.
- Переглядати перелік та детальну інформацію записів про техогляд доступних автомобілів, а також сортувати та фільтрувати ці записи.
- Отримувати сповіщення про заплановані техогляди доступних автомобілів.
- Переглядати статистику витрат доступних автомобілів.
- Переглядати знеособлену аналітику поширених проблем за типами автомобілів.
- Переглядати перелік виконаних ремонтних робіт певної СТО із оцінками інших користувачів.
- Створювати новий автомобіль.

При створенні нового автомобіля користувачу автоматично встановлюється роль Власника, а решта користувачів не мають жодного доступу до даних новоствореного автомобіля.

3.1.3. *Функціональність для Редактора*

Авторизований користувач типу Редактор має ті ж самі можливості, що й користувач типу Переглядач, але також має перелік додаткових можливостей, недоступних для Переглядача, а саме: додавати, редагувати та видаляти записи про проходження техогляду для доступних автомобілів.

3.1.4. *Функціональність для Власника*

Авторизований користувач типу Власник має ті ж самі можливості, що й користувач типу Редактор, але також має перелік додаткових можливостей, недоступних ні для Редактора, ні для Переглядача, а саме:

- Редагувати та видаляти доступні автомобілі.
- Змінювати дату отримання сповіщення.
- Вимикати сповіщення для усіх отримувачів.
- Надавати, змінювати та забороняти доступ інших користувачів до доступних автомобілів шляхом вказування певної ролі (Редактор або Переглядач).
- Передавати права Власника іншому користувачу.

При налаштуванні доступу інших користувачів до своїх даних Власник має обмеження: для певного автомобіля може бути визначений лише один користувач типу Власник та лише один користувач типу Редактор. Обмежень на кількість користувачів типу Переглядач не передбачено.

3.2. Загальна структура мобільного застосунка

Програмне забезпечення реалізоване у вигляді мобільного застосунка із використанням програмних засобів, обраних у розділі 2, та архітектурного шаблону MVC. Програмну систему можна поділити на дві частини:

1. Серверна частина, яка представляє модель та контролер архітектури MVC, містить усю бізнес-логіку застосунка та відповідає за роботу з базою даних.
2. Клієнтська частина, яка представляє вигляд й дає користувачу можливість повноцінно взаємодіяти із застосунком.

Основні елементи кожної з частин та зв’язки між ними представлено на структурній схемі мобільного застосунка (див. Додаток 1. Структурна схема мобільного застосунка). Клієнтська частина представляє з себе

мобільний застосунок на мобільному пристрої користувача, а серверна – програмний код та базу даних, що знаходяться у хмарному просторі AWS. Взаємодія між цими частинами відбувається через інтерфейс RESTful API за допомогою мережі Інтернет та фреймворку Ktor.

Пропонується окремо розглянути структуру та функції серверної та клієнтської частин мобільного застосунка.

3.2.1. *Структура серверної частини мобільного застосунка*

Серверна частина мобільного застосунка виконує такі функції:

1. Отримує, оброблює та відповідає на асинхронні HTTP запити від клієнтської частини.
2. Реалізує частини моделі (Model) та контролера (Controller) архітектурного шаблону MVC.
3. Керує доступом користувачів до даних.
4. Взаємодіє із базою даних для виконання операцій CRUD.
5. Формує та надсилає сповіщення користувачам.
6. Підбирає дані для формування статистики витрат.
7. Підбирає перелік робіт певної СТО із оцінками їх виконання.
8. Формує знеособлену аналітику поширених проблем автомобілів.

Робота серверної частини починається з отримання запиту від клієнта за допомогою фреймворку Ktor. Модуль RESTful API, що виступає роутером серверної частини, на основі отриманого запиту передає керування відповідному модулю, відповідальному за обробку цього запиту (опис доступних модулів наведений нижче). Відповідний модуль оброблює запит, перевіряє токен авторизації та ввід користувача, який передається від клієнта із запитом за допомогою об’єктів DTO, взаємодіє з базою даних, формує та надсилає відповідь клієнту.

Перевірка дійсності токену авторизації виконується за допомогою стороннього сервісу Firebase Auth [21], що дозволяє полегшити процес авторизації користувачів. Взаємодія з базою даних відбувається за

допомогою об’єктів DAO, які приховують від користувача особливості бази даних і представлення у ній даних, а також реалізацію методів взаємодії з цими даними. У свою чергу об’єкти DAO для взаємодії з базою даних використовують набір інструментів KMongo.

Серверна частина мобільного застосунку містить наступні модулі:

1. *Модуль роботи з автомобілями* – призначений для виконання операцій CRUD з даними самих автомобілів.
2. *Модуль роботи із записами про техогляд* – створений для виконання операцій CRUD з даними записів про техогляд.
3. *Модуль налаштування доступу користувачів до даних* – виконує операції CRUD із даними про рівні доступу користувачів до даних певних автомобілів, а також додає дані нових користувачів після успішної реєстрації на клієнті.
4. *Модуль статистики витрат* – підбирає та структурує дані про техогляди для формування статистики витрат автомобіля.
5. *Модуль сповіщення користувачів* – здійснює надсилання користувачам сповіщень, а також дає можливість змінювати дату сповіщення та значення прапорця, що вказує, чи потрібно продовжувати сповіщати певного користувача.
6. *Модуль аналітики поширених проблем автомобілів* – надає перелік робіт та замінених деталей, що найчастіше зустрічаються у записах про техогляд для певного типу автомобіля.
7. *Модуль переліку робіт СТО з оцінками виконання* – формує перелік робіт, що виконувалися на певній СТО, та обраховує їх середні оцінки на основні оцінок інших користувачів.

3.2.2. *Структура клієнтської частини мобільного застосунка*

Клієнтська частина мобільного застосунка виконує такі функції:

1. Реалізує інтерфейс для взаємодії з користувачем.
2. Реалізує частину вигляду (View) архітектурного шаблону MVC.

3. Отримує ввід користувача та виконує його первинну перевірку.
4. Формує, відправляє та отримує відповідь на асинхронні HTTP запити до серверної частини.
5. Реєструє та авторизує користувачів застосунка, а також відповідає за відновлення паролю.

Клієнтська частина має власну архітектуру, яка реалізує шаблон MVVM. Вибору саме цього архітектурного шаблону посприяв обраний у розділі 2 інструментарій розробки інтерфейсу користувача Jetpack Compose, що дозволяє двобічно пов’язати дані у моделі та вигляді клієнтської частини за допомогою моделі вигляду (ViewModel). Таким чином модель вигляду може проводити первинну перевірку вводу користувача ще до того, як передати отримані дані моделі, що надсилатиме запит до серверної частини.

Клієнтська частина мобільного застосунку містить наступні модулі:

1. *Модуль вигляду* – представляє інтерфейс користувача, що відображає інформацію, отриману від моделі вигляду, та отримує ввід користувача, який передає моделі вигляду для обробки.
2. *Модуль моделі вигляду* – являється проміжною ланкою між видом та моделлю, що оброблює ввід користувача, формує запити до моделі, отримує відповіді на запити та оновлює вигляд.
3. *Модуль моделі* – виступає бізнес-логікою клієнтської частини, що взаємодіє із сервером, а також виконує процедури авторизації та реєстрації користувачів.

Після отримання запиту та необхідних вхідних даних від моделі вигляду модуль моделі формує запит до сервера за допомогою фреймворку Ktor. За наявності вводу користувача до тіла запиту додаються об’єкти DTO. Отримавши відповідь від сервера, модель передає необхідну частину відповіді назад моделі вигляду, що у свою чергу оновлює вигляд та відображає результат користувачу.

Модуль моделі клієнтської частини також відповідає за реєстрацію та авторизацію користувачів. Для реалізації цих функцій використовується

сторонній сервіс Firebase Auth, що безпечним чином зберігає дані авторизації користувачів (електронна пошта та пароль) та полегшує процес реєстрації, авторизації та зміни пароля. При успішній авторизації Firebase Auth повертає токен авторизації, який зберігається на клієнтській частині та передається серверній частині разом із запитом для виконання операцій, що потребують авторизованого доступу.

3.3. Структура бази даних

На відміну від реляційних баз даних, нереляційні бази не вимагають попереднього задання чіткої схеми даних: у одній колекції можуть зберігатися документи, які містять різну кількість полів та зберігають дані різних типів. Але така неструктурованість даних водночас є великою проблемою для розробників програмного забезпечення: розробник буде вимушений постійно перевіряти, чи існує необхідне поле і якого типу дані воно містить, через що йому доведеться оброблювати безліч комбінацій обов'язкових та необов'язкових полів.

Але MongoDB, як і реляційні бази даних, підтримує створення схеми даних. За допомогою схеми можливо визначити мінімальний перелік обов'язкових полів, що мають бути наявні у документі, та типи даних, що зберігаються у цих полях. Для розробки мобільного застосунка було спроєктовано схему бази даних. Колекції документів, їх поля, типи даних полів та зв'язки між документами представлено у вигляді ER-діаграми (див. Додаток 1. Структура бази даних).

Пропонується детальніше розглянути кожну сутність спроєктованої ER-діаграми з описом їх призначення, полів і зв'язків з іншими сутностями.

Колекція *«Користувач» (табл. user)* представляє профіль зареєстрованого користувача та містить наступні обов'язкові поля:

1. Унікальний ідентифікатор користувача (id) – первинний ключ.
2. Ім'я користувача (name).

3. Електронна пошта (email) – унікальне значення, що використовується для авторизації користувача.

Колекція *«Автомобіль» (табл. car)* зберігає дані автомобілів користувачів та містить наступні обов’язкові поля:

1. Унікальний ідентифікатор автомобіля (id) – первинний ключ.
2. Назва автомобіля (name).
3. Виробник автомобіля (manufacturer).
4. Модель автомобіля (model).
5. Рік випуску автомобіля (year).
6. Номерний знак автомобіля (plateNumber).
7. Пробіг автомобіля (mileage) – при створенні задається користувачем, після створення система автоматично оновлює значення даними з поля currentMileage із запису про техогляд, що має дату проведення найближчу до поточної.
8. Тип пального автомобіля (fuelType).

Для того, щоб забезпечити однозначність даних виробників та моделей автомобілів, було створено колекцію-словник *«Словник типів автомобілів» (табл. car_dictionary)*. Це необхідно для коректної роботи модуля аналітики поширених проблем автомобілів, що групує дані автомобілів за їх типами: якби користувач самостійно вводив виробника та модель автомобіля, то було б неможливо передбачити усі можливі варіації, що позначали б один і той самий тип автомобіля, особливо якщо запис містить помилки (наприклад, Toyota Prius, prius toyota, Тойота Пріус, Tooyta Prius). При створенні автомобіля користувачі мають обрати модель та тип зі словника.

Колекція «Тип автомобілів» заповнюється адміністратором бази даних та містить наступні поля:

1. Назва виробника автомобіля (manufacturer).
2. Назва моделі автомобіля (model).

За тієї ж причини також створено колекцію-словник *«Словник типів пального» (табл. car_fuel_dictionary)*, що містить одне поле: назву типу пального (type). Ця колекція також заповнюється адміністратором бази даних та містить дані, однакові для усіх користувачів застосунка.

Між значеннями кожного з полів двох наведених колекцій-словників та відповідних полів документа у колекції «Автомобіль» встановлений зв’язок «один обов’язковий до одного обов’язкового».

Для того, щоб пов’язати користувачів із автомобілями, до яких вони мають доступ, та вказати ролі користувачів, була створена колекція-зв’язка *«Тип доступу до даних» (car_access_control)*. Ця колекція забезпечує зв’язок «багато до багатьох» між колекціями автомобілів та користувачів.

Колекція «Тип доступу до даних» містить наступні обов’язкові поля:

1. Унікальний ідентифікатор документа (id) – первинний ключ.
2. Унікальний ідентифікатор користувача (userId) – зовнішній ключ на поле id документів з колекції «Користувач».
3. Роль користувача (userRole) – одне з можливих значень: Власник, Редактор чи Переглядач.
4. Унікальний ідентифікатор автомобіля (carId) – зовнішній ключ на поле id документів з колекції «Автомобіль».

Колекція *«Запис про техогляд» (табл. checkup)* зберігає дані записів автомобілів про проходження техогляду. Ця колекція містить як обов’язкові, так і додаткові поля. Обов’язкові поля колекції наступні:

1. Унікальний ідентифікатор запису про техогляд (id) – первинний ключ.
2. Унікальний ідентифікатор автомобіля (carId) – зовнішній ключ на поле id документів з колекції «Автомобіль»; забезпечує зв’язок «один обов’язковий користувач до жодного або багатьох автомобілів».
3. Дата проведення техогляду (checkupDate).
4. Пробіг автомобіля на момент техогляду (currentMileage).

5. Список виконаних ремонтних робіт (services) – користувач має вказати щонайменше одну виконану роботу.
6. Загальна вартість техогляду (totalCost) – вираховується автоматично як сума вартості окремих компонент (виконаних ремонтних робіт та, за наявності, замінених деталей).
7. Дата нагадування про наступний техогляд (nextCheckupNotificationDate).

Алгоритм встановлення значення дати нагадування про наступний техогляд наведений у пункті 4.1.2.

Колекція «Запис про техогляд» також має наступні необов'язкові поля:

1. Список замінених деталей (parts).
2. Назва СТО, де виконувався техогляд (serviceStationName).

Для того, щоб забезпечити однозначність назв СТО, було створено колекцію-словник *«Словник СТО» (табл. service_station_dictionary)*, яка також заповнюється адміністратором бази даних та містить одне поле: назву СТО (name). Схожим чином, як і з колекціями-словниками для колекції «Автомобіль», введення цієї колекції необхідно для коректної роботи модуля формування переліку робіт СТО із оцінками користувачів. Між значенням поля serviceStationName колекції «Запис про техогляд» та полем даної колекції-словника встановлений зв'язок «один обов'язковий до одного обов'язкового». При створенні запису про техогляд користувачі обирають назву СТО зі словника.

Колекція «Запис про техогляд» також містить дві вкладених колекції: вкладену колекцію *«Виконані роботи» (табл. car_service)* та вкладену колекцію *«Замінені деталі» (табл. car_parts)*. Ці колекції представляють об'єкти, що зберігаються відповідно у полях services та parts колекції «Запис про техогляд» та слугують для збереження детальної інформації про проведені ремонтні роботи та замінені деталі.

Ці дві вкладені колекції містять однаковий перелік обов’язкових полів:

1. Назва роботи або деталі (name).
2. Вартість виконання роботи або ціна деталі (cost).
3. Тривалість виконання роботи або кількість використаних деталей (amount).
4. Загальна вартість роботи або деталей (totalCost) – вираховується автоматично як добуток полів cost та amount.

Для того, щоб забезпечити однозначність назв робіт та деталей, було створено дві колекції-словники *«Словник ремонтних робіт» (табл. car_service_dictionary)* та *«Словник деталей» (car_part_dictionary)*, що обидві заповнюються адміністратором бази даних та містять по одному полю: назва ремонтної роботи або деталі (name). Так само, як і у випадку з іншими колекціями-словниками, створення цих колекцій є необхідним для коректної роботи аналітичних модулів. Між значенням поля name вкладених колекцій «Виконані роботи» та «Замінені деталі» та поля name відповідної колекції-словника встановлений зв’язок «один обов’язковий до одного обов’язкового». При додаванні ремонтних робіт та замінених деталей до запису про техогляд користувачі мають обрати назву виконаної роботи або заміненої деталі з відповідного словника.

Вкладена колекція «Виконані роботи» також містить одне необов’язкове поле: відгук користувача (review). Це поле використовується для формування оцінок виконання робіт СТО.

Колекція *«Нагадування» (табл. notification)* зберігає дані нагадувань про майбутній техогляд. Ця колекція автоматично заповнюється системою та містить наступні поля:

1. Унікальний ідентифікатор нагадування (id) – первинний ключ.
2. Дата початку нагадування (firstNotificationDate) – береться з поля nextCheckupNotificationDate відповідного запису про техогляд.

3. Булевий прапорець, чи потрібно продовжувати сповіщати (isResolved) – при значенні false користувачі, що мають отримувати це сповіщення, більше не будуть його отримувати.
4. Унікальний ідентифікатор автомобіля (carId) – зовнішній ключ на поле id документів з колекції «Автомобіль».
5. Унікальний ідентифікатор запису про техогляд (checkupId) – зовнішній ключ на поле id документів з колекції «Запис про техогляд».
6. Унікальний ідентифікатор користувача, що для вказаного автомобіля має роль Власник (ownerId) – зовнішній ключ на поле id документів з колекції «Користувач».
7. Список отримувачів сповіщення (receivers) – список зовнішніх ключів на поле id документів з колекції «Користувач»; містить ідентифікатори лише тих користувачів, що мають будь-який рівень доступу до вказаного автомобіля.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації програмного забезпечення

4.1.1. *Особливості розмежування доступу до даних користувачів*

Користувач може одночасно мати доступ до кількох автомобілів із різними правами доступу, тож перелік сторінок, що є доступними для користувача, залежить від прав доступу до даних конкретного автомобіля. Основні сторінки застосунку та навігація між ними в залежності від прав доступу користувача представлено у вигляді діаграми (див. Додаток 1. Структура сторінок інтерфейсу).

При відкритті застосунка неавторизований користувач потрапляє на сторінку входу (рис. 2). Для авторизації користувач повинен ввести електронну пошту та пароль. З цієї сторінки неавторизований користувач може перейти на сторінку реєстрації нового облікового запису та сторінку відновлення паролю від існуючого облікового запису.

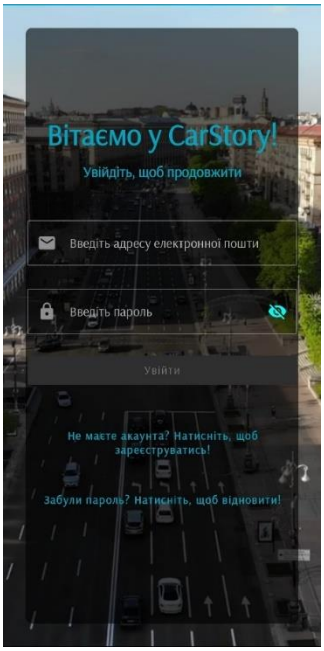


Рис. 2. Інтерфейс сторінки входу

Якщо користувач попередньо авторизувався у системі та закрив застосунок, але не вийшов з облікового запису, то такий користувач при

повторному відкритті застосунка переходить на іншу сторінку входу, на якій він має вказати лише пароль від свого облікового запису (рис. 3). Цей крок зроблений з метою захисту даних користувачів від неавторизованого доступу сторонніх осіб, які могли отримати безпосередній доступ до мобільного пристрою користувача зі збереженою сесією. З цієї сторінки користувач може перейти на сторінку відновлення паролю, але тоді система автоматично завершує поточну сесію користувача – і для входу у застосунок користувач має увійти вже як неавторизований користувач.

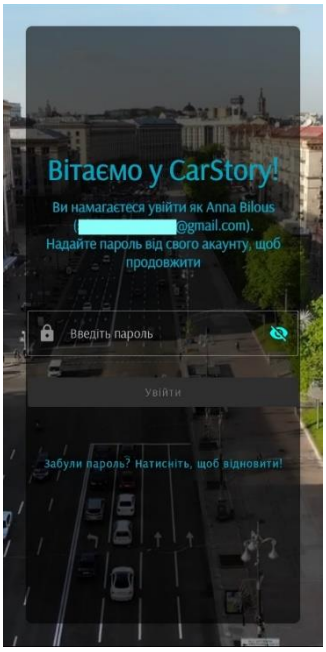


Рис. 3. Інтерфейс сторінки входу для попередньо авторизованого користувача

Керування сесією користувача відбувається за допомогою стороннього сервісу Firebase Auth, який здійснює процедури реєстрації, авторизації та відновлення паролю. Після успішної авторизації Firebase Auth зберігає сесію користувача локально на мобільному пристрої. Надалі за допомогою цієї сесії клієнтська частина отримує від сервісу Firebase Auth дані токену авторизації (лістинг 1), який є необхідним для виконання операцій, що потребують авторизованого доступу. Цей токен передається

на серверну частину у заголовку запиту, після чого серверна частина виконує перевірку дійсності токена і, якщо токен дійсний, оброблює запит.

Лістинг 1. Приклад токена авторизації Firebase Auth

```
{
   “token”: “eyJhb…VkC9Q”,
   “signInProvider”: “password”,
   “authTimestamp”: “1684664745”,
   “issuedAtTimestamp”: “1684664745”,
   “expirationTimestamp”: “1684668345”
}
```

Примітка: сам токен авторизації (поле token) наведений у скороченому виді через велику кількість символів у його складі.

Розмежування доступу авторизованих користувачів до даних певного автомобіля виконується за допомогою колекції-зв’язки *«Тип доступу до даних»*. Перед виконанням запиту серверна частина перевіряє, чи має користувач право на виконання вказаної у запиті дії. Для цього сервер шукає у базі даних запис про те, який рівень доступу певний користувач має до даних певного автомобіля. Відсутність у колекції такого запису свідчить про те, що користувач взагалі не має права робити будь-які дії з даними певного автомобіля. У разі відсутності права доступу або недостатнього рівня доступу для виконання вказаної операції сервер надсилає у відповідь код стану 403 Forbidden (Заборонено).

На рис. 4 розглянуто процес обробки запиту для роботи з даними певного автомобіля. Таким запитом може бути, наприклад, запит на редагування даних автомобіля, запит на створення запису про техогляд або запит на перегляд статистики витрат.

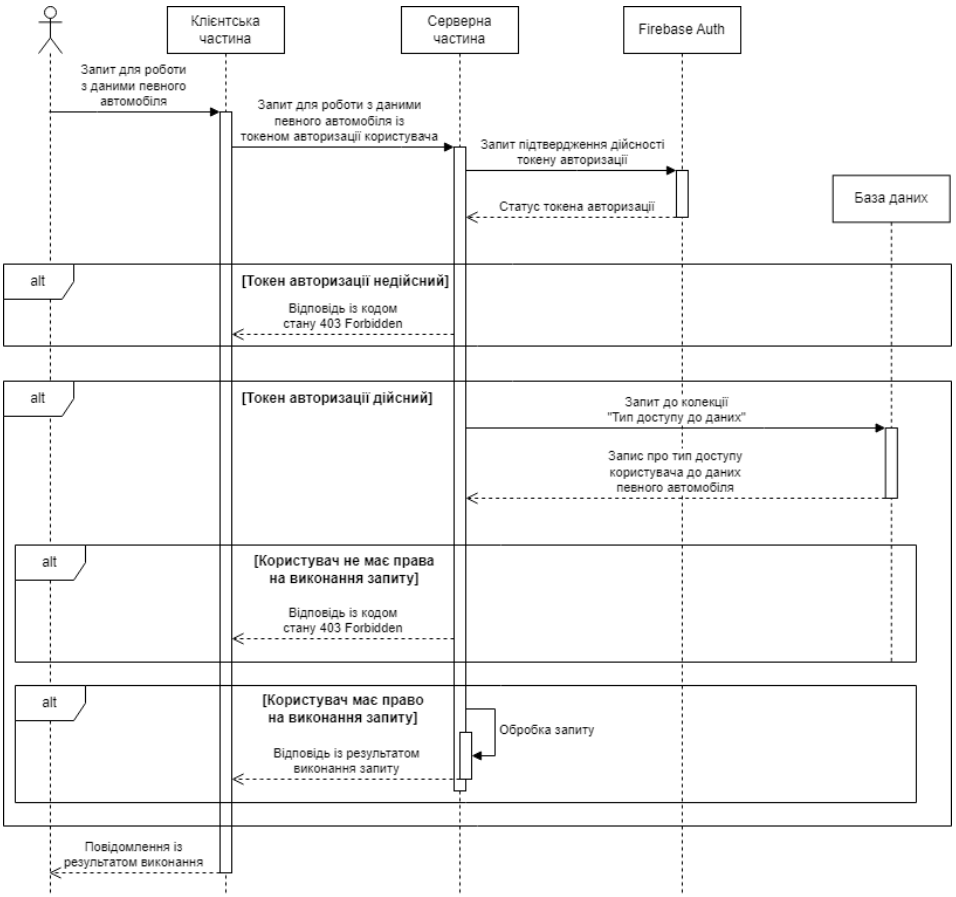


Рис. 4. Процес обробки запиту для роботи з даними певного автомобіля. Діаграма послідовності

Коли користувачі типу Власник редагують права доступу до даних інших користувачів, то серверна частина оновлює записи у колекції *«Тип доступу до даних»* вказаними новими записами. Також серверна частина автоматично оновлює у колекції *«Нагадування»* списки отримувачів для усіх майбутніх сповіщень певного автомобіля. Аналогічним чином відбувається оновлення вказаних колекцій і при передачі прав Власника іншому користувачеві.

4.1.2. *Особливості реалізації системи сповіщення користувачів*

При створенні користувачем нового запису про техогляд для цього запису автоматично створюється нагадування про майбутній техогляд. До

списку отримувачів нагадування записуються усі користувачі, що на момент створення мають будь-який рівень доступу до даних автомобіля.

Дата нагадування задається при створенні нового запису про техогляд. Але користувач може не задавати дату нагадування про наступний техогляд власноруч. У такому випадку система автоматично порахує «оптимальну» дату на основі останніх записів за наступною формулою (1):

$$t_{сповіщення} = t_{поточне} + \frac{\sum_{i=1}^{n} d_i}{n}, \quad (1)$$

де $t_{сповіщення}$ – дата, коли необхідно сповістити користувача, $t_{поточне}$ – дата створення запису, d_i – кількість днів, що пройшла між двома суміжними записами (береться до 5 останніх записів), n – кількість часових проміжків ($n_{max} = 4$).

Наприклад, користувач має три записи про проходження автомобілем техогляду, а при створенні четвертого запису не вказував дату нагадування самостійно. У результаті дата сповіщення буде вирахувана за формулою (2):

$$T_{сповіщення} = T + \frac{d_1 + d_2}{2}, \quad (2)$$

де T – поточна дата, d_1 – кількість днів, що пройшла між останнім та передостаннім техоглядами, d_2 – кількість днів, що пройшла між передостаннім та третім за давністю техоглядами.

Особливий випадок: користувач створює свій перший або другий запис про техогляд та не встановлює дату самостійно. У такому випадку система встановить датою сповіщення день за півроку від поточної дати.

Безпосереднє сповіщення користувачів відбувається один раз на добу за допомогою сервісу, що запущений на серверній частині у фоновому режимі. Алгоритм роботи цього сервісу зображено на рис. 5.

На першому етапі сервіс відбирає з бази даних такі записи про сповіщення, що одночасно задовольняють наступні критерії:

- Булевий прапорець isResolved має значення true (тобто користувачі мають отримати сповіщення про вказаний запис про проходження техогляду).

- Дата початку нагадування firstNotificationDate менша або дорівнює поточній даті.

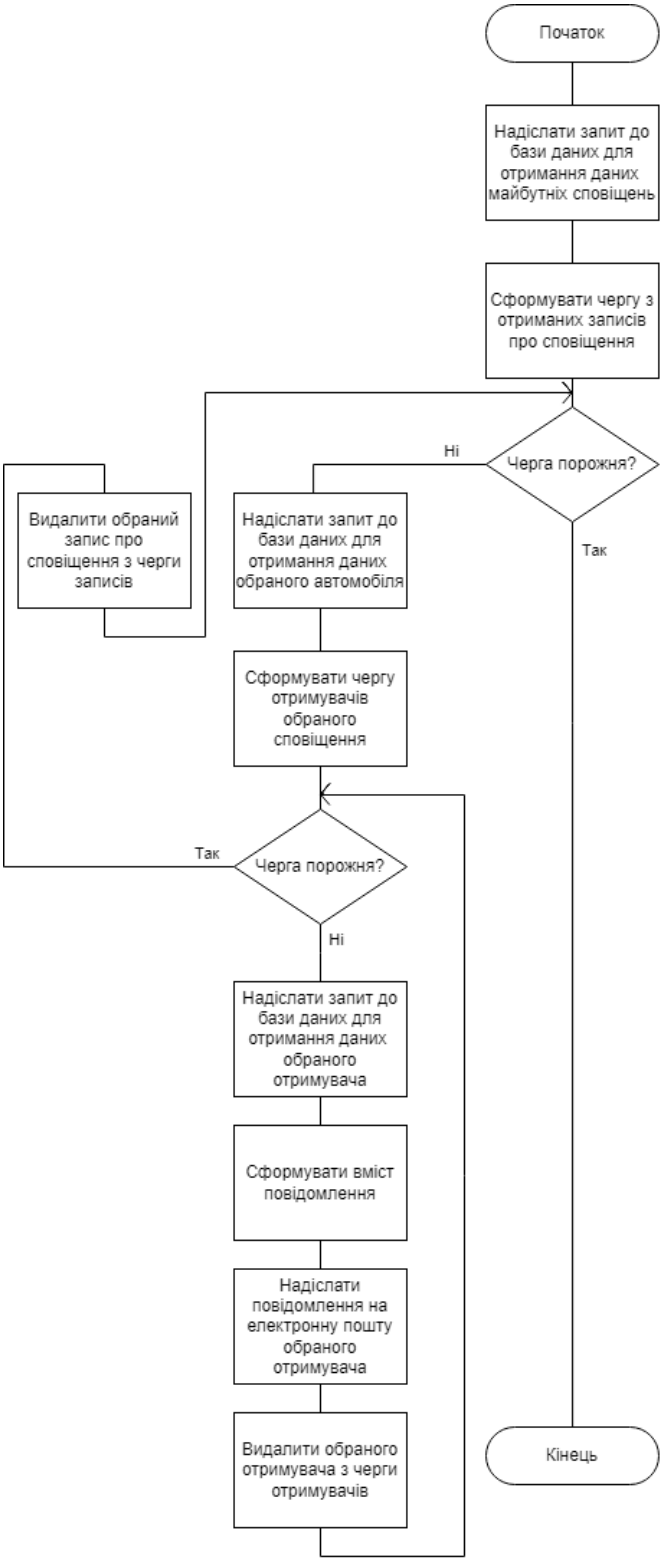


Рис. 5. Алгоритм сповіщення користувачів. Діаграма діяльності

Далі з отриманих записів формується черга. Для кожного запису у черзі з бази даних обираються дані обраного автомобіля, які використовуються при формуванні персоналізованого вмісту повідомлення, та формується власна черга з отримувачів сповіщення. У свою чергу для кожного отримувача у черзі з бази даних обираються дані облікового запису отримувача, формується повідомлення та надсилається на електронну пошту отримувача. Після надсилання листа отримувач видаляється з черги отримувачів певного сповіщення, а по закінченню черги отримувачів обраний запис про сповіщення видаляється з черги записів. Описаний цикл повторюється допоки черга сповіщень не закінчиться.

Вимкнути сповіщення може лише користувач типу Власник. Для цього він має у застосунку перейти на сторінку налаштування сповіщень (рис. 6), де представлений перелік сповіщень про майбутні техогляди. Кнопки, розташовані праворуч від кожного запису, дозволяють користувачу змінити дату першого сповіщення, а також статус сповіщення: позначити як прочитане (вимкнути) чи позначити непрочитаним (ввімкнути).

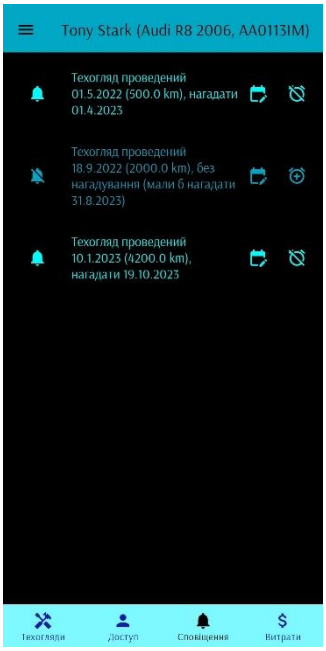


Рис. 6. Сторінка налаштування сповіщень

4.1.3. *Особливості формування аналітики поширених проблем автомобілів*

Для перегляду аналітики поширених проблем автомобілів користувачі мають скористатися виринаючим боковим меню та перейти на сторінку представлення аналітики поширених проблем автомобілів (рис. 7).

Рис. 7. Сторінка вибору автомобіля для представлення аналітики поширених проблем автомобілів

Задля спрощення представлення та обробки даних цей модуль було розбито на дві сторінки.

На першій сторінці користувачі обирають із наданого переліка тип автомобіля, для якого вони бажають переглянути аналітику, після чого на сервер надсилається запит на формування аналітики із обраним типом автомобіля. Алгоритм формування цієї аналітики на прикладі даних про виконані роботи представлений на рис. 8. та є аналогічним для формування аналітики для даних про замінені деталі.

У результаті клієнтська частина отримує перелік знеособлених даних, сформований сервером, у вигляді списку з об’єктів DTO (лістинг 2). Ці

об’єкти містять лише назву виконаної роботи чи заміненої деталі та частоту їх виконання чи заміни. Користувач автоматично переходить на другу сторінку, де йому відображаються отримані від сервера дані.

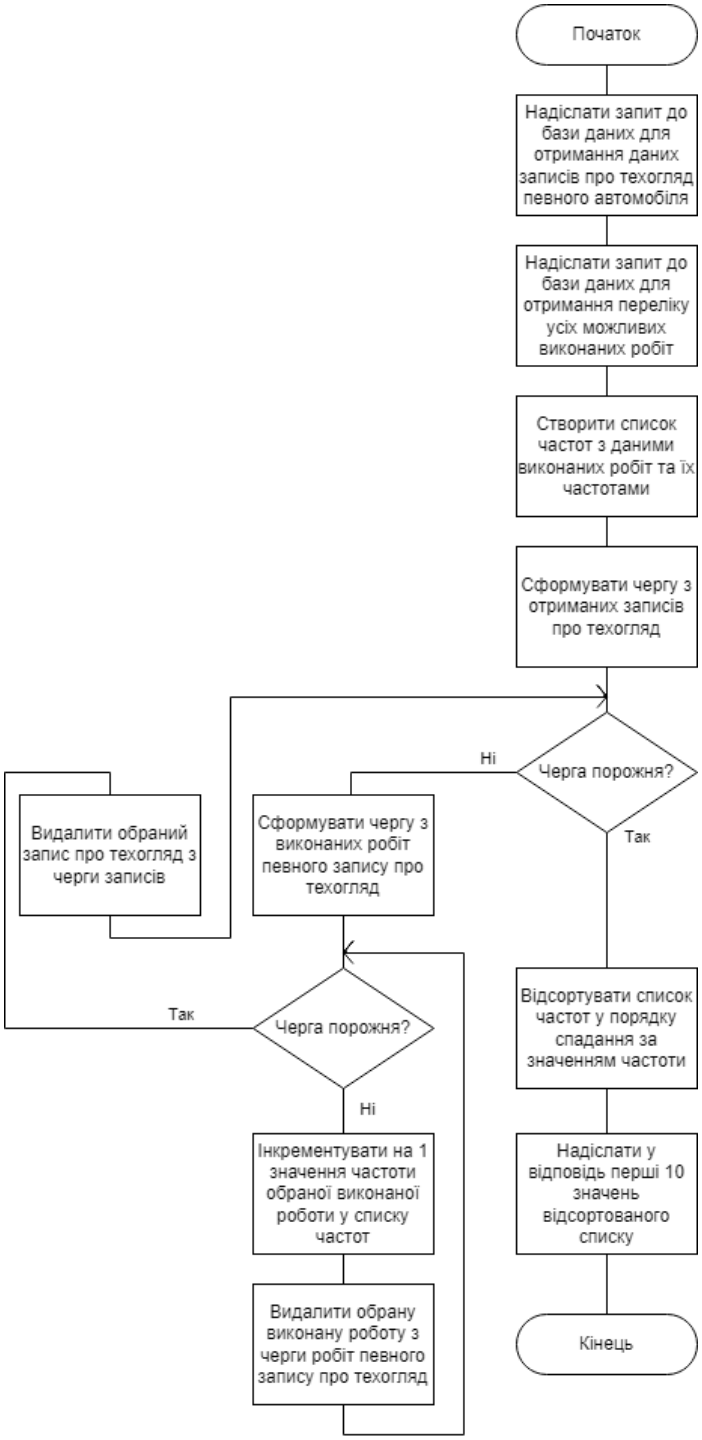


Рис. 8. Алгоритм відбору 10 найчастіше виконаних робіт певного автомобіля. Діаграма діяльності

Лістинг 2. Структура об’єкта DTO для аналітики поширених проблем автомобілів

```
/**
 * Common vehicle problems data transfer object
 *
 * @property problemName Service or part name
 * @property problemFrequency Frequency among other users (sum of all occurrences)
 */
@Serializable
data class CommonVehicleProblemsDTO(
    val problemName: String,
    val problemFrequency: Int
)
```

4.1.4. *Особливості формування переліку робіт СТО із оцінками виконання*

Для перегляду переліку робіт СТО із оцінками виконання іншими користувачами користувачі так само мають скористатися виринаючим боковим меню та перейти на сторінку представлення переліку робіт СТО із оцінками виконання іншими користувачами (рис. 9).

Рис. 9. Сторінка вибору СТО для представлення переліку робіт СТО із оцінками виконання іншими користувачами

Аналогічним чином, як і модуль аналітики поширених проблем автомобіля, цей модуль було розбито на дві сторінки.

На першій сторінці користувачі обирають із наданого переліка назву СТО, що їх цікавить, після чого на сервер надсилається запит на формування аналітики із обраною назвою СТО. Алгоритм формування цієї аналітики представлений на рис. 10.

У результаті клієнтська частина отримує перелік знеособлених даних, сформований сервером, у вигляді списку з об’єктів DTO (лістинг 3). Користувач автоматично переходить на другу сторінку, де йому відображаються отримані від сервера дані. Отримані об’єкти DTO містять лише назву виконаної роботи та середню оцінку їх виконання (за наявності). Якщо певна робота жодного разу не зустрічається у жодному із записів про техогляд, де вказана певна СТО, то замість оцінки вказується значення null, що свідчить про відсутність даних для формування оцінки.

Лістинг 3. Структура об’єкта DTO для переліку виконаних робіт певної СТО із оцінками виконання іншими користувачами

```
/**
 * Service station reviews data transfer object
 *
 * @property serviceName Service name (from dictionary)
* @property serviceAverageReview Review for given service station
 * (average of all reviews, null indicates no reviews at all)
 */
@Serializable
data class ServiceStationReviewsDTO(
    val serviceName: String,
    val serviceAverageReview: Double?
)
```

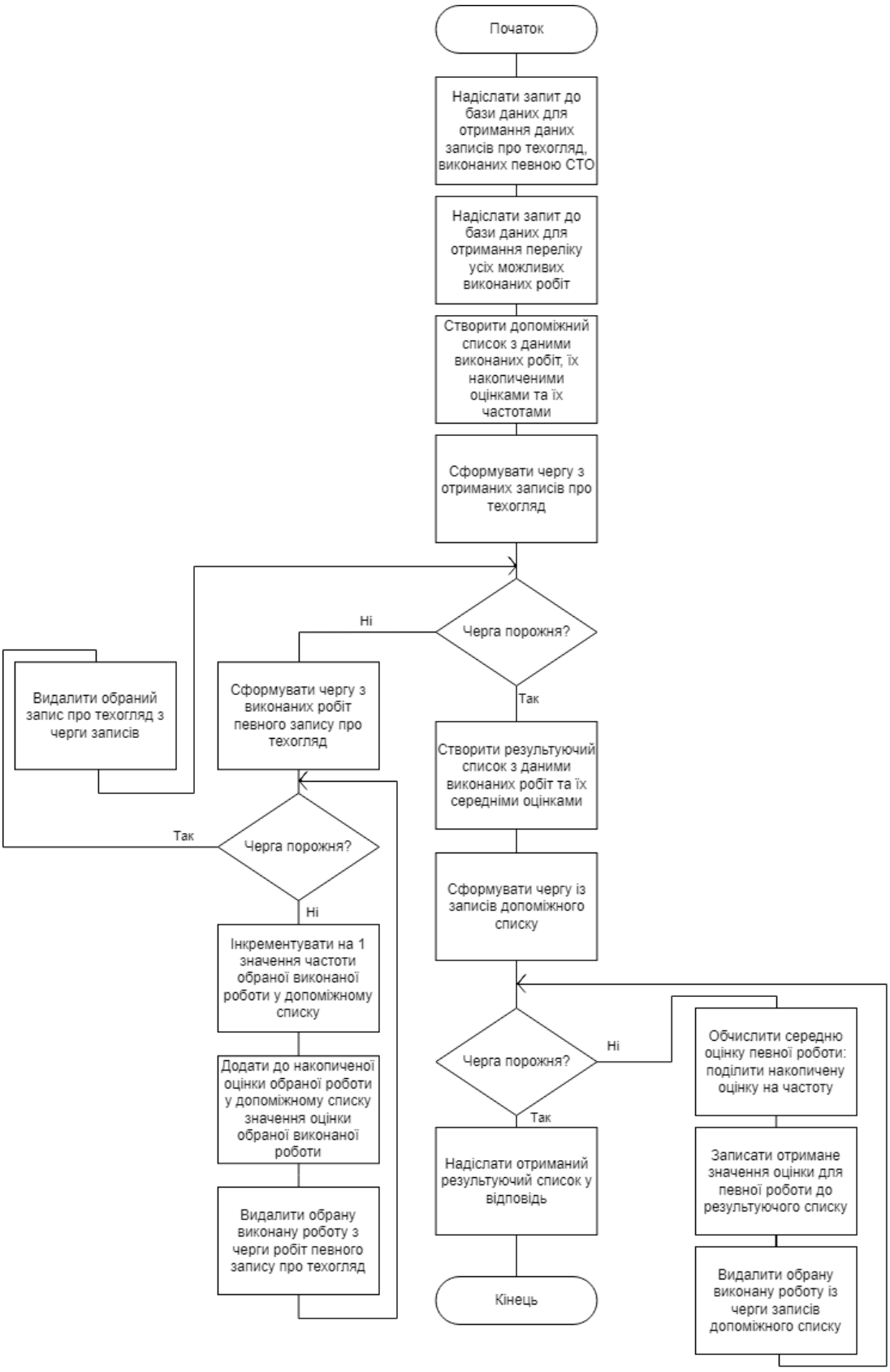


Рис. 10. Алгоритм формування переліку виконаних робіт певної СТО із оцінками виконання іншими користувачами. Діаграма діяльності

4.2. Тестування мобільного застосунка

Застосунок налаштований таким чином, що за відсутності з’єднання із сервером або із мережею Інтернет користувач отримує повідомлення про неможливість подальшої роботи, яке блокує використання застосунка, тому тестування клієнтської та серверної частини відбувалося одночасно. У якості методики тестування було обрано ручне тестування для моделювання дій користувача застосунка. Усі операції виконувалися за допомогою інтерфейсу. Таким чином було протестовано основні функції застосунка: реєстрація та авторизація користувача, операції CRUD з основними сутностями (автомобілі та записи про техогляд), налаштування доступу користувачів до даних, перегляд аналітики.

Більш детально розглянуто тестування чотирьох основних функцій мобільного застосунка: створення запису про техогляд, сповіщення про майбутній техогляд, надання доступу до даних автомобіля іншому користувачу, перегляд аналітики поширених проблем автомобіля. Тестування цих функцій виконувалося відповідно до техніки сірого ящику (gray box testing): перевірялися не тільки очікуваний результат у інтерфейсі застосунка, а і коректність збереження записів у базі даних.

4.2.1. *Тестування створення запису про проходження техогляду*

Передумови:

1. Користувач авторизований у застосунку.
2. Користувач має доступ до обраного автомобіля із правами типу Власник або Редактор.

Опис тестового випадку 4.2.1:

1. Користувач на сторінці додавання нового запису про проходження техогляду вводить необхідні дані, що успішно проходять перевірку на коректність, та натискає кнопку створення нового запису.

Результат тестового випадку 4.2.1:

1. Клієнт надсилає запит серверу, після чого:
   1.1. Сервер отримує дані від клієнта та перевіряє за токеном авторизації користувача, чи має користувач право на виконання даної операції.
   1.2. Дозвіл отримано, сервер додатково перевіряє ввід користувача на коректність.
   1.3. Дані коректні, до бази даних додається новий запис про проходження техогляду, автоматично додається запис нагадування про майбутній техогляд.
2. Сервер надсилає ідентифікатор новоствореної сутності, за допомогою якого клієнт перенаправляє користувача на сторінку детальної інформації нового запису.

4.2.2. *Тестування сповіщення про майбутній техогляд*

Передумови:

1. Користувач створив запис про проходження техогляду.
2. У базі даних у записі сповіщення дата співпадає або є раніше за поточну дату, а булевий прапорець, що вказує, чи потрібно ще сповіщати користувача, має значення true.

Опис тестового випадку 4.2.2:

1. Користувач отримує повідомлення на пошту із нагадуванням про майбутній техогляд.

Результат тестового випадку 4.2.2:

1. На сервері запускається запланована робота, яка виконує наступні дії:
   1.1. Отримує з бази даних усі сповіщення, для яких дата співпадає або є раніше за поточну дату, а булевий прапорець, що вказує, чи потрібно ще сповіщати користувача, має значення true.

1.2. Надсилає лист на електронні пошти тих користувачів, які вказані у списку отримувачів сповіщення.

4.2.3. *Тестування надання доступу до даних автомобіля іншому користувачу*

Передумови:

1. Користувач авторизований у застосунку.
2. Користувач має доступ до обраного автомобіля із правами типу Власник.
3. Користувачі, яким надається доступ, мають облікові записи.

Опис тестового випадку 4.2.3:

1. Користувач на сторінці редагування контролю доступу інших користувачів до даних обраного автомобіля вводить електронну пошту та тип доступу до даних нового користувача, після чого натискає кнопку збереження змін.

Результат тестового випадку 4.2.3:

1. Клієнт надсилає запит серверу, після чого:
   1.1. Сервер отримує дані від клієнта та перевіряє за токеном авторизації користувача, чи має користувач право на виконання даної операції.
   1.2. Дозвіл отримано, сервер перевіряє, чи існує обліковий запис користувача із вказаною поштою.
   1.3. Обліковий запис існує, до бази даних додається новий запис-зв’язка про тип доступу до даних обраного автомобіля, автоматично оновлюються записи сповіщень про майбутні техогляди, які ще не отримані (новий користувач додається до списку отримувачів).
2. Сервер надсилає статус операції, клієнт повідомляє користувача про результат виконання та перенаправляє на сторінку контролю доступу інших користувачів до даних обраного автомобілю.

4.2.4. *Тестування перегляду аналітики поширених проблем автомобіля*

Передумови:

1. Користувач авторизований у застосунку.

Опис тестового випадку 4.2.4:

1. Користувач на сторінці представлення аналітики поширених проблем автомобілів знаходить тип автомобіля, що його цікавить, та натискає на нього.

Результат тестового випадку 4.2.4:

1. Клієнт надсилає запит серверу, після чого:
    1.1. Сервер отримує дані типу автомобіля від клієнта та обирає з бази записи про техогляди для обраного типу автомобіля.
    1.2. Сервер формує аналітику: для кожної виконаної роботи та заміненої деталі зі словників підраховується кількість техоглядів, де ця робота чи деталь зустрічається, з отриманого переліку обираються по 10 записів з найбільшим значенням частоти.
2. Сервер надсилає отримані назви робіт та деталей та їх відповідну частоту клієнту, клієнт перенаправляє користувача на сторінку із представленням результатів.

4.3. Рекомендації щодо подальшого вдосконалення мобільного застосунка

У процесі розробки мобільного застосунка для систематизації інформації про проходження автомобілем технічних оглядів було виділено кілька додаткових функцій, що не вдалося реалізувати у рамках даного проєкту. Ці функції натомість було зведено у перелік напрямів подальшого вдосконалення мобільного застосунка.

4.3.1. *Додавання ширшого різноманіття даних*

У даній версії мобільного застосунка довідкові дані у словниках наявні у дуже малій кількості. Пропонується розширити переліки ремонтних робіт, замінених деталей, назв СТО та типів автомобілів. Ширше різноманіття цих даних дозволить користувачам обирати саме те, що їм потрібно, що покращить їх досвід використання застосунка.

Архітектура мобільного застосунка спроєктована таким чином, щоб розробник міг з легкістю додати підтримку інших типів автомобілів та пального. Пропонується додати підтримку роботи із електроавтомобілями, адже популярність цих автомобілів з кожним днем зростає – а, отже, збільшується кількість потенційних користувачів застосунка.

4.3.2. *Розширення модулів аналітики даних та статистики витрат*

Дана версія мобільного застосунку містить лише по одному варіанту представлення аналітики поширених проблем автомобілів, переліку робіт СТО із оцінками та статистики витрат. Пропонується доповнити модулі статистики та аналітики новими функціями, зокрема відображення та порівняння статистики витрат для кількох автомобілів користувача, пошук частоти виконання будь-якої роботи чи заміни будь-якої деталі певного автомобіля, перегляд складової оцінки виконаної роботи певної СТО. Така інформація допоможе користувачу більш точно оцінити витрати на технічне обслуговування власних автомобілів.

4.3.3. *Використання вбудованої системи сповіщення Android*

У розробленій версії мобільного застосунка сповіщення про майбутній техогляд надсилаються на електронну пошту користувачів. Так як застосунок є нативним та розроблений спеціально під ОС Android, то пропонується використати вбудовану систему сповіщення Android для

надсилання сповіщень про майбутні техогляди за допомогою push-сповіщень. Також пропонується надати можливість користувачу самостійно обирати, який тип сповіщення отримувати, і зробити цей вибір незалежним від вибору інших отримувачів сповіщення. Ця опція допоможе користувачу отримувати сповіщення вчасно та своєчасно проходити техогляд.

4.4.4. *Персоналізація даних*

Персоналізація є важливим та привабливим для користувача атрибутом сучасного програмного забезпечення. Тому пропонується додати можливість персоналізації даних облікового запису, автомобілів та записів про проходження техогляду користувачів, зокрема зміна імені користувача та іконки автомобіля. До того ж, архітектура мобільного застосунка дозволяє розширити перелік даних, що можуть зберігати користувачі: наприклад, додати можливість прикріпити до записів про техогляд фотографії акту виконаних робіт або коментарі. Ці зміни допоможуть зробити розроблений застосунок привабливим і корисним для користувача.

ВИСНОВКИ

Метою дипломного проєкту було створення мобільного застосунка для систематизації інформації про проходження автомобілем технічних оглядів.

Перед початком розробки було проведено аналіз предметної області, сформовано перелік основних проблем користувачів. За результатами аналізу існуючих конкурентів було виявлено ряд супроводжуючих проблем, з якими стикаються власники автомобілів, та представлено ідею вирішення поставленої проблеми.

Аналіз сучасних підходів до розробки мобільних застосунків показав доцільність створення нативного мобільного застосунка під операційну систему Android із клієнт-серверною архітектурою. Основною мовою програмування для розробки було обрано мову Kotlin, для розробки серверної частини і налаштування комунікації клієнта та сервера було обрано фреймворк Ktor. У якості системи керування базами даних було обрано MongoDB. Для авторизації та контролю сесії користувача було використано сервіс Firebase Auth, а для створення інтерфейсу користувача – набір інструментів та бібліотек Android Jetpack Compose.

Розроблений мобільний застосунок:

- Забезпечує розмежування доступу до даних за рахунок розділення авторизованих користувачів за ролями.
- Дозволяє систематизувати дані про автомобілі та пов’язані записи про проходження техогляду.
- Надсилає користувачам сповіщення про заплановані техогляди.
- Надає статистику витрат та іншу аналітику даних, що покликана допомогти користувачу оцінити власні витрати.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування проведено відповідно до затвердженої програми та методики тестування. Також було описано напрями для подальшого вдосконалення проєкту.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

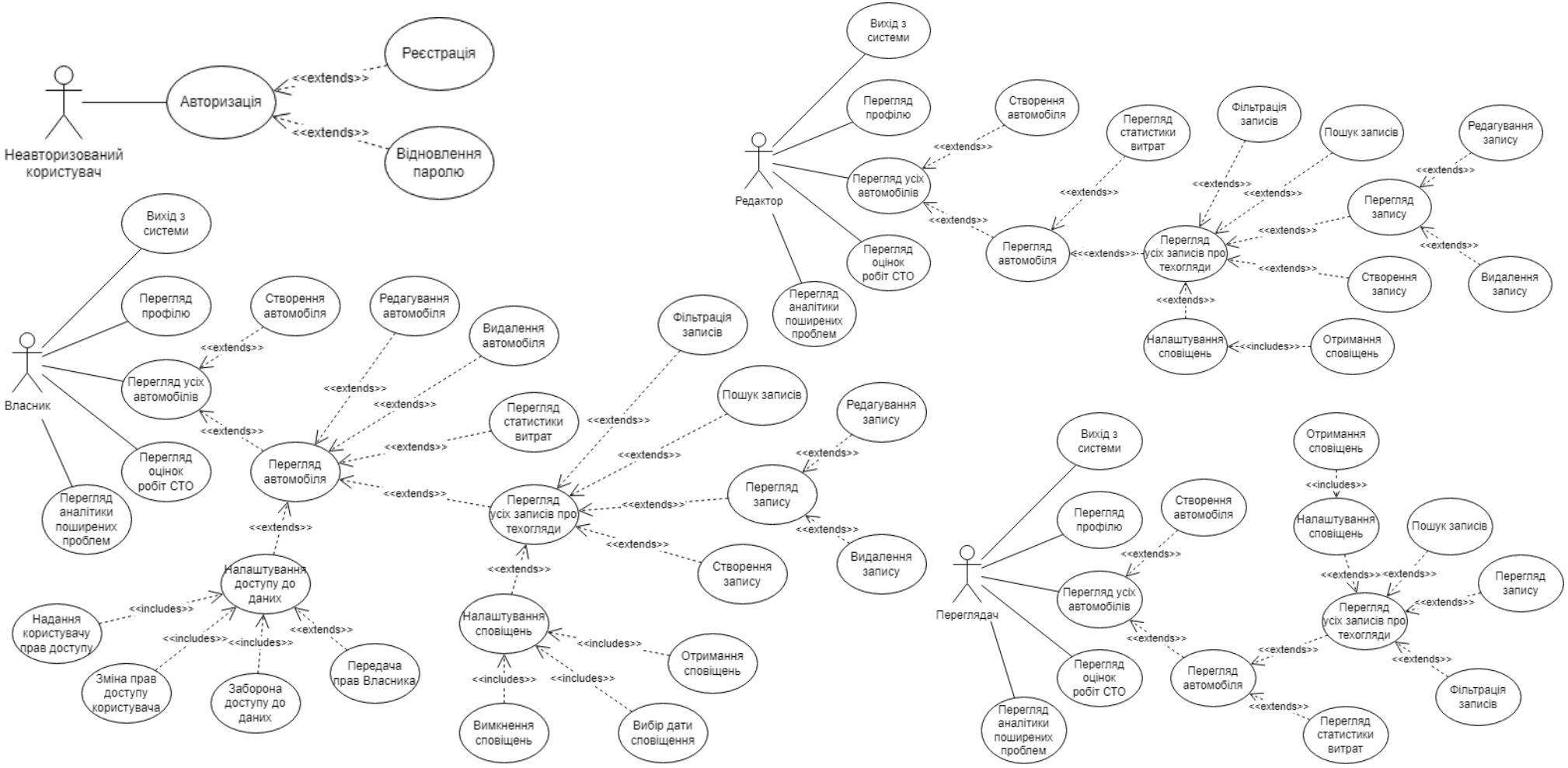
1. Відомості про транспортні засоби та їх власників. Набори даних [Електронний ресурс] // Єдиний державний веб-портал відкритих даних. — Режим доступу: https://data.gov.ua/dataset/06779371-308f-42d7-895e-5a39833375f0
2. В Киеве уровень автомобилизации превысил 400 авто на 1000 жителей [Електронний ресурс] // Auto-Consulting. — Режим доступу: https://www.autoconsulting.com.ua/article.php?sid=48496
3. Simply Auto: Car maintenance and Mileage Tracker app [Електронний ресурс]. — Режим доступу: https://simplyauto.app
4. Simple Fleet Maintenance Software Management System: AUTOsist [Електронний ресурс]. — Режим доступу: https://autosist.com
5. Drivvo App [Електронний ресурс]. — Режим доступу: https://www.drivvo.com/uk
6. Mobile Operating System Market Share Worldwide [Електронний ресурс] // StatCounter. — Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201303-202303
7. Android Auto [Електронний ресурс] // Android. — Режим доступу: https://www.android.com/auto
8. Gosling J. The Java Language Environment — A White Paper [Text] / James Gosling, Henry McGilton. — Mountain View, California : Sun Microsystems Computer Company, 1996. — 86 с.
9. Kotlin for server side [Електронний ресурс] // Kotlin Documentation. — Режим доступу: https://kotlinlang.org/docs/server-overview.html
10. JavaScript [Електронний ресурс] // MDN. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
11. About Python [Електронний ресурс] // Python.org. — Режим доступу: https://www.python.org/about
12. Ktor: Build Asynchronous Servers and Clients in Kotlin [Електронний ресурс] // Ktor Framework. — Режим доступу: https://ktor.io

13. What is MongoDB? [Електронний ресурс] // MongoDB. — Режим доступу: https://www.mongodb.com/what-is-mongodb
14. Fast NoSQL Key-Value Database – Amazon DynamoDB [Електронний ресурс] // Amazon Web Services. — Режим доступу: https://aws.amazon.com/dynamodb/
15. KMongo [Електронний ресурс]. — Режим доступу: https://litote.org/kmongo/
16. What is AWS [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/what-is-aws/
17. What is Azure – Microsoft Cloud Services [Електронний ресурс]. — Режим доступу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/
18. Why Google Cloud [Електронний ресурс]. — Режим доступу: https://cloud.google.com/why-google-cloud
19. Google I/O 2019: Empowering developers to build the best experiences on Android + Play [Електронний ресурс] // Android Developers Blog. — Режим доступу: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html
20. Why Compose [Електронний ресурс] // Android Developers. — Режим доступу: https://developer.android.com/jetpack/compose/why-adopt
21. Firebase Authentication [Електронний ресурс]. — Режим доступу: https://firebase.google.com/docs/auth

ДОДАТКИ

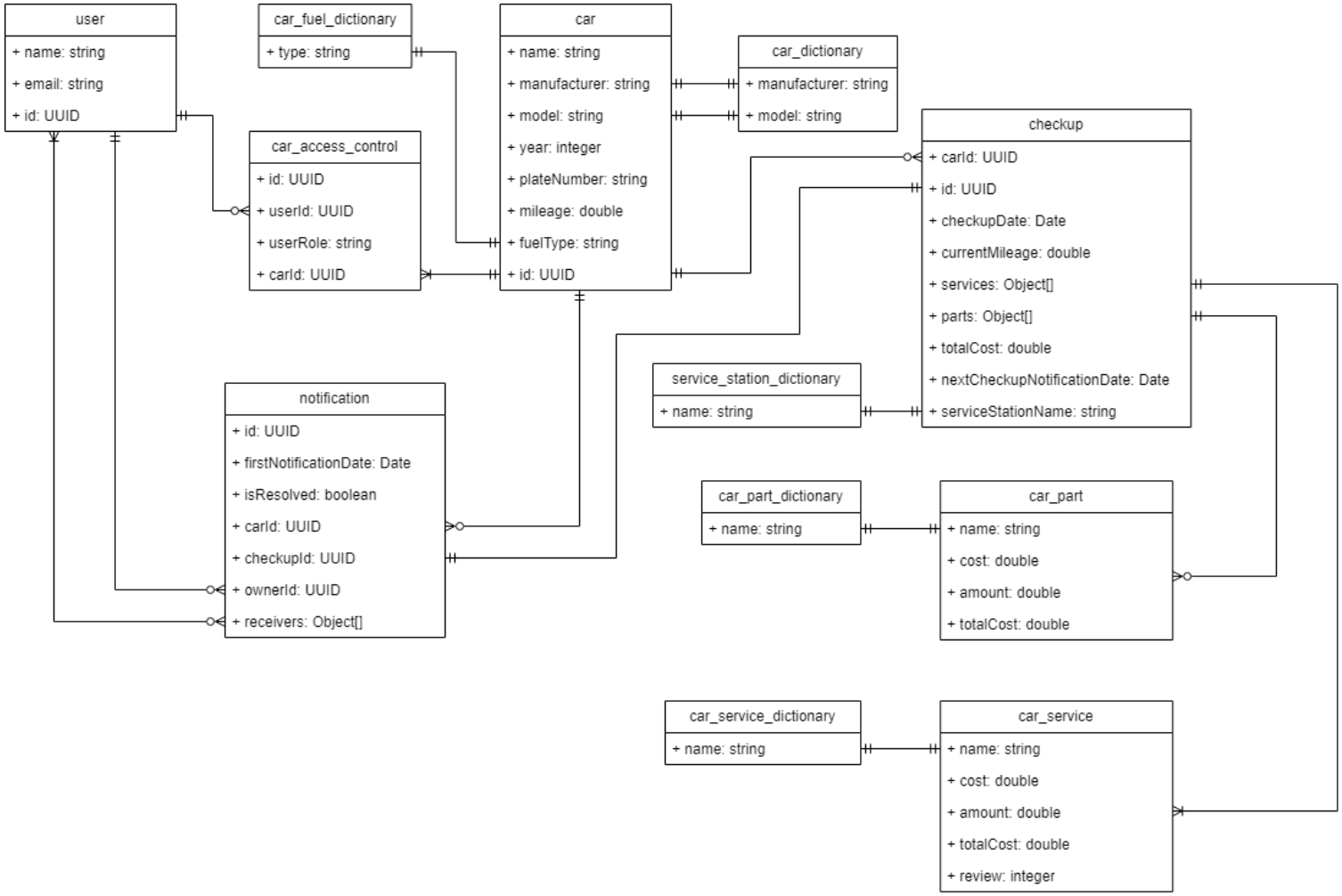
Додаток 1

Копії графічних матеріалів

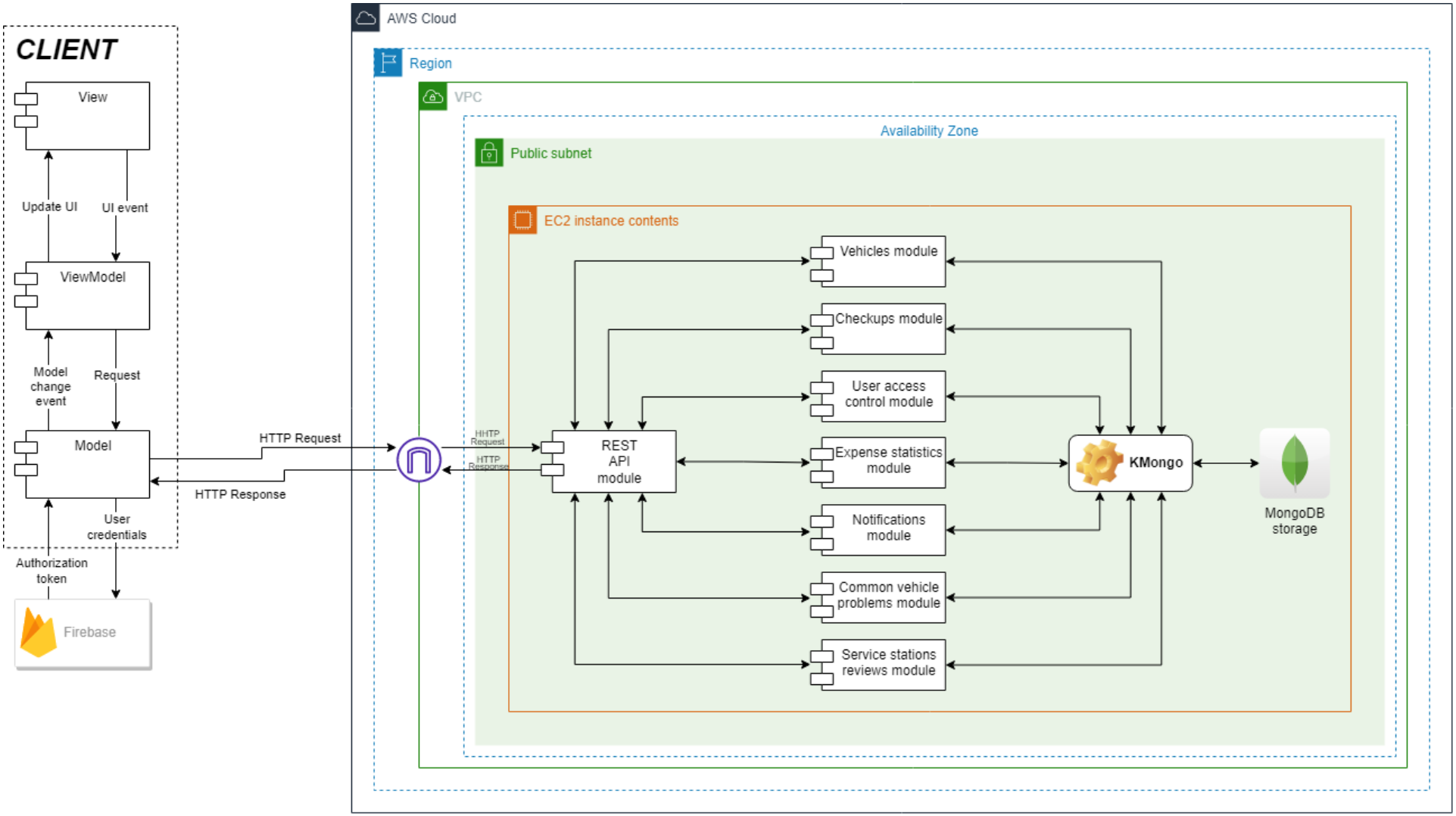


ДП.045440-06-99

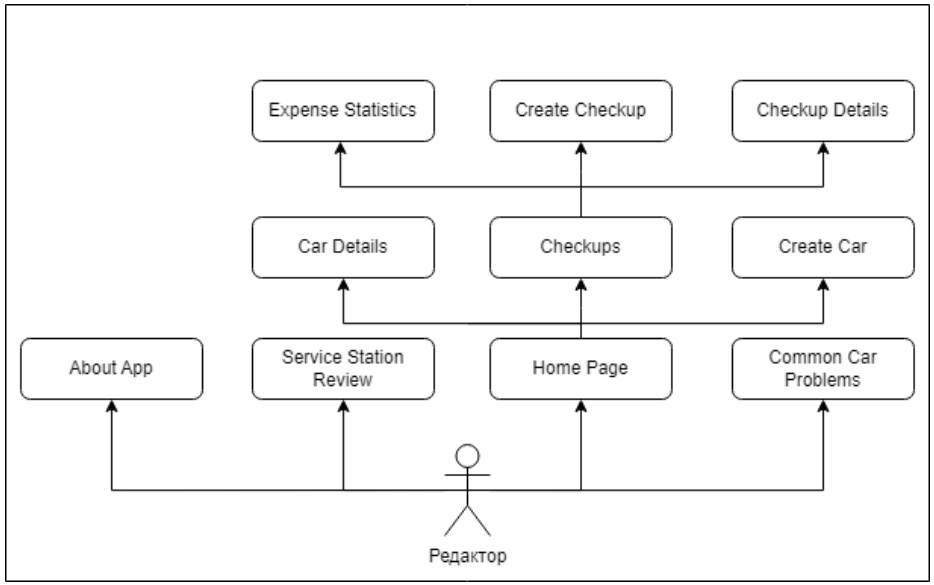
Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів. Функціональність мобільного застосунка. UML-діаграма прецедентів

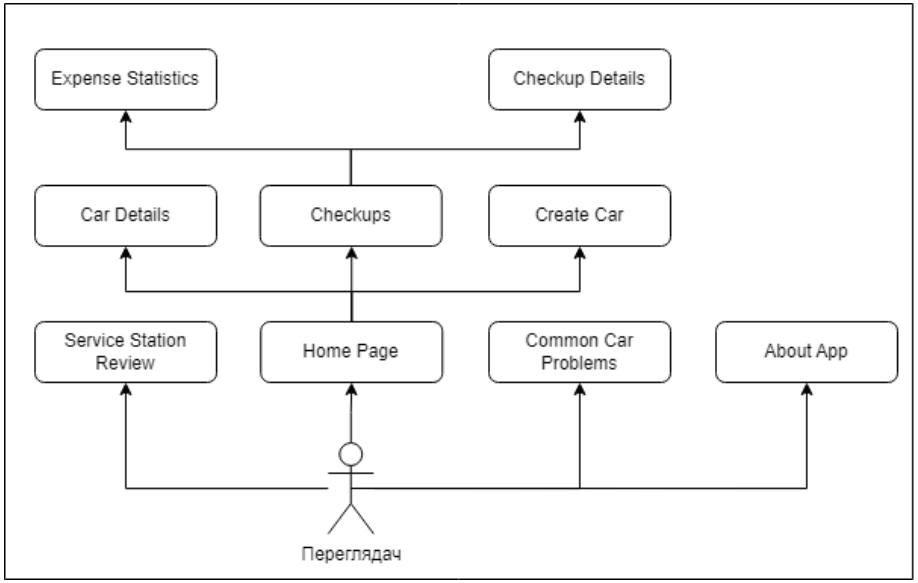


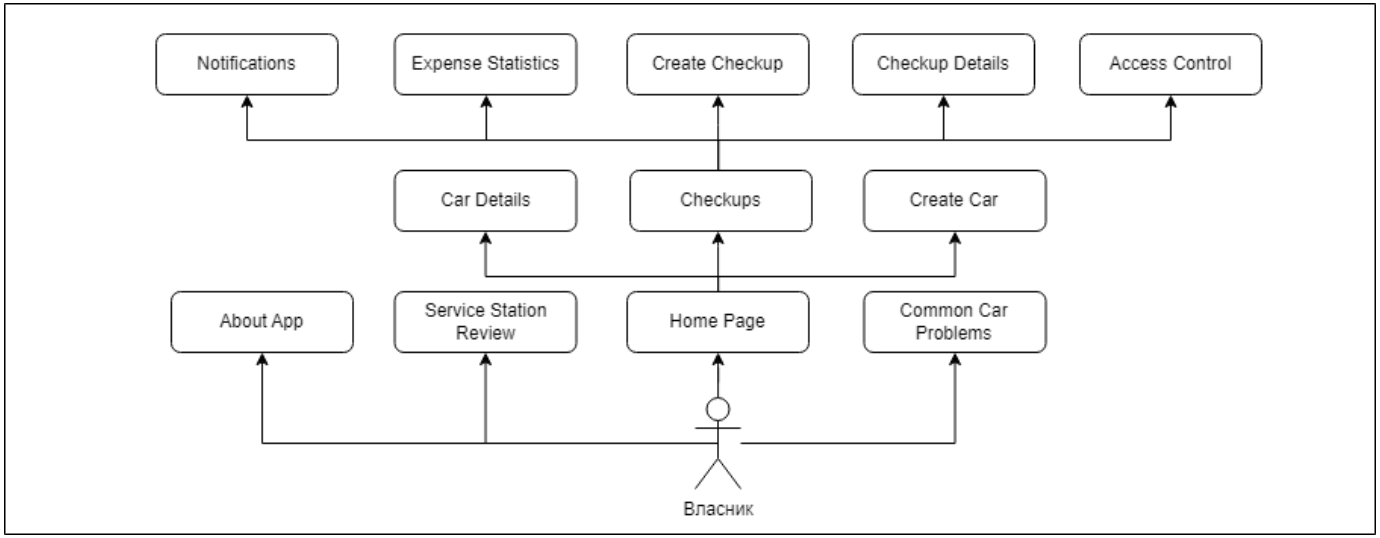
ДП.045440-07-99
Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів.
Структура бази даних. ER-діаграма

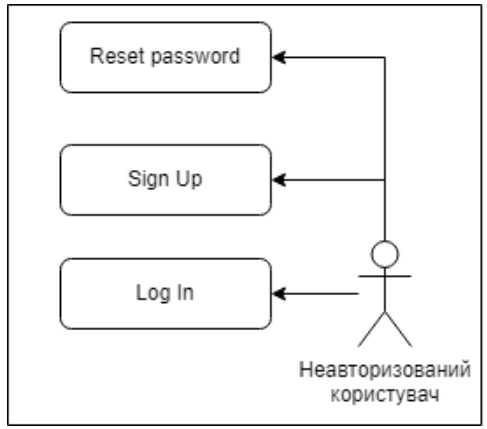


Структурна схема мобільного застосунка
Білоус А.Ю., група КП-93









Структура сторінок інтерфейсу
Білоус А.Ю., група КП-93

Додаток 2

Лістинг програми

repository/extensions/AccessType.kt

```
package com.aheadlight.carstory.repository.extensions

import com.aheadlight.carstory.dao.CarAccessControl
import com.aheadlight.carstory.repository.MongoRepo
import com.mongodb.client.model.DeleteManyModel
import com.mongodb.client.model.DeleteOneModel
import com.mongodb.client.model.InsertOneModel
import com.mongodb.client.model.ReplaceOneModel
import com.mongodb.client.model.WriteModel
import org.litote.kmongo.*

/**
 * Enum class containing names of allowed access rights
 *
 * @property rightName Name of access rights type
 */
enum class AccessRights(val rightName: String) {
    OWNER("owner"),
    EDITOR("editor"),
    VIEWER("viewer")
}

/**
 * Get given user's access types to all available cars
 *
 * @param userId User's ID
 * @return Car access types represented as list of DAO objects,
 * empty list indicated that there are no cars in the database :(
 */
suspend fun MongoRepo.getAccessTypesList(userId: String):
        List<CarAccessControl> {
    val collection = getInstance()
        .getCollection<CarAccessControl>(
            getColName(CarAccessControl::class.simpleName))

    return collection.find(CarAccessControl::userId eq userId)
        .toList()
}

/**
 * Get given user's access type to given car
 *
 * @param userId User's ID
 * @param carId Car's ID
 * @return Car access type represented as DAO object,
 * null indicated that given user doesn't have access to given car
 */
suspend fun MongoRepo.getAccessType(userId: String, carId: String):
        CarAccessControl? {
    val collection = getInstance()
        .getCollection<CarAccessControl>(
            getColName(CarAccessControl::class.simpleName))

    return collection.findOne(
        and(
            CarAccessControl::userId eq userId,
```

```
            CarAccessControl::carId eq carId
        )
    )
}

/**
 * Get data of users that have any access to given car
 *
 * @param carId Car's ID
 * @return List of all users who can access given car
 */
suspend fun MongoRepo.getUsersThatCanAccessCar(carId: String):
        MutableList<CarAccessControl> {
    val collection = getInstance()
        .getCollection<CarAccessControl>(
            getColName(CarAccessControl::class.simpleName))

    return collection.find(
        CarAccessControl::carId eq carId
    ).toList().toMutableList()
}

/**
 * Get IDs of users that have any access to given car
 *
 * @param carId Car's ID
 * @return List of IDs of all users who can access given car
 */
suspend     fun     MongoRepo.getIDListOfUsersThatCanAccessCar(carId:
String):
        MutableList<String> =
    getUsersThatCanAccessCar(carId).map { it.userId }.toMutableList()

/**
 * Get ID of given car's owner
 *
 * @param carId Car's ID
 * @return Owner's ID
 */
suspend fun MongoRepo.getCarOwner(carId: String): String {
    val collection = getInstance()
        .getCollection<CarAccessControl>(
            getColName(CarAccessControl::class.simpleName))

    return collection.findOne(
        and(
            CarAccessControl::carId eq carId,
            CarAccessControl::userRole                                  eq
AccessRights.OWNER.rightName
        )
    )?.userId ?: ""
}

/**
 * Checks if given user is given car's Owner
 *
 * @param userId User's ID
 * @param carId Car's ID
```

```
 * @return True if user has Owner access right
 */
suspend fun MongoRepo.isUserOwner(userId: String, carId: String):
Boolean =
    getAccessType(userId, carId)?.userRole ==
            AccessRights.OWNER.rightName

/**
 * Checks if given user is given car's Owner or Editor
 *
 * @param userId User's ID
 * @param carId Car's ID
 * @return True if user has Owner or Editor access right
 */
suspend fun MongoRepo.isUserOwnerOrEditor(
    userId: String, carId: String): Boolean {
    val roleToCheck = getAccessType(userId, carId)?.userRole
    return roleToCheck == AccessRights.OWNER.rightName ||
            roleToCheck == AccessRights.EDITOR.rightName
}

/**
 * Replace all access types for given car with provided values.
 * Automatically refreshes notifications receivers for given car.
 *
 * If only Owner data is provided, only changes car's Owner.
 * Always provide Owner data, otherwise Owner will be deleted!
 *
 * @param forCarWithId Car's ID
 * @param newTypes New values represented as list of DAO objects
 * (can be empty: in this way all access types will be deleted)
 * @return Operation result status
 */
suspend fun MongoRepo.replaceAccessTypesWith(newTypes:
List<CarAccessControl>,
        forCarWithId: String): Boolean {
    val collection = getInstance()
        .getCollection<CarAccessControl>(
            getColName(CarAccessControl::class.simpleName))

    // Separate car's Owner from the rest
    val (newOwner, rest) = newTypes
        .partition { it.userRole==AccessRights.OWNER.rightName }

    // Placeholder for all operations for bulk write
    val writeOperations: MutableList<WriteModel<CarAccessControl>> =
        mutableListOf()

    // Add delete operation for all previous entries (except Owner)
    writeOperations.add(DeleteManyModel(
        and(
            CarAccessControl::carId eq forCarWithId,
            CarAccessControl::userRole                                ne
AccessRights.OWNER.rightName
        )))

    // Process Owner entry
    if (newOwner.isEmpty()) {
```

```
        // Delete car's owner entry
        writeOperations.add(DeleteOneModel(
            and(
                CarAccessControl::carId eq forCarWithId,
                CarAccessControl::userRole                          eq
AccessRights.OWNER.rightName
            )
        ))
    }
    else {
        // Replace or create entry for car's owner
        writeOperations.add(ReplaceOneModel(
            and(
                CarAccessControl::carId eq forCarWithId,
                CarAccessControl::userRole                          eq
AccessRights.OWNER.rightName
            ),
            newOwner.first(),
            replaceUpsert()
        ))
    }

        // Process users of other access types

    // Add insert operation for each new entry (except Owner)
    for (newEntry in rest)
        writeOperations.add(InsertOneModel(newEntry))

    // Execute all operations with access type entries
    val isWriteSuccessful =
        collection.bulkWrite(writeOperations).wasAcknowledged()

    return if (isWriteSuccessful)
        // Update notification receivers for given car
        refreshReceivers(forCarWithId)
    else false
}
```

repository/extensions/Notification.kt

```
package com.aheadlight.carstory.repository.extensions

import com.aheadlight.carstory.dao.Checkup
import com.aheadlight.carstory.dao.Notification
import com.aheadlight.carstory.repository.MongoRepo
import com.aheadlight.carstory.repository.getCalendar
import com.aheadlight.carstory.repository.getTodayDate
import org.litote.kmongo.*
import java.util.*
import java.util.concurrent.TimeUnit
import kotlin.math.abs

/**
 * Get notifications for given car that are either with
 * notification date set in the future or aren't resolved
 *
 * @param carId Car's ID
 * @return Given car's notifications represented as list of DAO objects
```

```
 */
suspend    fun    MongoRepo.getNotificationsList(carId:    String):
List<Notification> {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.find(
        and(
            Notification::carId eq carId,
            or(
                Notification::isResolved eq false,
                Notification::firstNotificationDate               gte
getTodayDate()
            )
        )).toList()
}

/**
 * Get all notifications that have to be sent to users at the current
day.
 *
 * Select criteria: notification isn't resolved yet
 * and its first notification date is set to current date or earlier
 *
 * @return All unresolved notifications represented as list of DAO
objects
 */
suspend fun MongoRepo.getNotificationsToNotify(): List<Notification>
{
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.find(
        and(
            Notification::isResolved eq false,
            Notification::firstNotificationDate lte getTodayDate()
        )).toList()
}

/**
 * Get given notification
 *
 * @param notificationId Notification's ID
 * @return Notification with given ID represented as DAO object
 */
suspend   fun   MongoRepo.getNotification(notificationId:   String):
Notification? {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.findOne(Notification::id eq notificationId)
}

/**
 * Create notification for given checkup entry
```

```
 *
 * @param notificationDate Notification date
 * @param checkupId Checkup's ID
 * @param carId ID of a car for which checkup was conducted
 * @return Operation result status
 */
suspend fun MongoRepo.createNotification(notificationDate: Date,
        checkupId: String, carId: String): Boolean {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    // Create notification entry to insert
    val dataToInsert = Notification(
        firstNotificationDate = notificationDate,
        isResolved = false,
        carId = carId,
        checkupId = checkupId,
        ownerId = getCarOwner(carId),
        receivers = getIDListOfUsersThatCanAccessCar(carId)
    )

    return collection.insertOne(dataToInsert).wasAcknowledged()
}

/**
 * Calculate optimal notification date for given car
 *
 * @param carId Car's ID
 * @return Date object with notification date value
 */
suspend fun MongoRepo.calculateNotificationDate(carId: String): Date
{
    // Get 5 last checkups in ascending order
    val last5Checkups = getCheckupsList(carId, 5)

    // Return date half a year from given checkup date
    // if there are not enough entries to analyze
    if (last5Checkups.size<2) {
        val calendar = getCalendar()
        // Set default notification date 6 months forward from current
day
        calendar.add(Calendar.MONTH, 6)
        return calendar.time
    }

        // Calculate date programmatically

    // Accumulated difference of all dates
    var timeDiff: Long = 0

    // Accumulate date difference
    for (i in 0 until last5Checkups.size-1)
        timeDiff += abs(last5Checkups[i+1].checkupDate.time -
                last5Checkups[i].checkupDate.time)

    // Calculate average of accumulated date
    timeDiff /= (last5Checkups.size-1)
```

```
    // Calculate how many days later the notification date should be
    val daysToAdd = TimeUnit.DAYS
        .convert(timeDiff, TimeUnit.MILLISECONDS).toInt()

    println("Notification test. Days to add: $daysToAdd")

    val calendar = getCalendar()
    // Set notification date away from current day for daysToAdd days
    calendar.add(Calendar.DATE, daysToAdd)

    println("Notification test. Date: ${calendar.time}")

    return calendar.time
}

/**
 * Update given checkup's first notification date for both
 * Checkup and Notification entries
 *
 * @param checkupId Checkup's ID
 * @param newDate New first notification date
 * @return Operation result status
 */
suspend fun MongoRepo.updateNotifyDateEverywhere(checkupId: String,
        newDate: Date): Boolean {
    val currCheckupDate = getCheckup(checkupId)?.checkupDate

    return if (currCheckupDate!=null) {
        if (currCheckupDate<newDate)
            // Update notification date in Checkup and Notification
entries
            updateNotifyDateInNotifyEntry(checkupId, newDate) &&
                    updateNotifyDateInCheckupEntry(checkupId,
newDate)

        // Can't update if new notification date is
        // earlier or equal to checkup date
        else false
    }
    else false
}

/**
 * Update first notification date for Notification entry
 * (without updating its Checkup entry)
 *
 * @param checkupId Checkup's ID
 * @param newDate New checkup's notification date
 * @return Operation result status
 */
suspend    fun    MongoRepo.updateNotifyDateInNotifyEntry(checkupId:
String,
        newDate: Date): Boolean {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))
```

```
    return collection.updateOne(
        Notification::checkupId eq checkupId,
        setValue(Notification::firstNotificationDate, newDate)
    ).wasAcknowledged()
}

/**
 * Update first notification date for Checkup entry
 * (without updating its Notification entry)
 *
 * @param checkupId Checkup's ID
 * @param newDate New checkup's notification date
 * @return Operation result status
 */
suspend    fun    MongoRepo.updateNotifyDateInCheckupEntry(checkupId:
String,
        newDate: Date): Boolean {
    val collection = getInstance()
        .getCollection<Checkup>(
            getColName(Checkup::class.simpleName))

    return collection.updateOne(
        Checkup::id eq checkupId,
        setValue(Checkup::nextCheckupNotificationDate, newDate)
    ).wasAcknowledged()
}

/**
 * Update isResolved state for given notification
 *
 * @param notificationId Notification's ID
 * @param newState New notification's isResolved state
 * @return Operation result status
 */
suspend fun MongoRepo.updateNotificationResolvedState(notificationId:
String,
        newState: Boolean): Boolean {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.updateOne(
    Notification::id eq notificationId,
        setValue(Notification::isResolved, newState)
    ).wasAcknowledged()
}

/**
 * Updates notification receivers for given car
 *
 * @param carId Car's ID
 * @return Operation result status
 */
suspend fun MongoRepo.refreshReceivers(carId: String): Boolean {
    // Get given car's Owner
    val ownerId = getCarOwner(carId)

    // Get all users that have any access to given car
```

```
    val receiversList = getIDListOfUsersThatCanAccessCar(carId)

    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    // Update owner and receivers for all notifications of given car
    return collection.updateMany(
        Notification::carId eq carId,
        set(SetTo(Notification::ownerId, ownerId),
            SetTo(Notification::receivers, receiversList))
    ).wasAcknowledged()
}

/**
 * Delete notification for given checkup
 *
 * @param checkupId Checkup's ID
 * @return Operation result status
 */
suspend fun MongoRepo.deleteCheckupNotification(checkupId: String):
Boolean {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.deleteOne(Notification::checkupId eq checkupId)
        .wasAcknowledged()
}

/**
 * Delete all notifications for given car
 *
 * @param carId Car's ID
 * @return Operation result status
 */
suspend fun MongoRepo.deleteAllNotificationsFor(carId: String):
Boolean {
    val collection = getInstance()
        .getCollection<Notification>(
            getColName(Notification::class.simpleName))

    return collection.deleteMany(Notification::carId eq carId)
        .wasAcknowledged()
}
```

repository/extensions/Analytics.kt

```
package com.aheadlight.carstory.repository.extensions

import com.aheadlight.carstory.dao.CarDictionary
import com.aheadlight.carstory.dto.CommonVehicleProblemsDTO
import com.aheadlight.carstory.dto.ServiceStationReviewsDTO
import com.aheadlight.carstory.repository.MongoRepo

/**
 * Get 10 most frequent services for car with given type,
 * sorted by frequency in descending order
```

```
 *
 * @param carData DAO object representing car's manufacturer and model
 * @return Service names and their frequencies
 * represented as list of DTO objects
 */
suspend fun MongoRepo.getCommonVehicleServices(
    carData: CarDictionary): List<CommonVehicleProblemsDTO> {
    // Get map of service names and their frequency,
    // sort them in descending order by frequency,
    // and take a list of first 10 pairs of name and frequency values
    val services = getServicesListForCarWithType(carData)
        .groupingBy { it.name }.eachCount()
        .toList().sortedByDescending { it.second }.take(10)

    // Placeholder for result list
    val result = mutableListOf<CommonVehicleProblemsDTO>()

    // Transform list of pairs in result list suitable for transfer
    services.forEach { entry ->
        result.add(CommonVehicleProblemsDTO(entry.first,
entry.second))
    }

    return result
}

/**
 * Get 10 most frequently replaced parts for car with given type,
 * sorted by frequency in descending order
 *
 * @param carData DAO object representing car's manufacturer and model
 * @return Part names and their replacement frequencies
 * represented as list of DTO objects
 */
suspend fun MongoRepo.getCommonVehicleParts(
    carData: CarDictionary): List<CommonVehicleProblemsDTO> {
    // Get map of part names and their frequency,
    // sort them in descending order by frequency,
    // and take a list of first 10 pairs of name and frequency values
    val parts = getPartsListForCarWithType(carData)
        .groupingBy { it.name }.eachCount()
        .toList().sortedByDescending { it.second }.take(10)

    // Placeholder for result list
    val result = mutableListOf<CommonVehicleProblemsDTO>()

    // Transform list of pairs in result list suitable for transfer
    parts.forEach { entry ->
        result.add(CommonVehicleProblemsDTO(entry.first,
entry.second))
    }

    return result
}

/**
 * Get average reviews of services conducted by given service station,
 * sorted by review in descending order
```

```
 *
 * @param stationName Service station name
 * @return List of DTO objects with service name and its average review
 */
suspend fun MongoRepo.getServiceStationReviews(
    stationName: String): List<ServiceStationReviewsDTO> {
    // Get list of all services conducted by given service station
    val stationServices = getServicesByStation(stationName)

    // Calculate average review for each service
    val reviewsAvg = stationServices
        // Group services by name
        .groupBy { it.name }
        // Iterate over each group
        .map { s ->
            // Calculate sum of all reviews and divide by group's size
            (s.value.sumOf { it.review ?: 0 }).toDouble() /
(s.value.size)
        }

    // Placeholder for result
    val result = mutableListOf<ServiceStationReviewsDTO>()

    // Add services with calculated reviews
    stationServices.forEachIndexed { index, service ->
        result.add(ServiceStationReviewsDTO(service.name,
            if(index>reviewsAvg.size-1)
                null
            else if (reviewsAvg[index]==0.0)
                null
            else
                reviewsAvg[index]))
    }

    // Get all possible services
    val servicesDB = getServiceTypes()

    // Add empty values for services without reviews
    servicesDB.forEach { a ->
        if (result.none { it.serviceName==a.name })
            result.add(ServiceStationReviewsDTO(a.name, null))
    }

    // Return sorted list of formed DTO objects
    return result.sortedByDescending { it.serviceAverageReview }
}
```

controller/extensions/Notification.kt

```
package com.aheadlight.carstory.controller.extensions

import com.aheadlight.carstory.controller.*
import com.aheadlight.carstory.dto.NotificationDTO
import com.aheadlight.carstory.repository.FirebaseSDK
import com.aheadlight.carstory.repository.LocalProperties
import com.aheadlight.carstory.repository.extensions.*
import io.ktor.http.*
import io.ktor.server.application.*
```

```
import io.ktor.server.request.*
import io.ktor.server.response.*
import org.apache.commons.mail.DefaultAuthenticator
import org.apache.commons.mail.HtmlEmail
import java.text.SimpleDateFormat
import java.util.*

/**
 * Send notifications to all users
 *
 */
suspend fun Controller.notifyUsers() {
    // Get all notifications to be sent
    val notifications = getRepo().getNotificationsToNotify()

    // Process every notification
    notifications.forEach {entry ->
        // Cache car data
        val carData = getRepo().getCar(entry.carId)

        // Cache checkup data
        val checkupData = getRepo().getCheckup(entry.checkupId)

        // Cache owner's data
        val ownerData = getRepo().getUserDataByID(entry.ownerId)

        // Compose email subject
        val subject = "Нагадування про майбутній техогляд " +
                "для автомобіля ${carData?.name}"

        // Compose once same body part
        // for all receivers of given notification
        val body = StringBuilder()

        body.append("<p>${checkupData?.checkupDate?.let {
            SimpleDateFormat("dd.M.yyyy",
                Locale.getDefault()).format(it) }} " +
                "автомобіль ${carData?.getInfo()} проходив техогляд")

        if (checkupData?.serviceStationName!=null)
            body.append(" на СТО ${checkupData.serviceStationName}")

        body.append(".</p><p>Тоді було було виконано такі
роботи:</p><ul>")

        checkupData?.services?.forEach {service ->
            body.append("<li>${service.name}</li>")
        }
        body.append("</ul>")

        // Process every notification receiver
        entry.receivers.forEach {receiver ->
            // Get receiver's data
            // (use cache if it's owner)
            val userData = if(receiver!=entry.ownerId)
                getRepo().getUserDataByID(receiver)
            else ownerData
```

```
            // Compose email message
            val msg = StringBuilder()
            msg.append("<p>Вітаємо, ${userData?.name}!</p>")
            msg.append(body.toString())

            // Add part dependent on user access right
            if(receiver==entry.ownerId) {
                msg.append("<p>Не забудьте найближчим часом " +
                        "пройти техогляд!</p>")
                msg.append("<p>Щоб більше не отримувати сповіщення "
+
                        "про цей техогляд, зайдіть у розділ
\"Сповіщення\" " +
                        "для цього автомобіля та натисніть на " +
                        "іконку годинника біля даних цього
сповіщення</p>")
            }
            else {
                msg.append("<p>Не забудьте нагадати власнику " +
                        "автомобіля найближчим часом пройти
техогляд!</p>")
                msg.append("<p>Щоб більше не отримувати сповіщення "
+
                        "про цей техогляд, зв'яжіться із власником "
+
                        "цього автомобіля: ${ownerData?.email}</p>")
            }

            // Setup email sender
            val email = HtmlEmail()
            email.hostName = "smtp.googlemail.com"
            email.setSmtpPort(465)
            email.setCharset("UTF-8")
            email.setAuthenticator(DefaultAuthenticator(
                LocalProperties.getProperty("NOTIFY_EMAIL"),
                LocalProperties.getProperty("NOTIFY_PASSWORD")
            ))

email.setFrom(LocalProperties.getProperty("NOTIFY_EMAIL"))
            email.isSSLOnConnect = true

            // Setup email content
            email.subject = subject
            email.setMsg(msg.toString())

            // Set email receiver
            email.addTo(userData?.email)

            // Send email
            email.send()
        }
    }
}
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ СИСТЕМАТИЗАЦІЇ ІНФОРМАЦІЇ ПРО ПРОХОДЖЕННЯ АВТОМОБІЛЕМ ТЕХНІЧНИХ ОГЛЯДІВ

Виконав: студентка групи КП -93 Білоус Анна Юріївна

Керівник: доцент кафедри ПЗКС, к.т.н., Рибачок Наталія Антонівна

Київ – 2023

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів.

Завдання:

1. *Аналіз*
   - Проаналізувати існуючі аналоги.
   - Висунути вимоги до програмного забезпечення.
2. *Розроблення*
   - Обґрунтувати вибір технологій розроблення.
   - Розробити архітектуру застосунка.
3. *Результати*
   - Протестувати розроблений застосунок.
   - Сформувати перелік напрямів подальшого вдосконалення розробленого застосунка.

АКТУАЛЬНІСТЬ

1. Станом на 2021 рік кожен п’ятий українець має легковий автомобіль.
2. Попит на легкові автомобілі відновлюється: відповідно до відкритих даних МВС України, автомобільний ринок легкових автомобілів 2022 року становить 72,6% від ринку 2021 року.
3. Власники автомобілів мають труднощі з підтримкою задовільного стану своїх транспортних засобів.
4. Наявні рішення недосконалі та неповністю вирішують проблему користувача:
    1. мають надлишкову функціональність, не пов’язану з техоглядами;
    2. для збереження історії техогляду пропонують лише роботу із запрограмованими базовими сервісами;
    3. не націлені на українського користувача.

ОГЛЯД ПРОБЛЕМИ, ЯКУ ВИРІШУЄ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

АНАЛОГИ

Simply Auto

AUTOsist

drivvo

ПОРІВНЯННЯ З АНАЛОГАМИ

Критерій	Simply Auto	AUTOsist	Drivvo	Розроблене ПЗ
Україномовна версія	+	–	+	+
Безоплатна організація роботи з кількома автомобілями	+	–	+	+
Необмежений розподілений доступ до даних користувача	+	+	–	+
Розділення інформації про виконані ремонтні роботи та замінені деталі	–	+	–	+
Безоплатна персоналізація записів про техогляд	–	+	+	+
Виділення складових витрат	–	+	–	+
Наявність журналу пального	+	+	+	–

ФУНКЦІОНАЛЬНІ ВИМОГИ

1. Підтримка наступних ролей: Власник, Редактор, Переглядач.
2. Керування власними автомобілями (створення, перегляд, редагування, видалення) для користувачів типу Власник.
3. Збереження та керування записами про проведені ремонтні роботи та замінені деталі власних автомобілів (створення, фільтрація та пошук, перегляд, редагування, видалення) для користувачів типу Власник або Редактор.
4. Підтримка роботи з одними даними кількох типів користувачів із забезпеченням різних прав доступу (Власник, Редактор, Переглядач).

ФУНКЦІОНАЛЬНІ ВИМОГИ

5. Представлення статистики витрат для обраних автомобілів для всіх типів користувачів.
6. Сповіщення для всіх типів користувачів про майбутні обстеження автомобілів на основі інформації про попередні проведені обстеження.
7. Представлення для всіх типів користувачів знеособленої аналітики поширених проблем, пов’язаних із певним типом автомобіля.
8. Представлення для всіх типів користувачів переліку робіт, які найчастіше проводить певна СТО, з оцінками виконання цих робіт іншими користувачами.

ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

android

Kotlin

Jetpack Compose

Ktor

ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ СЕРВЕРНОЇ ЧАСТИНИ

Kotlin

Ktor

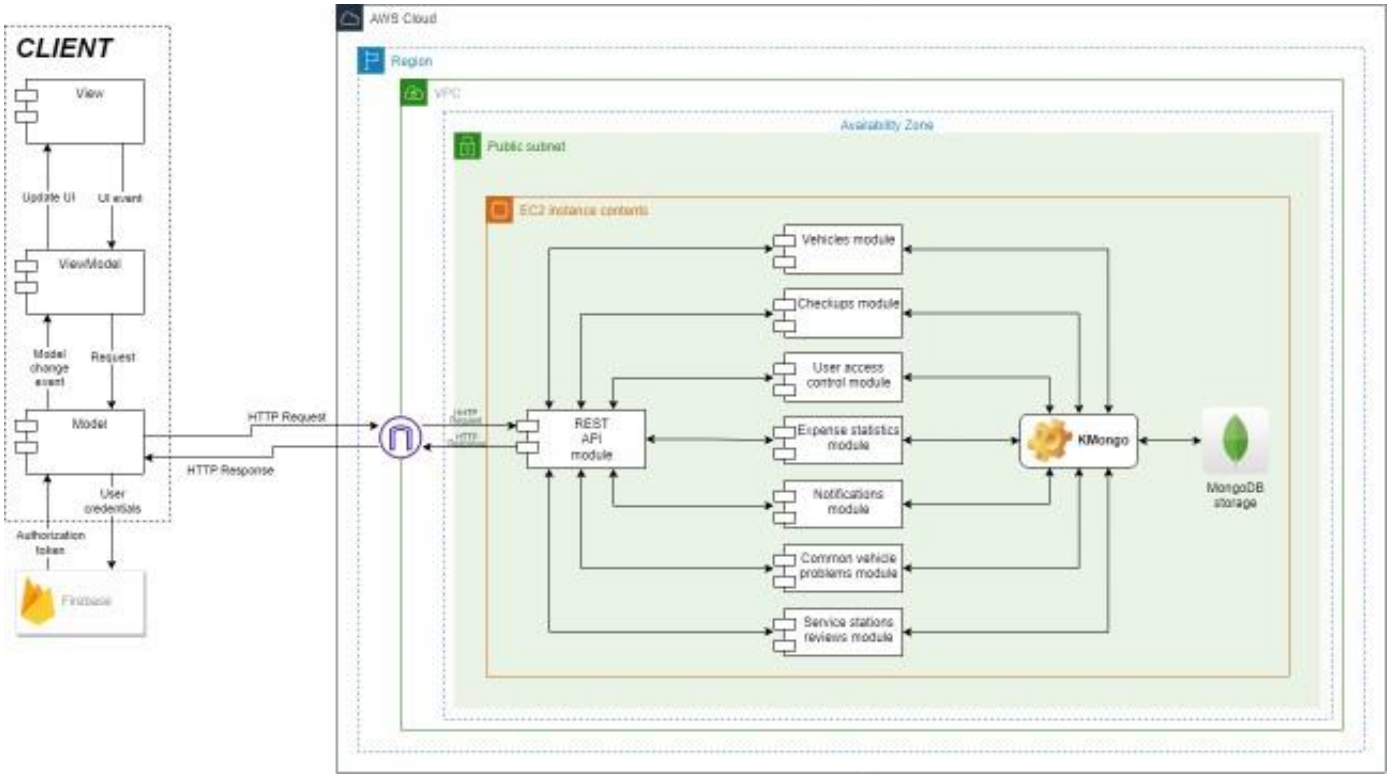
mongoDB.

Firebase Authentication

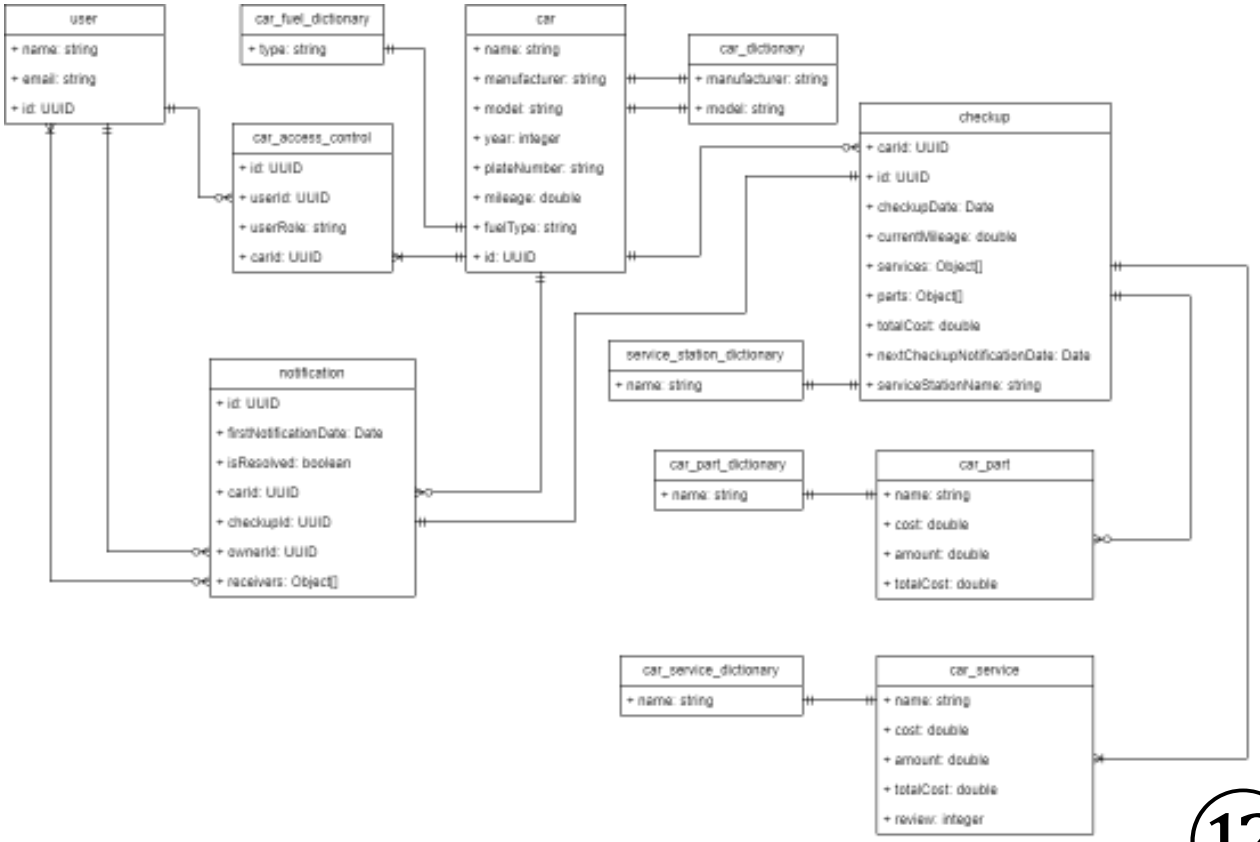
KMongo

Amazon EC2

АРХІТЕКТУРА СИСТЕМИ

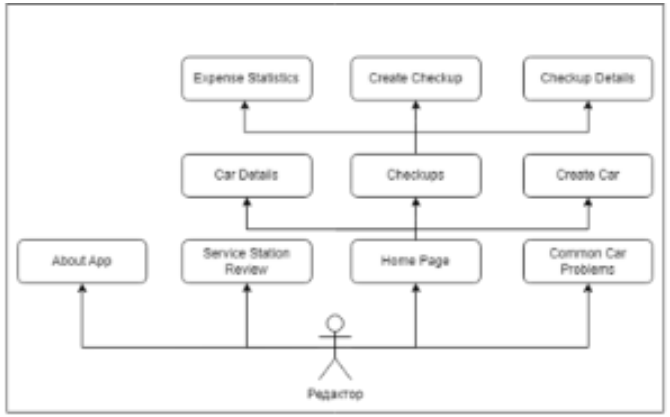


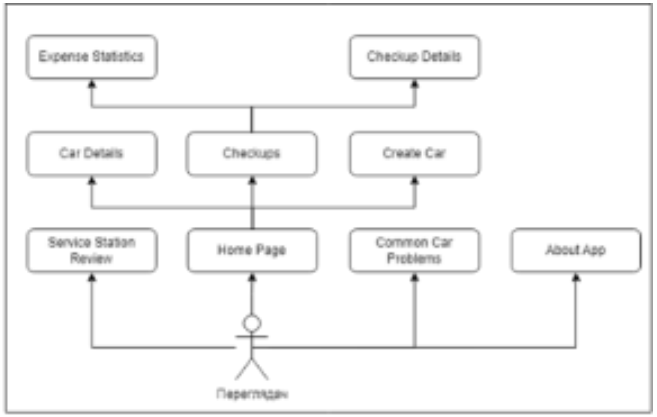
СТРУКТУРА БАЗИ ДАНИХ

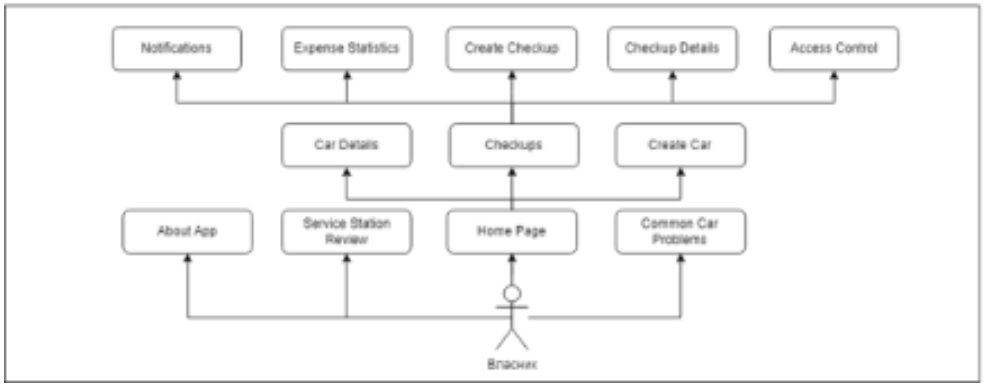


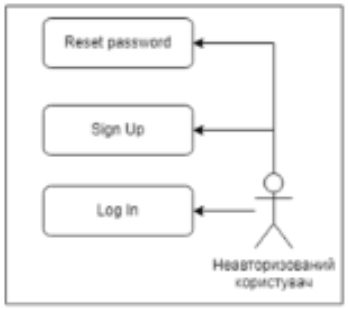
ФУНКЦІОНАЛЬНІСТЬ МОБІЛЬНОГО ЗАСТОСУНКУ

СТРУКТУРА СТОРІНОК ІНТЕРФЕЙСУ







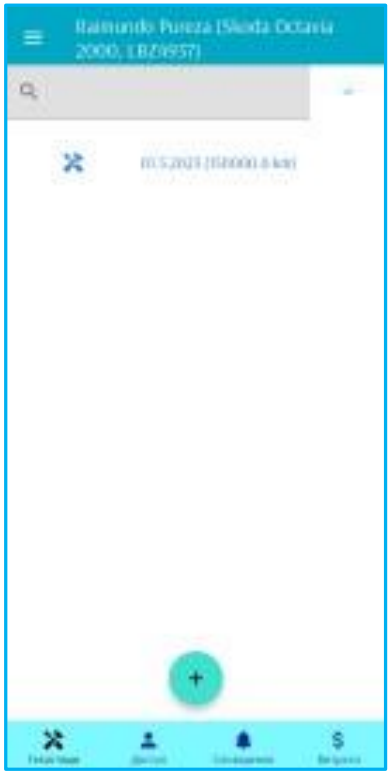


ТЕСТУВАННЯ

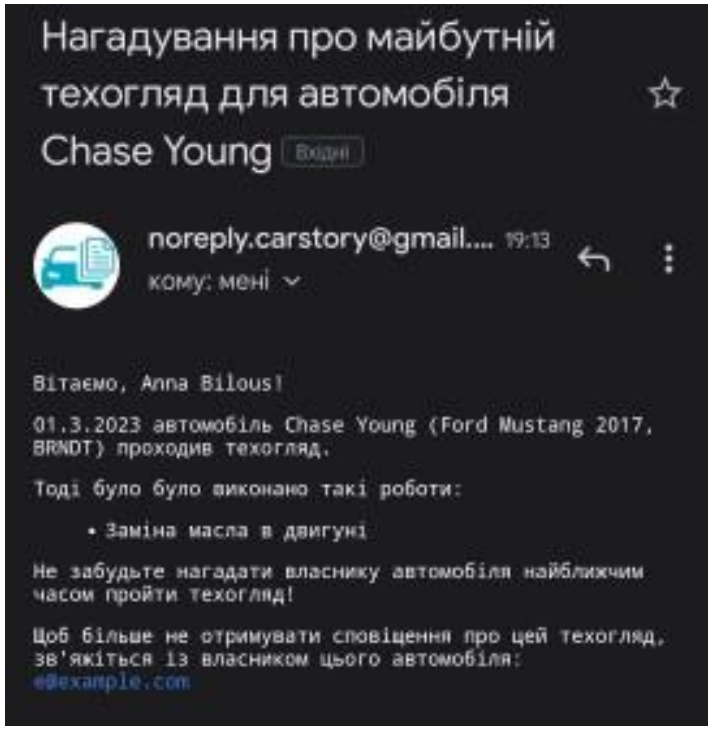
Повний процес тестування проходить у такому порядку:

1. Статичне тестування коду.
2. Динамічне ручне тестування полів вводу на граничні та неможливі значення.
3. Динамічне ручне тестування на відповідність поставленим функціональним вимогам.
4. Тестування інтерфейсу за різної роздільній здатності екрану.
5. Тестування роботи на пристроях з різними версіями операційної системи Android.
6. Тестування при конкурентному навантаженні на систему з боку декількох користувачів.
7. Тестування стабільності роботи за різних зовнішніх умов, зокрема при відсутності підключення до мережі Інтернет.

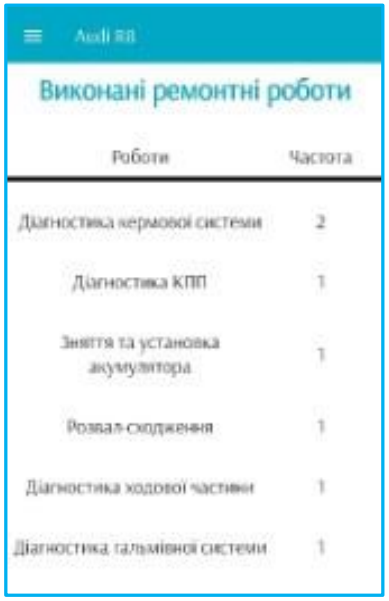
ПРИКЛАД РОБОТИ ЗАСТОСУНКУ. ПЕРЕГЛЯД ДАНИХ АВТОМОБІЛІВ ТА ЗАПИСІВ ПРО ТЕХОГЛЯД



ПРИКЛАД РОБОТИ ЗАСТОСУНКУ. СПОВІЩЕННЯ КОРИСТУВАЧІВ ПРО МАЙБУТНІ ТЕХОГЛЯДИ



ПРИКЛАД РОБОТИ ЗАСТОСУНКУ. АНАЛІТИКА ПОШИРЕНИХ ПРОБЛЕМ АВТОМОБІЛІВ



ПРИКЛАД РОБОТИ ЗАСТОСУНКУ. АНАЛІТИКА РОБІТ СТО ІЗ ОЦІНКАМИ ВИКОНАННЯ

НАПРЯМИ РОЗВИТКУ

- розширення переліків даних у словниках;
- підтримка роботи із електроавтомобілями.

- статистика витрат для кількох автомобілів одночасно;
- перегляд складової оцінки виконаної роботи певної СТО.

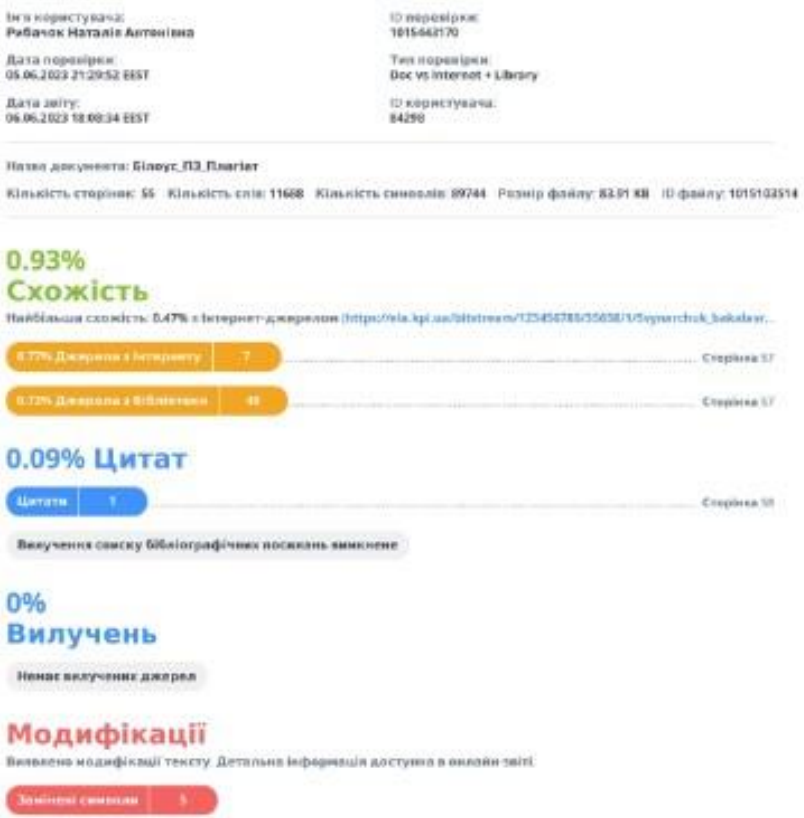
- push-сповіщення;
- вибір між сповіщеннями на пошту та push.

- персоналізація даних користувача;
- можливість додати до записів про техогляд фотографії або коментарі.

ВИСНОВКИ

1. Проаналізовано існуючі аналоги : Simply Auto, AUTOsist , Drivvo .
2. Сформовано вимоги до мобільного застосунка.
3. Обґрунтовано вибір технологій розроблення клієнтської та серверної частини застосунка.
4. Розроблено клієнт-серверну архітектуру застосунка, структуру бази даних (10 основних колекцій).
5. Реалізовано наступну функціональність застосунка:
   - функціональність для роботи із даними автомобілів та їх записів про проходження техогляду;
   - функціональності розмежування доступу користувачів, сповіщення, формування статистики витрат та знеособленої аналітики.
6. Проведено тестування окремих модулів та інтеграційне тестування модулів програми, ручне тестування інтерфейсу.
7. Сформовано перелік з чотирьох напрямів подальшого вдосконалення розробленого застосунка.

ПЕРЕВІРКА НА УНІКАЛЬНІСТЬ



Ім'я користувача:
Рябачок Наталія Антонівна

ID перевірки:
1015662170

Дата перевірки:
05.06.2023 21:29:52 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.06.2023 18:00:34 EEST

ID користувача:
84290

Назва документа: **Бізнус_ПЗ_Лаб1**

Кількість сторінок: **55** Кількість слів: **11688** Кількість символів: **89744** Розмір файлу: **83.91 KB** ID файлу: **1015103514**

0.93%
Схожість

Найбільша схожість: **0.47%** з Інтернет-джерелом

0.09% Цитат

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн звіті.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2022 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ СИСТЕМАТИЗАЦІЇ ІНФОРМАЦІЇ ПРО ПРОХОДЖЕННЯ АВТОМОБІЛЕМ ТЕХНІЧНИХ ОГЛЯДІВ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна БІЛОУС

2022

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів, написаний мовою програмування Kotlin із використанням фреймворку Ktor для розробки серверної частини та набору інструментів і бібліотек Android Jetpack Compose для розробки користувацького інтерфейсу.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Працездатність функцій реєстрації, автентифікації та відновлення паролю.
2. Попередження неавторизованого доступу та доступу з невідповідною роллю.
3. Функціональна працездатність елементів сторінок мобільного застосунка.
4. Доступ до бази даних через серверну частину.
5. Коректна робота функцій керування (створення, перегляд, редагування, видалення) для даних автомобілів, записів про проведені техогляди, записів про права доступу користувачів до даних автомобілів.
6. Робота сервера із модулем, що відповідає за сповіщення користувачів про майбутні техогляди на електронну пошту.
7. Робота сервера із модулями, що відповідають за формування статистики витрат, знеособленої аналітики поширених проблем автомобілів та знеособленого переліку робіт СТО із оцінками виконання іншими користувачами.
8. Забезпечення коректної обробки REST запитів та відповідей на них.
9. Відповідність дизайну вимогам програмного забезпечення.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Gox Testing. Тестування цим методом передбачає наявність обмежених знань про внутрішню структуру застосунка та відбувається на рівні «системного тестування». На відповідність функціональним вимогам перевіряються як програмний код, так і безпосередньо програмний продукт.

Використовуються наступні методи тестування:

1. Функціональне тестування, зокрема на рівні інтеграційного тестування (тестування взаємодії різних модулів програми).
2. Тестування продуктивності мобільного застосунка.
3. Ручне тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Повний процес тестування проходить у такому порядку:

1. Статичне тестування коду.
2. Динамічне ручне тестування полів вводу на граничні та неможливі значення.
3. Динамічне ручне тестування на відповідність поставленим функціональним вимогам.
4. Тестування інтерфейсу за різної роздільній здатності екрану.
5. Тестування роботи на пристроях з різними версіями операційної системи Android.
6. Тестування при конкурентному навантаженні на систему з боку декількох користувачів.
7. Тестування стабільності роботи за різних зовнішніх умов, зокрема при відсутності підключення до мережі Інтернет.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ СИСТЕМАТИЗАЦІЇ ІНФОРМАЦІЇ ПРО ПРОХОДЖЕННЯ АВТОМОБІЛЕМ ТЕХНІЧНИХ ОГЛЯДІВ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна БІЛОУС

2023

ЗМІСТ

1. СТРУКТУРА МОБІЛЬНОГО ЗАСТОСУНКА

Мобільний застосунок для систематизації інформації про проходження автомобілем технічних оглядів складається лише з динамічних сторінок.

Застосунок має світлу та темну теми інтерфейсу, яка автоматично встановлюється у відповідності до теми пристрою.

Головна мова інтерфейсу – англійська (за замовчуванням) або українська (для мобільних пристроїв, у яких системною мовою встановлена українська). Українська мова також використовується для показу назв виконаних ремонтних робіт, замінених деталей та СТО.

Сторінки застосунка: сторінка входу, сторінка входу попередньо авторизованого користувача, сторінка реєстрації, сторінка відновлення паролю, головна сторінка авторизованого користувача, сторінка детальної інформації обраного автомобіля, сторінка редагування обраного автомобіля, сторінка створення автомобіля, сторінка записів про техогляд обраного автомобіля, сторінка детальної інформації обраного запису про техогляд, сторінка редагування обраного запису про техогляд, сторінка створення запису про техогляд, сторінка контролю доступу інших користувачів до даних обраного автомобіля, сторінка редагування контролю доступу інших користувачів до даних обраного автомобіля, сторінка налаштування сповіщень обраного автомобіля, сторінка витрат обраного автомобіля, сторінка інформації про застосунок, сторінка вибору автомобіля для представлення аналітики поширених проблем автомобілів, сторінка представлення аналітики поширених проблем обраного автомобіля, сторінка вибору СТО для представлення переліку її робіт із оцінками виконання іншими користувачами, сторінка представлення переліку робіт обраної СТО із оцінками виконання іншими користувачами.

Сторінки входу, реєстрації та відновлення паролю використовують один шаблон (рис. 1): вміст сторінки розміщується у напівпрозорому контейнері, позаду якого відтворюється відео.

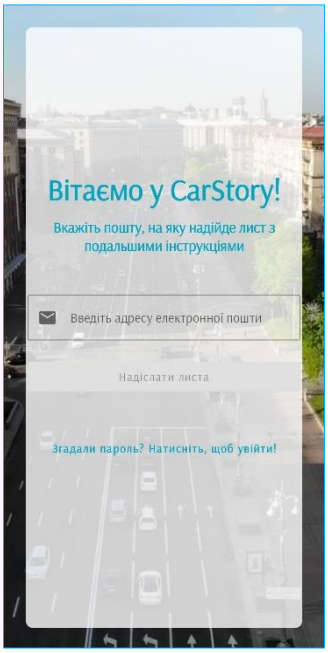


Рис. 1. Сторінка відновлення паролю

Решта сторінок використовують інший шаблон (рис. 2): вгорі розташована верхня панель із короткою інформацією про поточну сторінку, у лівій частині верхньої панелі розташований елемент, що при взаємодії відкриває виринаючу бокову панель (рис. 3), яка містить посилання на окремі сторінки та меню для виходу з облікового запису.

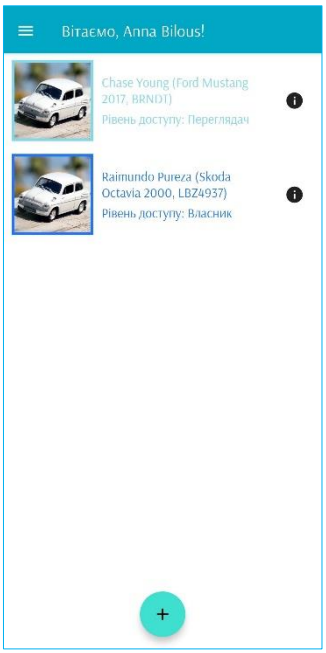


Рис. 2. Головна сторінка авторизованого користувача

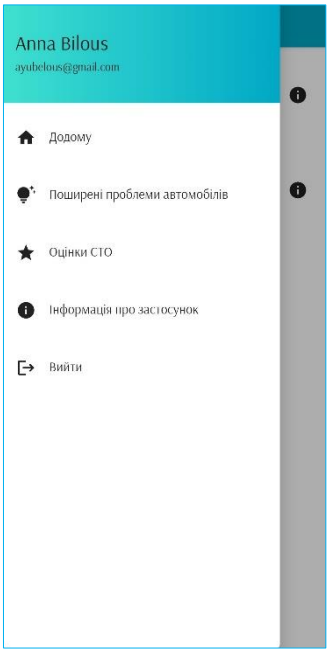


Рис. 3. Виринаюча бокова панель

Сторінка записів про техогляд обраного автомобіля, сторінка контролю доступу інших користувачів до даних обраного автомобіля, сторінка налаштування сповіщень обраного автомобіля та сторінка витрат обраного автомобіля також мають нижню панель (рис. 4), яка дозволяє переходити між цими сторінками. Нижня панель відрізняється в залежності від прав доступу до обраного автомобіля: користувачі з правами Редактора та Переглядача не мають кнопок для переходу на сторінки контролю доступу інших користувачів та налаштування сповіщень (рис. 5).

Рис. 4. Нижня панель для користувачів типу Власник

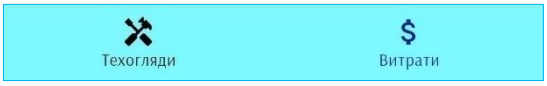


Рис. 5. Нижня панель для користувачів типу Редактор та Переглядач

2. ПРОЦЕДУРА РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Для використання можливостей мобільного застосунка користувач повинен пройти процедуру реєстрації та авторизації.

На сторінці реєстрації користувач має заповнити наступні поля: електронна пошта, ім’я користувача (не менше 5 і не більше 30 літер), пароль (не менше 6 символів), підтвердження паролю (має співпадати із введеним паролем).

Електронна пошта є унікальною для кожного користувача. У випадку співпадіння з’явиться помилка реєстрації (рис. 6).

Рис. 6. Сторінка реєстрації

Для авторизації користувач має заповнити на сторінці входу поля електронної пошти та пароля. Якщо дані не співпадають, з’явиться помилка авторизації (рис. 7).

Користувач, що попередньо авторизувався у системі та закрив застосунок, але не вийшов з облікового запису, при повторному відкритті застосунка переходить на сторінку входу попередньо авторизованого

користувача, на якій він має вказати лише пароль. Якщо пароль невірний, з’явиться помилка авторизації (рис. 8).

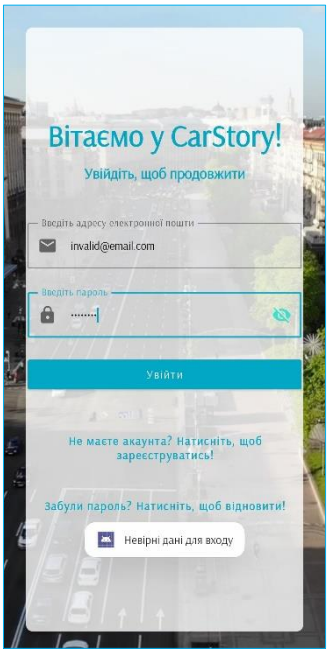


Рис. 7. Сторінка входу

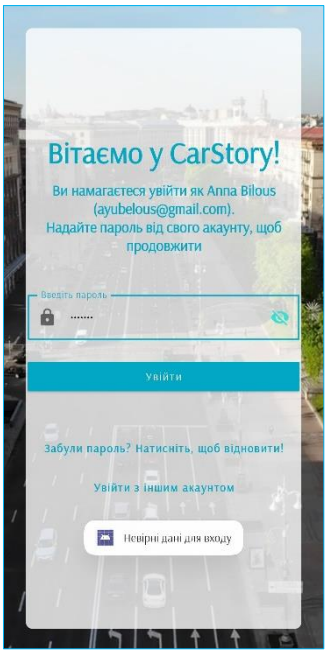


Рис. 8. Сторінка входу попередньо авторизованого користувача

3. ПРОЦЕДУРИ РОБОТИ З ДАНИМИ АВТОМОБІЛІВ

Авторизований користувач переходить на головну сторінку застосунка (рис. 2). На цій сторінці представлений перелік автомобілів, до яких користувач має доступ. Кожен елемент переліку містить коротку інформацію про автомобіль та рівень доступу користувача до даних автомобіля, з правого боку елемента розташована іконка інформації, яка при взаємодії перенаправляє користувача на сторінку детальної інформації обраного автомобіля.

У нижній частині екрану головної сторінки розташована кнопка з плюсом. При взаємодії з цією кнопкою користувач переходить на сторінку створення автомобіля. Для створення автомобіля користувач має заповнити наступні обов’язкові поля: назва автомобіля, виробник та модель, рік випуску, номерний знак, поточний пробіг, тип пального. Тип пального, виробника і модель користувач має обрати із запропонованих списків варіантів. При спробі створити автомобіль із виробником та моделлю не зі списку варіантів з’явиться помилка створення (рис. 9).

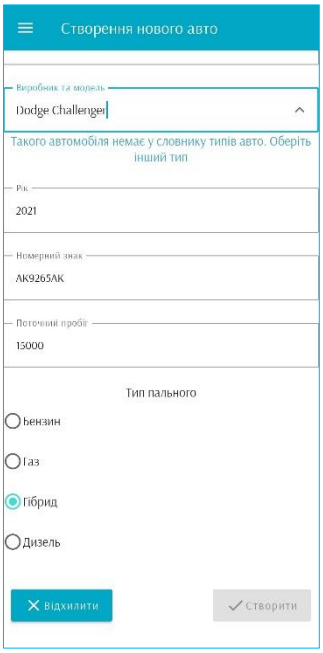


Рис. 9. Сторінка створення автомобіля

Для підтвердження створення нового автомобіля користувач має натиснути кнопку «Створити». Поточний користувач автоматично встановлюється системою як Власник нового автомобіля.

При успішному створенні автомобіля користувач автоматично перенаправляється на сторінку детальної інформації обраного автомобіля. На цій сторінці представлена така інформація про автомобіль: назва, рівень доступу користувача до даних, виробник, модель, рік випуску, номерний знак, поточний пробіг та тип пального. Для користувачів із правами Власника після детальної інформації автомобіля також відображаються іконки олівця та сміттєвого бака (рис. 10).

Рис. 10. Сторінка детальної інформації обраного автомобіля для користувачів типу Власник

При взаємодії із іконкою олівця користувач переходить на сторінку редагування обраного автомобіля (рис. 11). Користувач може редагувати таку інформацію: назва, виробник, модель, рік випуску, номерний знак та тип пального. Для збереження змін користувач має натиснути кнопку «Зберегти зміни».

Рис. 11. Сторінка редагування обраного автомобіля

При взаємодії із іконкою сміттєвого бака на сторінці детальної інформації обраного автомобіля висвічується діалогове вікно підтвердження видалення (рис. 12). Для відхилення операції видалення та закриття вікна користувач має натиснути кнопку «Ні, залишити». Для видалення автомобіля користувач має натиснути кнопку «Так, видалити».

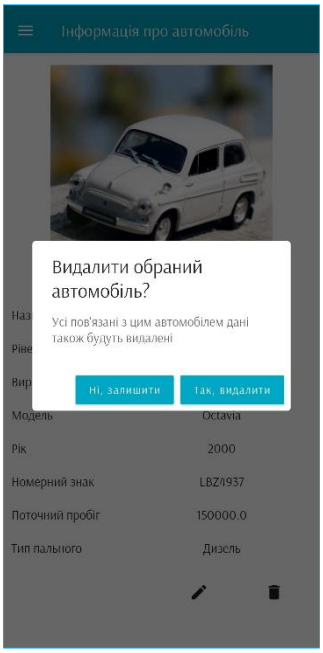


Рис. 12. Діалогове вікно підтвердження видалення обраного автомобіля

4. ПРОЦЕДУРИ РОБОТИ З ДАНИМИ ЗАПИСІВ ПРО ТЕХОГЛЯДИ

При натисненні на певний запис автомобіля на головній сторінці користувач переходить на сторінку записів про техогляд обраного автомобіля (рис. 13). На цю сторінку користувач також переходить при натисненні на меню «Техогляди» у нижній панелі.

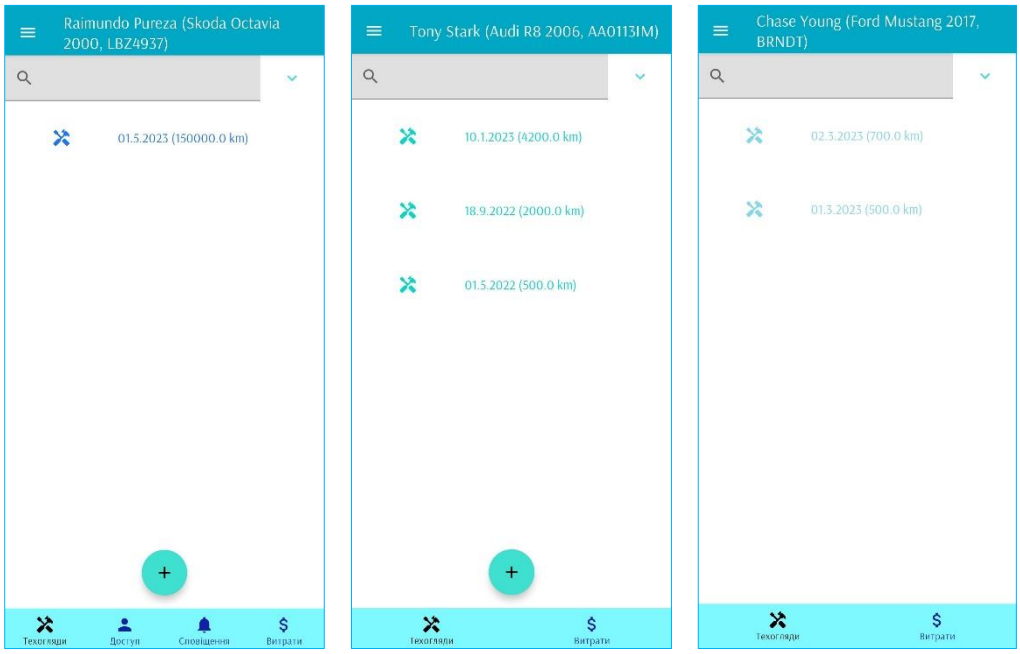


Рис. 13. Сторінка записів про техогляд обраного автомобіля для користувачів типу Власник (ліворуч), Редактор (по центру), Переглядач (праворуч)

У верхній частині екрана, під верхньою панеллю, розташоване поле пошуку та іконка стрілки вниз. При натисненні на іконку стрілки вниз відбудеться сортування записів про техогляд за пробігом у зростаючому порядку, а стрілка униз зміниться на стрілку вгору. При натисненні на іконку стрілки вгору відбудеться сортування записів про техогляд за пробігом у спадному порядку, а стрілка угору заміниться на стрілку вниз.

При зміні вводу у полі пошуку записи про техогляд автоматично фільтруються у відповідності до значення вводу. Пошук співпадінь із

введеним значенням відбувається у назвах виконаних робіт та замінених деталей, назві СТО та значенні пробігу.

Нижче представлений перелік записів про техогляд автомобіля. Для кожного запису відображається іконка інструментів та коротка інформація про техогляд: дата проведення та пробіг автомобіля на момент техогляду. При взаємодії із окремим елементом списку користувач переходить на детальної інформації обраного запису про техогляд.

У нижній частині екрану для користувачів з правами доступу Власник або Редактор розташована кнопка з плюсом. При взаємодії з цією кнопкою користувач переходить на сторінку створення запису про техогляд (рис. 14).

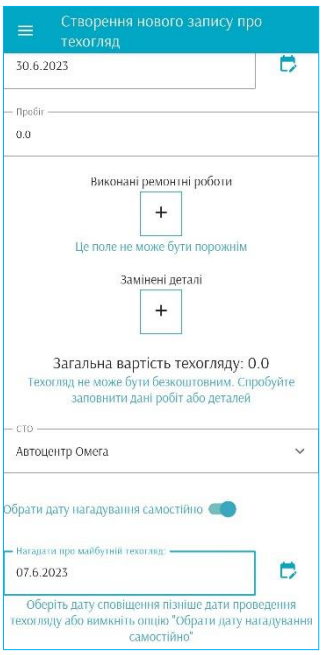


Рис. 14. Сторінка створення запису про техогляд

Для створення запису про техогляд користувач має заповнити наступні обов’язкові поля: дата проведення техогляду, пробіг автомобіля на момент техогляду, виконані ремонтні роботи (вказати хоча б одну роботу). Загальна вартість робіт, деталей та усього техогляду вираховується автоматично. За замовчуванням дата нагадування також вираховується автоматично, але користувач може встановити її самостійно: для цього необхідно увімкнути перемикач «Обрати дату нагадування самостійно». У

такому випадку дата нагадування повинна бути встановлена пізніше дати проведення техогляду. Якщо перемикач «Обрати дату нагадування самостійно» увімкнений, а дата нагадування є меншою або дорівнює даті проведення техогляду, з’явиться помилка створення.

Користувач може опціонально вказати перелік замінених деталей та назву СТО, що виконувала техогляд. Для кожної ремонтної роботи та заміненої деталі користувач має заповнити такі поля: назва зі списку, вартість виконання роботи або ціна деталі, тривалість виконання роботи або кількість використаних деталей. Для кожної ремонтної роботи користувач може опціонально вказати оцінку виконання за допомогою іконок зірочок: кількість зафарбованих зірочок відповідає оцінці за шкалою від 1 до 5.

Назва ремонтних робіт, замінених деталей та СТО користувач має обрати із запропонованих списків варіантів. При спробі створити запис про техогляд із значеннями не зі списку варіантів з’явиться помилка створення.

При успішному створенні запису про техогляд користувач автоматично перенаправляється на сторінку детальної інформації обраного запису про техогляд (рис. 15).

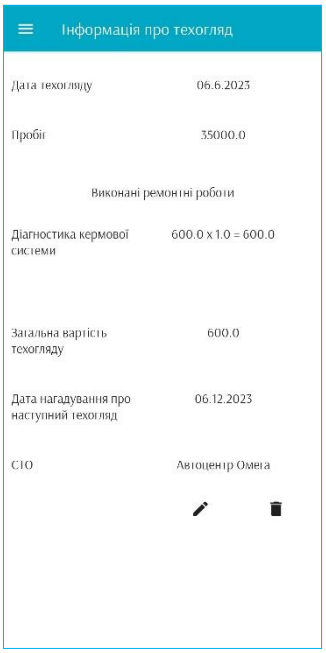


Рис. 15. Сторінка детальної інформації обраного запису про техогляд для користувачів типу Власник

На цій сторінці представлена така інформація про запис про техогляд: дата проведення техогляду, пробіг автомобіля на момент техогляду, перелік виконаних ремонтних робіт та їх детальної інформації, перелік замінених деталей та їх детальної інформації (за наявності), загальна вартість техогляду, дата нагадування про наступний техогляд, назва СТО (за наявності). Для користувачів із правами Власника або Редактора після детальної інформації запису також відображаються іконки олівця та сміттєвого бака.

При взаємодії із іконкою олівця користувач переходить на сторінку редагування обраного запису про техогляд (рис. 16). Користувач може редагувати таку інформацію: дата проведення техогляду, пробіг автомобіля на момент техогляду, виконані ремонтні роботи, замінені деталі, назва СТО. При редагуванні дати техогляду система автоматично посуне дату сповіщення на ту саму кількість днів, на яку змінилася дата техогляду.

Для відхилення операції редагування та повернення на сторінку записів про техогляд обраного автомобіля користувач має натиснути кнопку «Відхилити». Для збереження змін користувач має натиснути кнопку «Зберегти зміни».

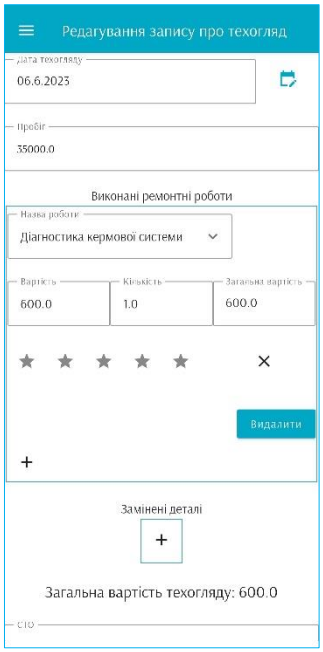


Рис. 16. Сторінка редагування обраного запису про техогляд

При взаємодії із іконкою сміттєвого бака на сторінці детальної інформації обраного запису про техогляд висвічується діалогове вікно підтвердження видалення (рис. 17).

Для відхилення операції видалення та закриття вікна користувач має натиснути кнопку «Ні, залишити». Для видалення запису користувач має натиснути кнопку «Так, видалити».

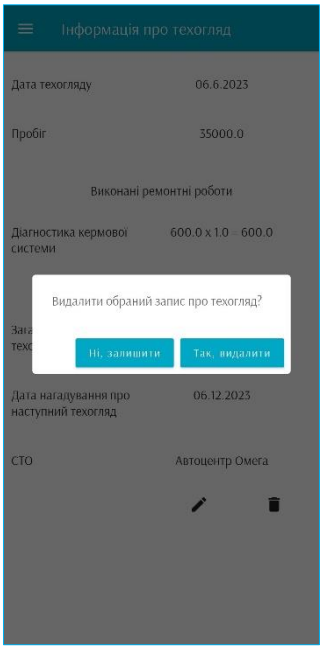


Рис. 17. Діалогове вікно підтвердження видалення обраного запису про техогляд

При натисненні на меню «Витрати» у нижній панелі користувач переходить на сторінку витрат обраного автомобіля (рис. 18). На початку сторінки розташовані дві кнопки, за допомогою яких користувач може обирати період, за який відображати дані: ліва кнопка встановлює дату початку періоду, права кнопка встановлює дату кінця періоду. Техогляди, проведені у обрані дати, також включаються до статистики витрат.

Нижче представлені кругові діаграми, які відображають у спадному порядку витрати за вказаний період: загальна вартість кожного техогляду, вартість ремонтних робіт, вартість замінених деталей (за наявності даних).

Під кожною діаграмою є легенда, що представляє перелік компонент діаграми, їх загальну вартість та їх відсотковий вклад у загальну суму.

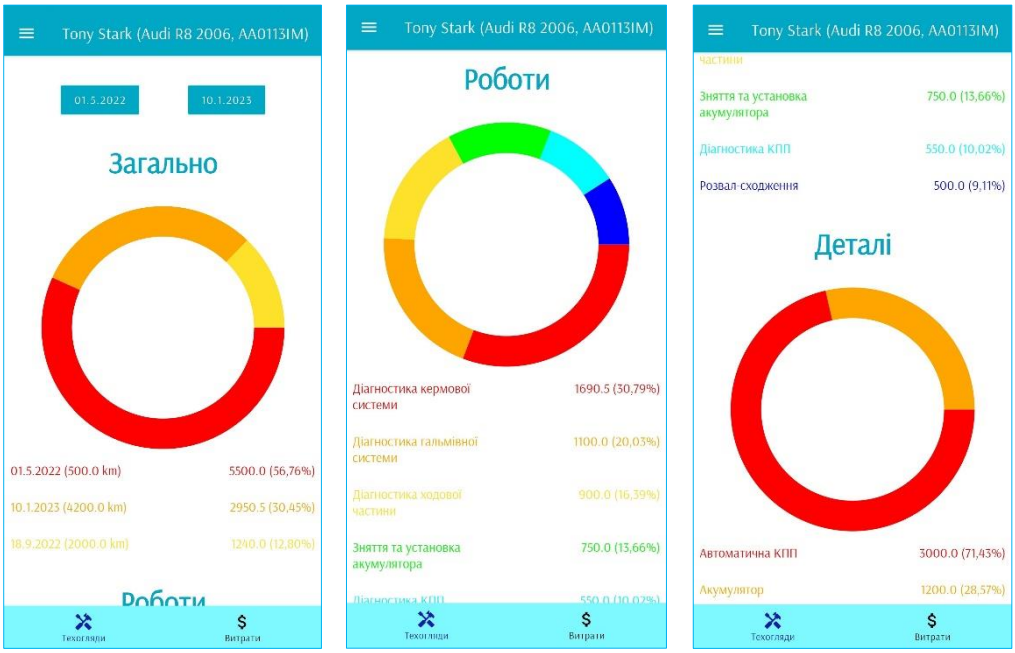


Рис. 18. Сторінка витрат обраного автомобіля

5. ПРОЦЕДУРА СПОВІЩЕННЯ ПРО МАЙБУТНІ ТЕХОГЛЯДИ

При натисненні на меню «Сповіщення» у нижній панелі користувач переходить на сторінку налаштування сповіщень обраного автомобіля (рис. 19). На сторінці представлений перелік усіх сповіщень, які мають дату нагадування у майбутньому або ще не є вирішеними (користувач не позначив, що не хоче отримувати певне сповіщення). Кожен елемент переліку містить іконку дзвіночка (перекреслений дзвіночок означає, що сповіщення позначене вирішеним та не буде надходити), коротку інформацію про сповіщення (зокрема дату проведення техогляду, пробіг автомобіля на момент проходження техогляду та дату сповіщення), іконку календаря з олівцем та іконку годинника (перекреслений годинник або годинник зі знаком плюс на циферблаті).

При натисненні на іконку календаря з олівцем користувач може змінити дату нагадування за допомогою діалогового вікна із календарем. Календар автоматично налаштований так, що користувач не зможе обрати дату нагадування раніше дати проведення техогляду або у той же день. Для скасування редагування користувач має натиснути кнопку «Скасувати».

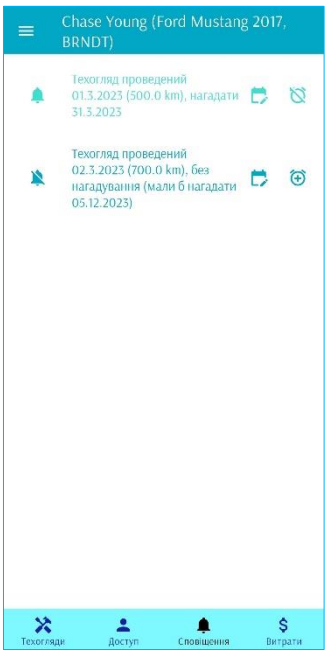


Рис. 19. Сторінка налаштування сповіщень обраного автомобіля

При натисненні на іконку годинника зі знаком плюс на циферблаті іконка заміниться на іконку перекресленого годинника, а сповіщення буде позначене як невирішене (увімкнене): користувачу прийде сповіщення про техогляд у дату сповіщення. При натисненні на іконку перекресленого годинника іконка заміниться на іконку зі знаком плюс на циферблаті, а сповіщення буде позначене як вирішене (вимкнене): користувачу не прийде сповіщення про техогляд у дату сповіщення.

Сповіщення приходять один раз на добу на електронну пошту користувача для усіх автомобілів, до яких користувач має будь-який доступ.

6. ПРОЦЕДУРА КОНТРОЛЮ ДОСТУПУ ДО ДАНИХ АВТОМОБІЛІВ

При натисненні на меню «Доступ» у нижній панелі користувач переходить на сторінку контролю доступу інших користувачів до даних обраного автомобіля (рис. 20). На сторінці представлений перелік імен та електронних адрес усіх користувачів, що мають доступ до обраного автомобіля, який розділений на три групи за правами доступу користувачів. Якщо обраний автомобіль не має користувачів із правами доступу Редактора або Переглядача, відповідні групи відображатися не будуть.

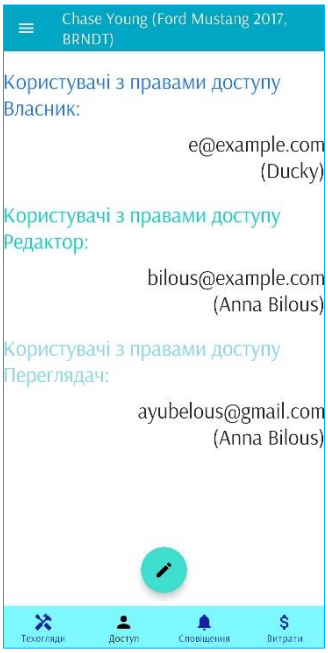


Рис. 20. Сторінка контролю доступу інших користувачів до даних обраного автомобіля

У нижній частині екрану розташована кнопка з олівцем. При взаємодії з цією кнопкою користувач переходить на сторінку редагування контролю доступу інших користувачів до даних обраного автомобіля (рис. 21).

Користувач може додати, змінити або видалити користувачів з правами доступу Редактора або Переглядача шляхом редагування

електронної пошти користувача. При спробі надати права доступу неіснуючому користувачу з’явиться помилка оновлення.

Користувач також може передати права Власника іншому користувачу, вказавши пошту нового Власника у відповідному полі вводу. У такому випадку для збереження змін користувач також має ознайомитися із попередженням унизу сторінки про наслідки зміни Власника та позначити прапорець «Я розумію наслідки та хочу змінити Власника».

Для відхилення операції редагування та повернення на головну сторінку користувач має натиснути кнопку «Відхилити». Для збереження змін користувач має натиснути кнопку «Зберегти зміни».

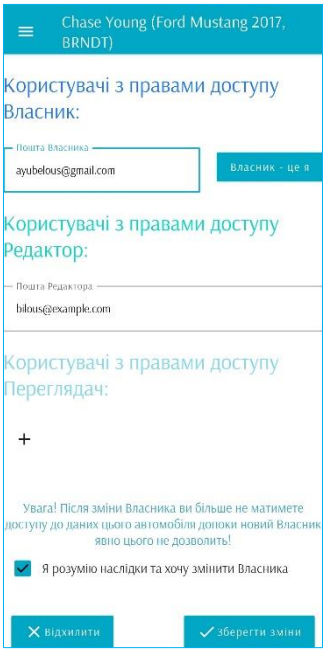


Рис. 21. Сторінка редагування контролю доступу інших користувачів до даних обраного автомобіля

7. ПРОЦЕДУРИ ПЕРЕГЛЯДУ ЗНЕОСОБЛЕНОЇ АНАЛІТИКИ

7.1. Аналітика поширених проблем автомобілів

При натисненні на меню «Поширені проблеми автомобілів» у виринаючому боковому меню користувач переходить на сторінку вибору автомобіля для представлення аналітики поширених проблем автомобілів (рис. 22). У верхній частині екрана, під верхньою панеллю, розташоване поле пошуку. При зміні вводу у полі пошуку перелік типів автомобілів фільтрується у відповідності до значення вводу. Пошук співпадінь із введеним значенням відбувається у будь-якому слові типу автомобіля.

Нижче представлений перелік типів автомобілів, що складається з об’єднаних у один рядок виробника та моделі. При взаємодії із окремим елементом списку користувач переходить на сторінку представлення аналітики поширених проблем обраного автомобіля (рис. 23).

Рис. 22. Сторінка вибору автомобіля для представлення аналітики поширених проблем автомобілів

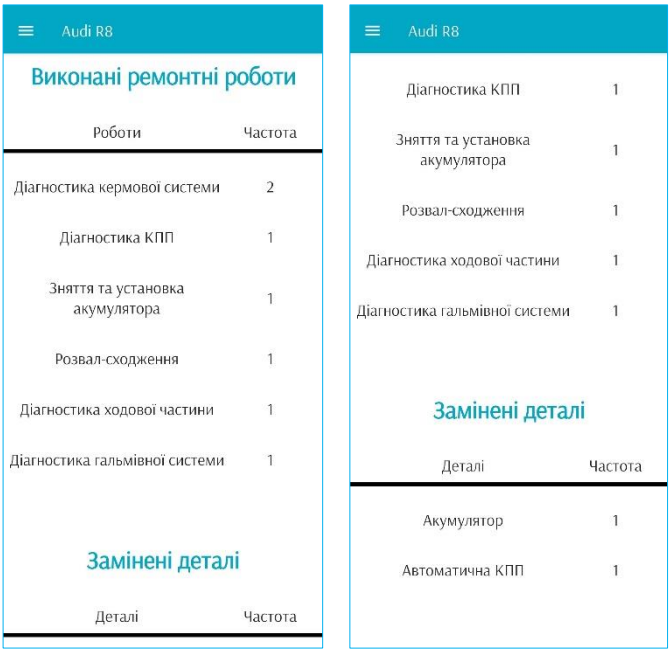


Рис. 23. Сторінка представлення аналітики поширених проблем обраного автомобіля

На сторінці представлення аналітики поширених проблем обраного автомобіля представлена таблиця, що містить перелік з 10 робіт, що найчастіше виконувалися для обраного автомобіля, та частот виконання цих робіт. Роботи у таблиці відсортовані за частотою у порядку спадання. За наявності даних про замінені деталі нижче буде представлена аналогічна таблиця для 10 найчастіше замінених деталей.

7.2. Аналітика робіт СТО із оцінками виконання

При натисненні на меню «Оцінки СТО» у виринаючому боковому меню користувач переходить на сторінку вибору СТО для представлення переліку її робіт із оцінками виконання іншими користувачами (рис. 24). У верхній частині екрана, під верхньою панеллю, розташоване поле пошуку. При зміні вводу у полі пошуку перелік назв СТО фільтрується у відповідності до значення вводу. Пошук співпадінь із введеним значенням відбувається у будь-якому слові назви СТО. Під полем пошуку представлений перелік назв СТО.

Рис. 24. Сторінка вибору СТО для представлення переліку її робіт із оцінками виконання іншими користувачами

При взаємодії із окремим елементом списку користувач переходить на сторінку представлення переліку робіт обраної СТО із оцінками виконання іншими користувачами (рис. 25).

Рис. 25. Сторінка представлення переліку робіт обраної СТО із оцінками виконання іншими користувачами

На цій сторінці представлена таблиця, що містить перелік робіт, що виконувалися обраною СТО, та середні оцінки виконання цих робіт. Роботи у таблиці відсортовані за середньою оцінкою у порядку спадання. Під цією таблицею розташований перелік виконаних робіт, що не були оцінені жодним іншим користувачем.