Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Цифрова сервісна книжка для автомобіля

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-6
Максим ОЛЕФІР
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Ольга ЄРОШЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Олефіру Максиму Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Цифрова сервісна книжка для автомобіля

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи платформа для розробки: Android;
фреймворк для побудови інтерфейсу: XML;
архітектурний шаблон: MVVM;
місце зберігання даних: локальна база даних (Room);
основні функціональні вимоги: ведення історії технічного обслуговування, створення профілю автомобіля, перегляд записів, нагадування про ТО.

4. Перелік питань, що потрібно опрацювати у роботі
аналіз існуючих рішень у сфері цифрових сервісних книжок автомобіля;
вибір та обґрунтування платформи, мови програмування і програмних засобів для реалізації;
проєктування архітектури мобільного застосунку та моделі даних;
програмна реалізація Android-застосунку з використанням Kotlin та Jetpack Compose;
тестування програмного продукту та оцінка його функціональних можливостей;
висновки та перспективи подальшого розвитку цифрової сервісної книжки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Вибір теми, узгодження з науковим керівником	27.05.2025 – 28.05.2025	
2	Збір та аналіз літературних джерел за темою	29.05.2025 – 30.05.2025	
3	Підготовка оглядово-аналітичного розділу	31.05.2025 – 01.06.2025	
4	Постановка задачі та формалізація вимог до ПЗ	02.06.2025 – 03.06.2025	
5	Проєктування архітектури застосунку та бази даних	04.06.2025 – 05.06.2025	
6	Реалізація програмного засобу	06.06.2025 – 10.06.2025	
7	Тестування, налагодження, внесення коригувань	11.06.2025 – 12.06.2025	
8	Оформлення пояснювальної записки	13.06.2025 – 14.06.2025	
9	Перевірка на плагіат, рецензування, попередній захист	15.06.2025	
10	Захист кваліфікаційної роботи	16.06.2025	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Ольга ЄРОШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 63 с., 20 рис., 4 табл., 1 дод., 9 джерел.

АВТОМОБІЛЬ, СЕРВІСНА КНИЖКА, МОБІЛЬНИЙ ДОДАТОК, VIN-КОД, ПРОБІГ, ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ, НАГАДУВАННЯ, ЗАПИС СЕРВІСУ, КОРИСТУВАЧ, МАЙСТЕР.

Метою дослідження є розробка зручного, функціонального та надійного інструменту, який дозволяє водіям та автосервісам фіксувати, зберігати та переглядати історію технічного обслуговування транспортного засобу в цифровому форматі.

У процесі виконання роботи було проаналізовано наявні рішення на ринку, розглянуто технологічні платформи та засоби реалізації, обґрунтовано вибір Android як цільової платформи, мови програмування Kotlin та класичного підходу до побудови інтерфейсу користувача на основі XML-розмітки. Архітектура застосунку побудована відповідно до шаблону MVVM (Model–View–ViewModel), який забезпечує розподіл відповідальностей між компонентами системи, полегшує масштабування й тестування коду, а також спрощує розробку та підтримку в майбутньому. Було реалізовано локальну базу даних на основі бібліотеки Room, що дозволяє ефективно зберігати й обробляти інформацію на пристрої без необхідності постійного підключення до інтернету.

ABSTRACT

Bachelor’s thesis: 63 pages, 20 figures, 4 tables, 1 appendices, 9 sources.

CAR · SERVICE BOOK · MOBILE APPLICATION · VIN CODE · MILEAGE · MAINTENANCE · REMINDER · SERVICE RECORD · USER · TECHNICIAN.

The goal of the research is to create a user-friendly, functional, and reliable tool that allows drivers and auto services to record, store, and view the maintenance history of a vehicle in digital format.

During the development process, existing market solutions were analyzed, technological platforms and implementation tools were reviewed, and the choice of Android as the target platform was justified, along with Kotlin as the programming language and the classic XML-based user interface approach. The application architecture is built using the MVVM (Model–View–ViewModel) pattern, which enables separation of concerns, facilitates scalability and testing, and simplifies development and future maintenance. A local database using the Room library was implemented to efficiently store and manage data on the device without requiring a constant internet connection.

ЗМІСТ

ВСТУП 8
1 СЕРВІСНА КНИЖКА ЯК ЦИФРОВИЙ ЗАСТОСУНОК: ПРИЗНАЧЕННЯ, МОЖЛИВОСТІ ТА АНАЛОГИ 10
1.1 Призначення цифрової сервісної книжки 10
1.2 Основні функції та сценарії використання. 11
1.3 Аналоги та огляд сучасних цифрових рішень 12
1.4 Переваги запропонованого рішення 12
2 СТРУКТУРА ПРОЄКТУ ANDROID (CARSERVICEAPP) 14
2.1 Загальна структура файлів 14
2.2 Kotlin-класи (пакет com.example.carserv) 16
2.3 Графічні ресурси та меню (drawable, menu, mipmap) 18
2.4 Ресурси для перекладу та стилів (values, values-uk, themes) 20
3 ОСНОВНІ ФУНКЦІОНАЛЬНІ КОМПОНЕНТИ 25
3.1 Авторизація та реєстрація користувача 25
3.2 Додавання та перегляд автомобілів 27
3.3 Сервісна книжка автомобіля 29
3.4 Створення та перегляд нагадувань 32
3.5 Перемикач мови інтерфейсу та локальні сповіщення 34
3.6 Збереження моделей (Car, ServiceRecord, Reminder) (NotificationHelper) та сервісні регламенти (ServiceScheduleProvider) та сервісні регламенти 38
4 ІНТЕРФЕЙС КОРИСТУВАЧА 44
4.1 Структура XML-макетів активностей 44
4.2 Адаптери списків (RecyclerView Adapters) 46
4.3 Стилі, кольори та переклади 48
ВИСНОВКИ 52
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 54

ДОДАТОК А Графічний матеріал кваліфікаційної роботи..... 55

ВСТУП

У сучасному світі автомобіль став невід'ємною частиною повсякденного життя, а його технічне обслуговування – важливим чинником безпеки та ефективності експлуатації. Ведення історії обслуговування традиційно здійснюється у вигляді паперових сервісних книжок, які мають низку недоліків: висока ймовірність втрати, фальсифікації записів, складність доступу до інформації та обмежена інтеграція з іншими системами. Світові тенденції розвитку інформаційних технологій демонструють перехід до цифрових рішень, які підвищують надійність зберігання даних, забезпечують зручність користування і можливість оперативного обміну інформацією між учасниками процесу обслуговування [1].

Актуальність дослідження полягає в необхідності створення надійного, функціонального та захищеного цифрового програмного забезпечення, яке б дозволяло ефективно вести облік технічного обслуговування автомобілів. Така система має відповідати вимогам сучасного ринку, бути масштабованою, адаптивною до різних типів користувачів і пристроїв, а також відповідати вимогам до захисту персональних даних.

Існує значна потреба в цифрових рішеннях для ведення сервісної документації в автосервісах, страхових компаніях, серед приватних автовласників та державних структур. Перспектива автоматизації таких процесів відкриває можливості для зменшення бюрократії, підвищення прозорості та зручності доступу до інформації [2].

Мета роботи. Розробка програмного засобу для ведення цифрової сервісної книжки автомобіля, що забезпечує створення, зберігання, перегляд та редагування даних про технічне обслуговування транспортного засобу із забезпеченням зручного інтерфейсу та доступу через інтернет.

Результати дипломної роботи можуть бути застосовані в автосервісних підприємствах, компаніях з обслуговування автопарків, страховому секторі, а

також серед приватних користувачів, які бажають зберігати повну історію обслуговування свого автомобіля в електронному вигляді.

1 СЕРВІСНА КНИЖКА ЯК ЦИФРОВИЙ ЗАСТОСУНОК: ПРИЗНАЧЕННЯ, МОЖЛИВОСТІ ТА АНАЛОГИ

1.1 Призначення цифрової сервісної книжки

У період стрімкого розвитку інформаційних технологій зростає потреба в цифрових рішеннях, що автоматизують повсякденні та виробничі процеси. В умовах активного використання автотранспорту, як у приватному, так і у корпоративному секторі, особливої актуальності набуває задача ефективного управління технічним обслуговуванням транспортних засобів. Одним із найперспективніших інструментів у цій сфері є цифрова сервісна книжка – мобільний застосунок, який дозволяє користувачеві вести повноцінний електронний облік технічного стану авто [3].

На відміну від застарілих паперових документів, цифрова сервісна книжка поєднує в собі переваги мобільності, інтерактивності та безпечного зберігання даних, що значно підвищує якість управління експлуатацією автомобіля.

Цифрова сервісна книжка – це приклад предметно-орієнтованого мобільного застосунку, розробленого для зручного та структурованого документування даних про технічне обслуговування транспортного засобу. Основними функціональними завданнями такого програмного рішення є:

Формування електронної історії обслуговування: усі дії фіксуються у формі впорядкованих записів із зазначенням ключових параметрів: дати, пробігу, типу виконаних робіт, коментарів користувача тощо.

Підвищення прозорості технічного стану авто: цифровий запис усуває ризики фальсифікації даних, втрати документа або несанкціонованого внесення змін.

Оптимізація взаємодії з сервісними центрами: можливість експорту даних, синхронізація з майстром чи страховою компанією через інтерфейс

застосунку.

Персоналізоване планування обслуговування: функція автоматичних нагадувань дозволяє своєчасно інформувати користувача про наближення регламентного обслуговування.

Таким чином, сервісна книжка у вигляді мобільного застосунку виконує роль цифрового асистента, що забезпечує повноцінну інформаційну підтримку процесу експлуатації транспортного засобу [4].

1.2 Основні функції та сценарії використання.

Цифрова сервісна книжка, як сучасне програмне забезпечення, реалізує широкий спектр функцій, що охоплюють як базові, так і додаткові сценарії користування. Зокрема:

- реєстрація транспортного засобу: користувач вносить технічні характеристики (марка, модель, рік, VIN-код, поточний пробіг), які в подальшому використовуються для ідентифікації та обробки даних;

- ведення журналу обслуговування: кожна сервісна операція зберігається у вигляді структурованого запису, що дозволяє переглядати історію у хронологічному порядку або за категоріями;

- створення нагадувань: застосунок дозволяє формувати персоналізовані або типові інтервали ТО на основі пробігу чи часу, з автоматичним генеруванням локальних сповіщень;

- можливість фільтрації та аналітики: реалізація фільтрації записів за параметрами (тип обслуговування, дата, пробіг) та потенційне виведення статистичних звітів;

- підтримка багатомовного інтерфейсу: завдяки використанню локалізованих ресурсів користувач може обирати мову взаємодії відповідно до своїх уподобань;

- офлайн-доступ: збереження усіх функцій навіть за відсутності інтернет-з'єднання — важлива перевага для користувачів у польових умовах.

- застосунок орієнтований як на приватних автовласників, так і на комерційних користувачів (автопарки, СТО, страхові компанії), що робить його універсальним інструментом для ринку автосервісу [5].

1.3 Аналоги та огляд сучасних цифрових рішень

На ринку інформаційних технологій існує низка рішень, що частково або повністю реалізують функціонал електронної сервісної книжки:

- Carfax Car Care (США) – застосунок із можливістю автоматичного завантаження історії ТО за VIN-кодом з національної бази США;
- Drivvo – мобільна платформа для контролю витрат на автомобіль, включно з обслуговуванням, заправками та страховками;
- My Car – персональний цифровий журнал автомобіліста з підтримкою декількох авто;
- Fleet Complete, AUTOSOFT CRM – корпоративні рішення для обліку обслуговування у автопарках або СТО.

Однак, незважаючи на функціональну насиченість, більшість із зазначених продуктів:

- не адаптовані до україномовної аудиторії;
- вимагають платної підписки для повного доступу до функцій;
- не враховують специфіки вітчизняного сервісного ринку;
- не підтримують гнучку кастомізацію регламентів ТО.

Ці фактори визначають актуальність створення власного застосунку, який буде повністю орієнтований на потреби локального користувача та ринок України [6].

1.4 Переваги запропонованого рішення

На відміну від типових комерційних продуктів, розроблений у межах проєкту застосунок має такі конкурентні переваги:

- локалізація інтерфейсу – повна підтримка української та англійської мов з можливістю динамічного перемикання;
- гнучкість конфігурації – можливість вручну створювати категорії обслуговування, регламентні інтервали, нагадування;
- автономність – повноцінна робота без підключення до інтернету;
- локальне зберігання даних – із використанням Room або SharedPreferences, з перспективою синхронізації через Firebase;
- простота інтерфейсу – інтуїтивна логіка навігації та оформлення згідно з рекомендаціями Material Design;
- масштабованість – архітектура застосунку дозволяє в подальшому додати роль "майстра", облікові записи, push-сповіщення, хмарне зберігання, веб-кабінет тощо.

Запропонований цифровий застосунок реалізує повний життєвий цикл ведення сервісної історії автомобіля, забезпечуючи прозорість, ефективність і зручність використання. Він є не лише функціональним інструментом для користувача, а й має потенціал інтеграції в ширшу екосистему цифрового автосервісу.

2 СТРУКТУРА ПРОЄКТУ ANDROID (CARSERVICEAPP)

2.1 Загальна структура файлів

Мобільний додаток Car Service App розроблений з використанням Android SDK, мови програмування Kotlin та архітектури, заснованої на Activity-based підході, що передбачає чіткий поділ інтерфейсів за окремими екранами (активностями). Основна увага при проєктуванні була приділена підтримці масштабованості, модульності та локалізації застосунку.

Проєкт реалізовано у середовищі Android Studio та організовано за типовою ієрархією Android-додатку, що забезпечує зручність навігації, розширення функціоналу та супровід проєкту.

Структура каталогу має такий вигляд: /app/src/main/java/com.example.carserviceapp/- основна директорія, в якій зосереджено всі Kotlin-класи, що реалізують бізнес-логіку, інтерфейсну взаємодію та допоміжні утиліти. Вона включає:

- класи активностей (Activity), які відповідають за кожен окремий екран додатку;
- моделі даних (Car, ServiceRecord, Reminder);
- адаптери для списків (RecyclerView Adapter);
- утиліти (наприклад, LocaleHelper, NotificationHelper, ServiceScheduleProvider).

/res/layout/- Каталог XML-файлів, які описують графічне компонування інтерфейсів для кожної активності. Сюди входять файли, такі як: activity_login.xml, activity_add_car.xml, activity_service_log.xml, що визначають структуру UI-елементів: кнопок, полів введення, заголовків, списків тощо.

/res/drawable/- Містить графічні ресурси додатку. До них належать:

- векторні іконки (наприклад, зображення автомобіля, кнопки

додавання чи видалення);

- фонові елементи;
- графічні стилі для кнопок або елементів списку.

Ресурси формату .xml використовуються як scalable vector graphics (SVG) для забезпечення якісного відображення на екранах з різною роздільною здатністю.

/res/menu/ У цьому каталозі зберігаються XML-файли, що описують меню, зокрема:

- файл menu_language.xml, який реалізує спливаюче меню для перемикання мови та доступу до довідкової інформації;
- можливість динамічного керування пунктами меню в залежності від активності або ролі користувача.
- /res/values/ та /res/values-uk/ Каталоги ресурсів для підтримки багатомовності інтерфейсу. Файли strings.xml містять локалізовані рядки:
- values/strings.xml – основна мова інтерфейсу (англійська);
- values-uk/strings.xml – український переклад усіх текстових елементів.

Також у цих директоріях зберігаються файли стилів (styles.xml), кольорів (colors.xml), розмірів (dimens.xml) та тем (themes.xml), що використовуються для уніфікованого оформлення застосунку згідно з вимогами Material Design.

AndroidManifest.xml Головний конфігураційний файл Android-додатку. Він містить декларації всіх активностей, які використовуються у застосунку, а також необхідні дозволи. Наприклад:

- дозвіл на надсилання сповіщень: <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>;
- визначення стартової активності (LauncherActivity);
- позначення ролі кожної активності (exported, intent-filters, label, theme тощо).
- build.gradle.kts Файл збірки, написаний з використанням Kotlin DSL,

який:

- описує застосовані плагіни (kotlin("android"), com.android.application);
- вказує SDK, версії додатку, мінімальний API-рівень;
- перелік залежностей: AndroidX-бібліотеки, Material Components, RecyclerView, Room тощо.

Цей файл є критично важливим для успішної компіляції, запуску додатку та підключення зовнішніх бібліотек.Таким чином, структура проєкту є чітко регламентованою, відповідає найкращим практикам розробки Android-застосунків та забезпечує легкість масштабування функціоналу, тестування, локалізації та підтримки у майбутньому [7].

2.2 Kotlin-класи (пакет com.example.carserv)

Весь функціонал мобільного застосунку Car Service App реалізовано за допомогою мови програмування Kotlin у межах пакету com.example.carserviceapp. Архітектурно класи поділено на логічні групи відповідно до їх призначення: активності, моделі даних, адаптери та допоміжні утиліти. Така структура забезпечує модульність, полегшує тестування та супровід коду [8].

1. Активності (UI-екрани).Ці класи реалізують графічний інтерфейс та обробку подій користувача. Кожна активність відповідає окремому екрану в додатку.

- LauncherActivity – стартовий екран, де користувач обирає свою роль (користувач або майстер), а також переходить до входу або реєстрації.
- LoginActivity, RegisterActivity – екрани для автентифікації та створення облікового запису з валідацією електронної пошти та паролю.
- MainActivity – основна панель користувача, що відображає список доданих автомобілів.
- AddCarActivity – форма додавання нового автомобіля, з обов’язковим введенням марки, моделі, року випуску, VIN-коду та пробігу.

- ServiceLogActivity – інтерфейс перегляду сервісної історії для вибраного автомобіля.

- AddServiceActivity – активність для введення нового запису обслуговування (дата, тип робіт, пробіг, коментар).

- EditServiceRecordActivity – дозволяє змінювати наявний сервісний запис (наприклад, у випадку помилкового введення).

- ReminderActivity, RemindersActivity – інтерфейси для створення індивідуальних нагадувань та перегляду типових сервісних регламентів.

- MasterDashboardActivity – спеціальний інтерфейс для майстрів, який дозволяє переглядати автомобілі клієнтів за VIN-кодами.

- HelpActivity – довідковий екран, що містить інструкцію користувача та опис функціоналу обома мовами (українською та англійською).

2. Моделі даних. Ці класи визначають структуру об’єктів, які використовуються у додатку. Вони є основою для зберігання та передачі інформації між активностями:

- Car – модель транспортного засобу. Поля: марка, модель, рік, VIN-код, пробіг.

- ServiceRecord – запис про технічне обслуговування. Поля: дата, тип робіт, коментар, пробіг.

- Reminder – об’єкт нагадування. Поля: опис, рекомендований пробіг або дата, статус.

Усі моделі реалізовано як data class, що забезпечує зручне серіалізоване зберігання та передачу.

3. Адаптери для RecyclerView. Для виведення динамічних списків (автомобілів, сервісних записів, нагадувань) використовується компонент RecyclerView, що вимагає створення адаптерів:

- CarAdapter – відповідає за виведення списку автомобілів у MainActivity.

- ServiceRecordAdapter – забезпечує відображення сервісної історії в ServiceLogActivity.

- ReminderAdapter – використовується для відображення рекомендацій з обслуговування у RemindersActivity.

- MasterCarAdapter – адаптер для відображення переліку автомобілів у інтерфейсі майстра (MasterDashboardActivity).

Кожен адаптер реалізує власний ViewHolder і забезпечує прив’язку даних до елементів UI та обробку подій (натискання, видалення тощо).

4. Допоміжні утиліти.До складу додатку входить низка утилітних класів, що реалізують допоміжну логіку, не пов’язану безпосередньо з інтерфейсом:

- LocaleHelper, LanguageUtil – забезпечують зміну мови інтерфейсу без перезапуску додатку. Реалізовано підтримку англійської та української мов.

- ServiceScheduleProvider – містить типовий регламент обслуговування автомобілів (наприклад, заміна масла кожні 10 000 км).

- ReminderRepository – відповідає за збереження, отримання та оновлення списку нагадувань.

- NotificationHelper – створює та надсилає локальні сповіщення у разі наближення терміну планового обслуговування.

Архітектура проєкту базується на чіткому розділенні відповідальностей між класами, що відповідає принципам чистої архітектури (Separation of Concerns). Така організація дозволяє легко масштабувати застосунок, модифікувати окремі компоненти без впливу на інші та забезпечує високу якість супроводу програмного коду.

2.3 Графічні ресурси та меню (drawable, menu, mipmap)

У застосунку Car Service App візуальні елементи інтерфейсу реалізовано за допомогою стандартної системи ресурсів Android, що забезпечує підтримку адаптивного дизайну для пристроїв із різною роздільною здатністю. Графічні ресурси зберігаються у відповідних підкаталогах директорії res/, зокрема: drawable/, menu/, mipmap/.

Каталог res/drawable/ містить графічні ресурси, які використовуються в інтерфейсі додатку. В основному, це векторні зображення у форматі XML (VectorDrawable), що дозволяє зберігати високу якість при масштабуванні незалежно від розміру екрана пристрою.

Основні ресурси:

- baseline_add.xml – іконка кнопки додавання нового елемента (авто або сервісного запису);
- baseline_delete.xml – іконка для видалення об'єктів зі списку;
- ic_car_placeholder.xml – векторне зображення машини, що використовується як стандартне авто (у випадку відсутності фото);
- baseline_edit.xml, baseline_language.xml – допоміжні іконки для меню або дій користувача.

Призначення ресурсів:

- створення візуального відгуку на дії користувача (додавання, редагування, видалення);
- візуалізація елементів сервісної книжки;
- підвищення зручності сприйняття інтерфейсу.

Каталог res/menu/ містить XML-файли, що описують контекстні або верхні спливаючі меню для активностей. У застосунку реалізовано файл menu_language.xml, який використовується в головному екрані для перемикання мови інтерфейсу та доступу до довідкової інформації (лістинг 2.1).

Лістинг 2.1 – Фрагмент коду menu_language.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/lang_ua" android:title="Українська"/>
    <item android:id="@+id/lang_en" android:title="English"/>
    <item android:id="@+id/help" android:title="Допомога"/>
</menu>
```

Призначення меню:

- дозволяє динамічно перемикати інтерфейс між українською та

англійською мовами;

- забезпечує швидкий доступ до інструкції користувача (екран HelpActivity);

- підвищує гнучкість взаємодії користувача з додатком без необхідності повного перезапуску застосунку.

Каталог res/mipmap/ призначений для збереження іконок додатку на робочому столі пристрою та в системних меню. Android використовує іконки з mipmap замість drawable з метою оптимізації якості відображення та продуктивності на різних екранах і конфігураціях.

Типові файли:

- ic_launcher.png – основна іконка додатку;
- ic_launcher_round.png – округла версія іконки (відповідно до вимог Android 8+).

Кожна версія іконки має декілька розмірів: mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi. Переваги:

- автоматичне масштабування іконки для відповідного розміру екрана;
- узгоджений вигляд у системному лаунчері, панелі повідомлень, меню запущених додатків.

Графічні ресурси, меню та іконки в додатку Car Service App реалізовано відповідно до вимог дизайну Android-додатків. Стандартизований підхід до створення та використання ресурсів забезпечує адаптивність, зручність користування та візуальну послідовність інтерфейсу на всіх пристроях.

2.4 Ресурси для перекладу та стилів (values, values-uk, themes)

Підтримка багатомовності та єдиного стилю інтерфейсу є невід’ємною частиною якісного користувацького досвіду у сучасних мобільних застосунках. У додатку Car Service App реалізовано систему локалізації та стилізації за допомогою ресурсних файлів, що розташовані у відповідних

підкаталогах директорії res/.

Локалізовані текстові ресурси res/values/strings.xml. Цей файл містить основні текстові ресурси інтерфейсу англійською мовою – яка є мовою за замовчуванням у додатку. До таких ресурсів належать:

- заголовки екранів;
- назви кнопок;
- підказки (hints) у полях вводу;
- текст повідомлень (наприклад, про помилку або підтвердження дії);
- пункти меню.

res/values-uk/strings.xml Цей файл містить ідентичні ключі ресурсів, але перекладені українською мовою (лістинг 2.2). Android автоматично завантажує відповідний файл на основі мовних налаштувань системи або встановленої локалі в додатку.

Лістинг 2.2 – Приклад перекладеного рядка

```
<string name="add_car">Add Car</string>
<string name="add_car">Додати авто</string>
```

Зміна мови реалізована динамічно, тобто без потреби перевстановлення додатку або зміни мовних налаштувань у системі Android. За це відповідає клас LocaleHelper, який змінює локаль застосунку програмно (лістинг 2.3).

Лістинг 2.3 – Ключовий виклик

```
LocaleHelper.setLocale(context, "uk")
```

Після виклику функції відбувається оновлення контексту інтерфейсу та перезапуск поточної активності, що дозволяє миттєво застосувати нові мовні ресурси.

Теми та стилі інтерфейсу

res/values/themes.xml

Файл themes.xml відповідає за єдину візуальну стилізацію застосунку

та містить:

- базову тему для всіх екранів (Theme.AppCompat.Light.NoActionBar);
- визначення кольорів (фонові, основні, акцентні);
- стилі для елементів UI – кнопок, тексту, AppBar (Toolbar);
- підтримку Material Design;
- colors.xml, styles.xml, dimens.xml;
- colors.xml – містить кольори, які використовуються в усьому додатку (напр., primaryColor, backgroundColor, textColorPrimary) (лістинг 2.4).;
- styles.xml – визначає стандартні шрифти, розміри тексту, стилі кнопок;
- dimens.xml – уніфіковані розміри відступів, шрифтів, полів.

Лістинг 2.4 – Приклад кольору з colors.xml

```
<color name="primaryColor">#2196F3</color>
<color name="backgroundColor">#FFFFFF</color>
```

У застосунку Car Service App реалізовано повноцінну підтримку двомовного інтерфейсу (англійська та українська) з можливістю перемикання мови в реальному часі. Оформлення інтерфейсу здійснено згідно з принципами єдиного стилю, що підвищує читабельність, зручність і візуальну послідовність усіх екранів додатку.

У рамках проєкту Car Service App використано два основні конфігураційні файли: AndroidManifest.xml та build.gradle.kts. Ці файли відіграють ключову роль у визначенні структури додатку, його компонентів, дозволів, параметрів компіляції та зовнішніх залежностей. Вони є обов’язковими складовими кожного Android-застосунку.

Файл AndroidManifest.xml. Цей файл виконує функцію центрального конфігураційного опису додатку. У ньому зазначено всі компоненти, які застосунок використовує під час виконання, а також визначаються дозволи на доступ до системних ресурсів Android (лістинг 2.5).

Основні функції:

- оголошення активностей – кожен екран, реалізований як клас Activity, повинен бути задекларований у цьому файлі (лістинг 2.6).

- вказання стартової активності – активність, яка запускається першою після встановлення (у даному випадку – LauncherActivity) (лістинг 2.7).

- визначення дозволів – зокрема, дозвіл на надсилання сповіщень.

Лістинг 2.5 – Приклад дозволу

```
<uses-permission
android:name="android.permission.POST_NOTIFICATIONS"/>
```

Лістинг 2.6 – Приклад оголошення активності

```
<activity
android:name=".MainActivity"
android:exported="true"
android:label="Car Service App"/>
```

Лістинг 2.7 – Приклад оголошення стартової активності

```
<activity
android:name=".LauncherActivity"
android:exported="true">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
```

Файл build.gradle.kts є конфігураційним файлом збірки проєкту, реалізованим з використанням Kotlin DSL (Domain Specific Language) (лістинг 2.8). Він визначає параметри компіляції, залежності, версії SDK, плагіни та модулі. Цей файл обробляється системою Gradle під час складання застосунку (лістинг 2.9).

Лістинг 2.8 – Структура

```
plugins {
id("com.android.application")
kotlin("android")
android {
compileSdk = 33
defaultConfig {
```

```
applicationId = "com.example.carserviceapp"
minSdk = 21
targetSdk = 33
versionCode = 1
versionName = "1.0"
```

Лістинг 2.9 – Підключення бібліотек

```
dependencies {
implementation("androidx.appcompat:appcompat:1.6.1")
implementation("androidx.recyclerview:recyclerview:1.3.1")
implementation("com.google.android.material:material:1.9.0")
```

Основні можливості Gradle-конфігурації:

- визначення мінімальної (minSdk) та цільової (targetSdk) версій Android;
- встановлення версії додатку (versionCode, versionName);
- керування зовнішніми бібліотеками через систему залежностей;
- інтеграція Material Design компонентів, підтримка RecyclerView, сумісність із сучасними API Android.

Файли AndroidManifest.xml та build.gradle.kts забезпечують узгодженість і коректне функціонування всіх складових застосунку. Вони дозволяють централізовано визначати права доступу, структуру додатку, а також керувати середовищем розробки та виконання. Завдяки використанню Kotlin DSL у Gradle значно спрощується налаштування проєкту та підтримка залежностей, що відповідає сучасним вимогам до Android-розробки.

3 ОСНОВНІ ФУНКЦІОНАЛЬНІ КОМПОНЕНТИ

3.1 Авторизація та реєстрація користувача

Функціональний компонент авторизації та реєстрації реалізовано з метою забезпечення контролю доступу до персоналізованих даних користувача. Цей механізм гарантує, що кожен користувач взаємодіє лише з власною інформацією – зокрема транспортними засобами, сервісними записами та нагадуваннями [8]. Після запуску застосунку користувачеві надається можливість обрати тип облікового запису:

- користувач – фізична особа, яка обліковує обслуговування власного транспортного засобу;
- майстер – фахівець сервісного центру або технічного обслуговування, який може переглядати дані клієнтів за VIN-кодом;
- обрана роль зберігається локально та впливає на подальший доступ до функціоналу додатку. Наприклад, користувач працює зі списком власних авто, а майстер – з автомобілями клієнтів;
- інтерфейс реєстрації користувача реалізовано у вигляді стандартної форми з двома основними полями;
- Email-адреса – унікальний ідентифікатор користувача;
- пароль – захищений текстовий рядок, який не відображається при введенні (рисунок 3.1).

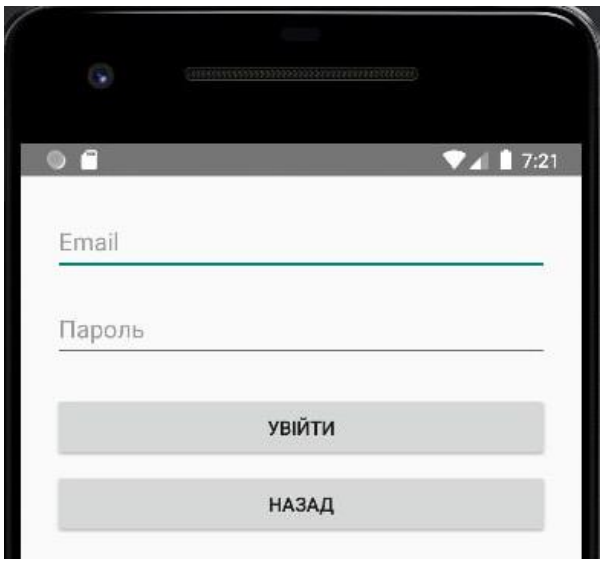

Рисунок 3.1 – Реєстрація користувача

Перед відправленням даних виконується валідація введених значень, зокрема:

- перевірка правильності формату email;
- мінімальні вимоги до довжини паролю (наприклад, не менше 6 символів);
- уникнення порожніх полів.

У разі успішної реєстрації дані зберігаються локально, й користувач автоматично авторизується та переходить на головну активність, відповідну до своєї ролі.

У разі повторного запуску застосунку користувач може увійти в систему через екран авторизації. Уведені дані перевіряються на відповідність раніше збереженим обліковим записам. У разі помилки система виводить відповідне повідомлення (наприклад: «Неправильний пароль» або «Користувача не знайдено»).

Інтерфейс екрану реєстрації та входу побудовано на основі XML-макетів із дотриманням сучасних принципів UX/UI. Передбачено:

- повернення на попередній екран (через стрілку "Назад" у Toolbar);
- кнопки для перемикання між реєстрацією та входом;
- візуальні підказки у полях введення;

- повідомлення про помилки вводу у вигляді Toast або TextView.

У поточній реалізації дані облікових записів зберігаються локально, що достатньо для прототипу або MVP-рівня. У майбутньому проєкт може бути масштабований за допомогою:

- інтеграції з Firebase Authentication;
- захищеного зберігання токенів доступу;
- підтримки відновлення паролю або двофакторної автентифікації.

Реалізація авторизації та реєстрації користувача є фундаментальним етапом функціонування мобільного застосунку. Чітке розмежування ролей, перевірка даних та підтримка зручного інтерфейсу забезпечують безпечний і персоналізований доступ до сервісів, які пропонує Car Service App [9].

3.2 Додавання та перегляд автомобілів

Основний екран мобільного застосунку Car Service App (активність MainActivity) призначений для відображення списку транспортних засобів, зареєстрованих поточним користувачем. Система реалізує зручний візуальний інтерфейс, у якому кожен автомобіль представлено у вигляді окремого елемента списку.

Для кожного транспортного засобу у списку зберігаються та відображаються такі атрибути:

- марка та модель автомобіля (наприклад, BMW 320i);
- рік випуску (у форматі YYYY);
- ідентифікаційний номер транспортного засобу (VIN) — унікальний 17-значний код;
- поточний пробіг, зазначений у кілометрах.

Вивід даних реалізовано за допомогою компонента RecyclerView у поєднанні з адаптером CarAdapter, який забезпечує динамічне оновлення списку автомобілів. Кожен елемент списку форматується відповідно до XML-шаблону item_car.xml, де передбачено іконку авто, текстову

інформацію та кнопку видалення.

Функція додавання транспортного засобу реалізована через окрему активність — AddCarActivity (рисунок 3.2). Вона містить форму з обов’язковими полями введення:

- марка автомобіля (EditText);
- модель;
- рік випуску (обмеження: лише числове значення в межах допустимого інтервалу);
- VIN-код (перевірка на довжину і допустимі символи);
- пробіг (лише числові значення).

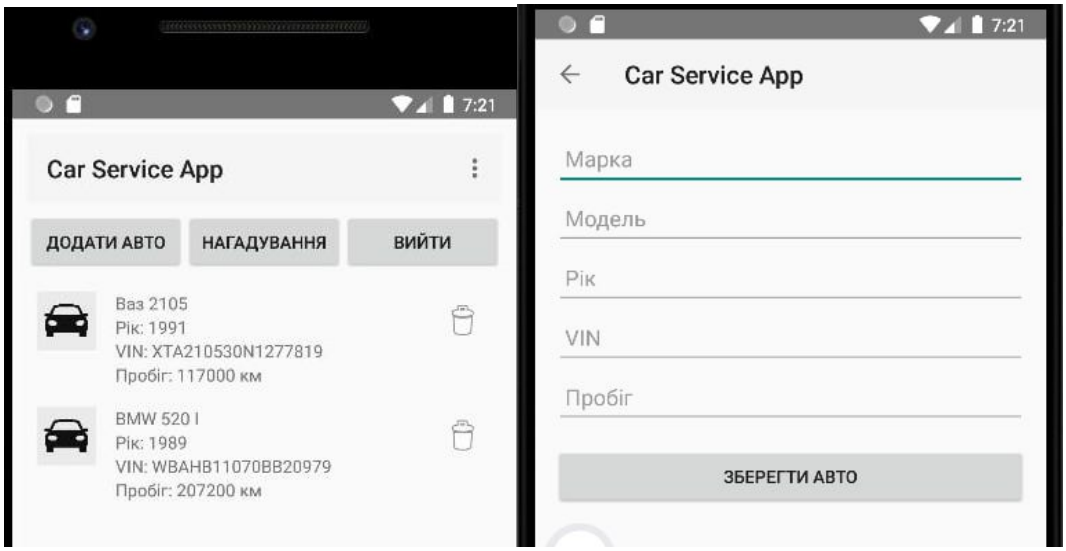

Рисунок 3.2 – Додавання авто

Після заповнення форми користувач натискає кнопку «Зберегти авто», у результаті чого:

- виконується валідація введених даних (наприклад, перевірка непорожніх полів, формату VIN);
- дані зберігаються у локальній базі (наприклад, через SharedPreferences або Room);
- користувач повертається на головний екран, де новий автомобіль з’являється у загальному списку.

Для кожного елемента списку передбачено кнопку видалення, яка дозволяє миттєво вилучити автомобіль із локальної бази даних. Після натискання на цю кнопку виконується підтвердження дії (через діалог або Snackbar), і у разі згоди – відповідний об'єкт видаляється, а список оновлюється.

Графічне оформлення екрану адаптовано до рекомендацій Material Design:

- використано стандартні компоненти: CardView, TextView, ImageView, FloatingActionButton;

- реалізовано плаваючу кнопку «+» у правому нижньому куті для додавання нового авто;

- кожен елемент списку візуально розділений і має чітку ієрархію інформації.

Функціональність додавання та перегляду автомобілів є центральною для користувача. Вона дозволяє створити повноцінну базу транспортних засобів, з якою пов’язуються інші компоненти застосунку — сервісна книжка, нагадування тощо. Завдяки реалізації через RecyclerView та адаптери забезпечено гнучкість, масштабованість та високу продуктивність навіть при великій кількості записів.

3.3 Сервісна книжка автомобіля

Функціональність ведення електронної сервісної книжки є ключовим компонентом мобільного застосунку Car Service App. Вона дозволяє користувачам документувати всі технічні обслуговування власного автомобіля у зручному цифровому форматі. Таким чином, забезпечується повна історія експлуатації транспортного засобу, яка зберігається локально та доступна у будь-який момент користувача перенаправляє до активності ServiceLogActivity, де відображаються всі записи, пов’язані з цим авто. Список реалізовано за допомогою RecyclerView, що дозволяє ефективно

працювати з великою кількістю елементів. Після натискання на конкретний автомобіль у списку (MainActivity).

Кожен запис у сервісній книжці є об’єктом моделі ServiceRecord і містить такі атрибути:

- дата обслуговування – вказується вручну або вибирається через діалогове вікно вибору дати (DatePickerDialog);

- тип робіт – вибирається із заздалегідь визначеного переліку (наприклад, «двигун», «ходова частина», «електрика», «кузов»);

- пробіг на момент виконання обслуговування – числове значення у кілометрах;

- коментар користувача – довільне текстове поле, що дозволяє уточнити суть або деталі виконаних робіт.

Форма для створення нового запису відкривається за допомогою плаваючої кнопки «+», що знаходиться на екрані сервісної книжки. Відповідна активність (AddServiceActivity) містить усі необхідні поля для введення даних.

Після натискання кнопки «Зберегти»:

- здійснюється валідація введених даних (усі поля обов’язкові).

- об'єкт ServiceRecord додається до локального списку, асоційованого з відповідним авто.

- список записів у ServiceLogActivity оновлюється автоматично.

Для кожного запису реалізовано дві основні операції:

- редагування (EditServiceRecordActivity) – дозволяє змінити будь-який параметр наявного запису, зокрема тип обслуговування, дату чи коментар. Після збереження зміни негайно відображаються у списку.

- видалення – виконується за допомогою відповідної кнопки біля елемента списку. Після підтвердження дії запис вилучається з локальної пам’яті, а інтерфейс оновлюється.

Кожен запис оформлено за допомогою XML-шаблону item_service_record.xml, у якому структуровано вивід дати, типу, пробігу та

коментаря. Використано компоненти CardView, TextView, ImageButton для реалізації сучасного і зручного вигляду згідно з рекомендаціями Material Design.

Основним перевагами електронної сервісної книжки є (рисунок 3.3):

- відсутність ризику втрати, як у випадку з паперовими аналогами;
- миттєвий доступ до історії обслуговування в будь-який момент;
- можливість редагування, доповнення і сортування даних;
- підвищення відповідальності користувача щодо обслуговування власного авто.

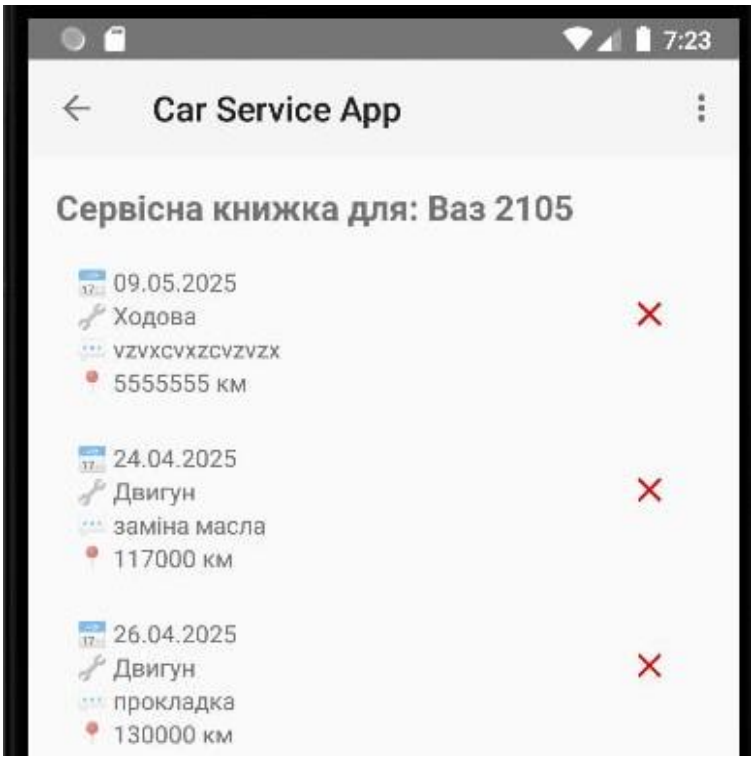

Рисунок 3.3 – Сервісна книжка автомобіля

Підсистема сервісної книжки у застосунку Car Service App забезпечує зручний механізм збереження, перегляду та редагування записів обслуговування автомобіля. Її реалізація дозволяє користувачеві підтримувати повноцінну історію технічних робіт, що підвищує ефективність експлуатації транспортного засобу та покращує планування майбутніх сервісів.

3.4 Створення та перегляд нагадувань

З метою підвищення надійності експлуатації транспортного засобу та зручності користувача, у мобільному застосунку Car Service App реалізовано функціональність створення та перегляду нагадувань про планове технічне обслуговування. Ця підсистема дозволяє своєчасно інформувати власника автомобіля про потребу в здійсненні регламентних робіт згідно з рекомендованими інтервалами.

На відповідній вкладці застосунку (активність RemindersActivity) реалізовано хронологічний список рекомендованих сервісних процедур, що супроводжуються інтервалами обслуговування за пробігом (у кілометрах) (таблиця 3.1).

Таблиця 3.1 - Типові нагадування

Тип обслуговування	Рекомендований пробіг
Заміна моторної оливи	10 000 км
Заміна повітряного фільтра	15 000 км
Перевірка ходової частини	20 000 км
Заміна гальмівної рідини	30 000 км
Заміна ременя ГРМ	60 000 км

Усі записи представлені в інтерфейсі за допомогою RecyclerView, який адаптований для відображення об’єктів моделі Reminder через адаптер ReminderAdapter (рисунок 3.4).

Формування списку нагадувань здійснюється за допомогою спеціального класу ServiceScheduleProvider, який містить наперед визначений перелік регламентних робіт та відповідних пробігів. Структура побудована на базі масиву або списку об'єктів Reminder, що динамічно передається адаптеру для виведення на екран (лістинг 3.1).

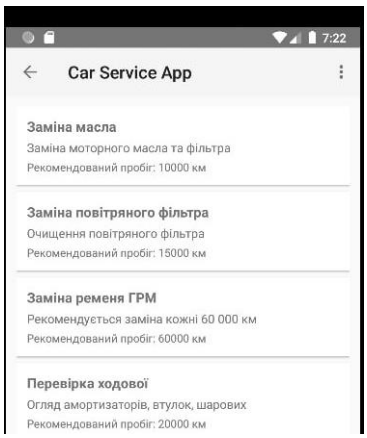

Рисунок 3.4 - Список рекомендованих сервісних процедур

Лістинг 3.1 - Приклад створення запису

```
Reminder("Заміна масла", 10000)
Reminder("Гальмівна рідина", 30000)
```

Завдяки цій реалізації користувач не повинен вводити дані вручну – додаток автоматично формує перелік сервісів згідно з типовими інструкціями з технічного обслуговування, рекомендованими виробниками.

У поточній реалізації перелік нагадувань є уніфікованим для всіх авто. Проте програмна структура класу ServiceScheduleProvider передбачає можливість подальшого розширення функціоналу:

- індивідуалізація рекомендацій для різних моделей авто;
- введення користувачем власних правил обслуговування;
- інтеграція з сервісними записами та пробігом для автоматичного сповіщення (наприклад, через NotificationHelper при досягненні визначеного порогу).

Інтерфейс вкладки нагадувань розроблено відповідно до рекомендацій Material Design. Кожне нагадування представлено у вигляді `CardView`-елемента з описом робіт та рекомендованим пробігом. При потребі реалізація може бути доповнена індикаторами стану: «Виконано», «Скоро», «Прострочено».

Функціональність нагадувань у мобільному застосунку Car Service App дозволяє користувачам планувати технічне обслуговування згідно з

типовими регламентами. Автоматизоване надання рекомендацій через ServiceScheduleProvider зменшує ймовірність пропуску важливих сервісів, підвищує рівень технічної обізнаності користувача та сприяє збереженню належного технічного стану транспортного засобу.

3.5 Перемикач мови інтерфейсу та локальні сповіщення

З метою підвищення доступності та зручності використання, мобільний застосунок Car Service App реалізує повноцінну підтримку багатомовного інтерфейсу, що дозволяє користувачам обирати мову взаємодії відповідно до власних уподобань. У поточній версії реалізовано підтримку двох мов — української та англійської, що відповідає вимогам до сучасних користувацьких продуктів у міжнародному середовищі.

Перемикання мови реалізовано через вбудоване меню, розташоване у правій верхній частині головного екрана (активність MainActivity) (рисунок 3.5), (лістинг 3.2).

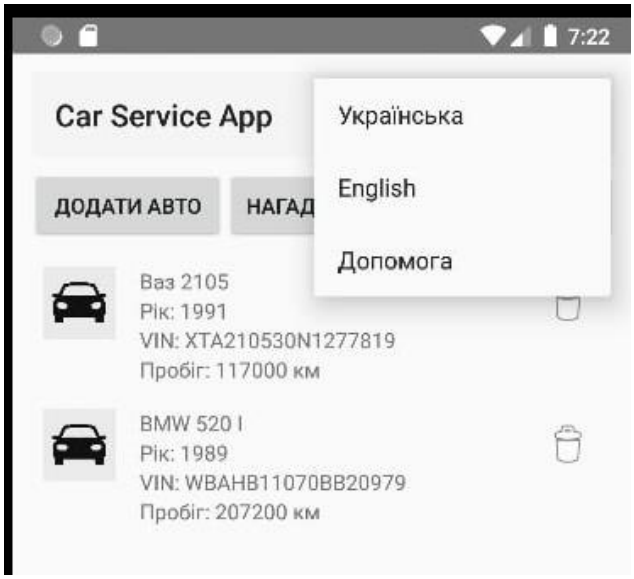

Рисунок 3.5 - Перемикач мови інтерфейсу

Лістинг 3.2 - Відповідне меню (menu_language.xml) містить пункти

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
  <item android:id="@+id/lang_ua" android:title="Українська"/>
  <item android:id="@+id/lang_en" android:title="English"/>
  <item android:id="@+id/help" android:title="Допомога"/>
</menu>
```

При виборі пункту «Українська» або «English» застосунок динамічно змінює мову інтерфейсу без потреби повного перезапуску.

Реалізація через LocaleHelper.Ф ункціональність перемикання мови реалізовано за допомогою спеціального класу LocaleHelper, який виконує зміну мовних налаштувань у контексті Android-додатку (лістинг 3.3).

Лістинг 3.3 - Основний метод

```
LocaleHelper.setLocale(context, "uk")
```

Цей виклик змінює поточну локаль, оновлює контекст і перезапускає активність, щоб інтерфейсна частина була повністю оновлена відповідно до нових ресурсів. Завдяки цьому забезпечується миттєве застосування вибраної мови, що підвищує зручність користування додатком.

Інтерфейс додатку використовує стандартні ресурсні файли strings.xml, розміщені у відповідних каталогах (лістинг 3.4):

- res/values/strings.xml — англійська мова (мова за замовчуванням);
- res/values-uk/strings.xml — українська локалізація.

Кожен текстовий елемент у додатку прив'язано до ключа ресурсу, що забезпечує гнучку підтримку локалізації та централізоване оновлення інтерфейсу.

Лістинг 3.4 - Приклад локалізованих рядків

```
<string name="add_car">Add Car</string>

<!-- UK -->
<string name="add_car">Додати авто</string>
```

Переваги реалізованої локалізації:

- зручність для користувачів, які не володіють англійською мовою;
- можливість розширення локалізації на інші мови у майбутньому (наприклад, німецька, польська тощо);
- підвищення конкурентоспроможності продукту на міжнародному ринку;
- відповідність практикам i18n (інтернаціоналізації) в Android-розробці.

Механізм перемикання мови у застосунку Car Service App реалізовано відповідно до рекомендацій платформи Android, із використанням ресурсів strings.xml та класу LocaleHelper. Це забезпечує зручну зміну мови без необхідності перезапуску всього додатку, підвищуючи якість взаємодії з інтерфейсом та орієнтованість застосунку на ширшу аудиторію.

З метою підвищення зручності та інформативності взаємодії користувача із застосунком Car Service App, реалізовано механізм локальних сповіщень, який дозволяє оперативно повідомляти про наближення терміну виконання регламентного технічного обслуговування. Такий підхід сприяє підвищенню відповідальності користувача, забезпечує своєчасність дій і знижує ризики, пов’язані з пропуском критичних сервісних процедур.

Сповіщення формуються автоматично, на основі:

- даних про пробіг, які пов’язані з конкретним транспортним засобом;
- інформації про зареєстровані нагадування (модель Reminder);
- типових регламентів обслуговування, наданих класом ServiceScheduleProvider.

Користувач отримує системне повідомлення без потреби відкривати застосунок, що дозволяє реагувати на потребу в обслуговуванні навіть поза межами активного використання програми (лістинг 3.5).

Архітектура реалізації: клас NotificationHelper.Механізм побудований на основі окремого класу NotificationHelper, який виконує усі операції, пов’язані зі створенням і відображенням сповіщень (лістинг 3.6).

Створення Notification Channel – відповідно до вимог Android 8.0 (API

26+) (лістинг 3.7).

Лістинг 3.5 - Визначення рівня важливості

```
val channel = NotificationChannel(
    CHANNEL_ID,
    "Car Service Notifications",
    NotificationManager.IMPORTANCE_DEFAULT
)
```

Лістинг 3.6 - Налаштування зовнішнього вигляду сповіщення

```
val notification = NotificationCompat.Builder(context,
CHANNEL_ID)
    .setSmallIcon(R.drawable.ic_car_notification)
    .setContentTitle("Час обслуговування")
    .setContentText("Заміна моторної оливи на пробігу 10 000
км")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .build()
```

Лістинг 3.7 - Ініціація показу

```
val manager =
context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
manager.notify(notificationId, notification)
```

При досягненні певного порогу пробігу (наприклад, 10 000 км) застосунок виконує перевірку наявних нагадувань. Якщо визначена сервісна операція має бути виконана – активується метод класу NotificationHelper, який надсилає повідомлення користувачу.

Приклад повідомлення: «Час обслуговуванняЗаміна моторної оливи – пробіг досяг 10 000 км»

Це дозволяє користувачу прийняти рішення про обслуговування навіть без відкриття сервісної книжки.

Переваги реалізації:

- висока інтеграція з екосистемою Android (відповідність API-рівню 26+);
- повідомлення активуються без втручання користувача;

- підвищення надійності експлуатації авто за рахунок нагадувань.
- можливі напрямки розвитку;
- інтеграція з календарем для відстеження ТО за датою;
- можливість встановлення персональних інтервалів обслуговування;
- розширення системи сповіщень для повторних нагадувань або нагадувань за часом (не лише за пробігом);
- реалізація push-сповіщень у разі підключення до серверної бази або Firebase.

Система локальних сповіщень, реалізована у застосунку Car Service App, є ефективним інструментом для проактивного інформування користувача про технічне обслуговування автомобіля. Вона сприяє зменшенню людського чинника, дозволяє підтримувати транспортний засіб у належному стані та забезпечує високий рівень зручності взаємодії із

3.6 Збереження моделей (Car, ServiceRecord, Reminder) (NotificationHelper) та сервісні регламенти (ServiceScheduleProvider) та сервісні регламенти

Основою функціональної логіки мобільного застосунку Car Service App є три ключові моделі даних, які забезпечують структуроване представлення об’єктів предметної області – транспортних засобів, сервісних записів та нагадувань. Усі моделі реалізовано за допомогою data class у Kotlin, що спрощує їх серіалізацію, передачу між активностями та збереження.

Модель Car описує транспортний засіб користувача та містить такі обов’язкові поля:

- марка (String) – назва виробника (наприклад, Toyota);
- модель (String) – конкретна модель авто (наприклад, Corolla);
- рік випуску (Int) – рік виробництва;
- VIN-код (String) – унікальний 17-символьний ідентифікатор транспортного засобу;

- пробіг (Int) – поточний пробіг у кілометрах.

Ця модель є основною для побудови списку авто на головному екрані (MainActivity) та базовою сутністю, до якої прив'язуються інші моделі.

Об’єкт ServiceRecord представляє запис у сервісній книжці для конкретного автомобіля. Він містить детальну інформацію про проведене технічне обслуговування:

- дата обслуговування (String / LocalDate) – дата проведення сервісу;
- тип робіт (String) – одна з попередньо визначених категорій: «двигун», «ходова частина», «трансмісія» тощо;
- пробіг на момент обслуговування (Int) – значення одометра на момент ремонту;
- коментар (String) – довільний текстовий опис виконаних робіт або рекомендацій.

Ця модель використовується для побудови сервісної книжки (ServiceLogActivity) та дозволяє користувачеві формувати повну історію експлуатації авто.

Модель Reminder призначена для зберігання нагадувань про планове технічне обслуговування. Вона містить такі поля:

- назва нагадування (String) – опис сервісної процедури (наприклад, «Заміна моторної оливи»);
- рекомендований пробіг або дата (Int або String) – інтервал, через який слід виконати обслуговування;
- статус виконання (Boolean або Enum) – ознака виконаності нагадування (може бути розширено до: «очікує», «виконано», «прострочено»).

Нагадування формуються на основі класу ServiceScheduleProvider або можуть бути створені вручну у майбутніх версіях застосунку.

У поточній реалізації проєкту усі дані зберігаються локально у пам’яті пристрою. Для цього використовуються такі підходи:

- SharedPreferences – для зберігання облікових записів користувачів,

обраної мови тощо;

- локальні списки у пам’яті (MutableList) – для зберігання тимчасових даних моделей Car, ServiceRecord, Reminder;
- серіалізація (лістинг 3.8) до JSON-формату – моделі перетворюються у рядки (String) за допомогою бібліотек типу Gson або kotlinx.serialization і зберігаються як JSON-структури (лістинг 3.9).

Лістинг 3.8 - Приклад серіалізації

```
val json = gson.toJson(carList)
sharedPreferences.edit().putString("cars", json).apply()
```

Лістинг 3.9 - Приклад десеріалізації

```
val json = sharedPreferences.getString("cars", "")
val carList = gson.fromJson(json, object :
TypeToken<List<Car>>() {}.type)
```

Подальший розвиток: використання Room.

У майбутніх версіях додатку планується впровадження Room-персистентної бібліотеки, яка забезпечує:

- створення локальної SQLite-бази даних;
- автоматичне оновлення змін у списках через LiveData;
- забезпечення цілісності даних за допомогою анотацій (@Entity, @Dao, @Query).

Це значно підвищить масштабованість, продуктивність та безпеку збереження даних.

Три основні моделі – Car, ServiceRecord та Reminder – формують основу внутрішньої структури даних застосунку Car Service App. Завдяки зручному механізму серіалізації та локального зберігання реалізовано повноцінне керування об’єктами у межах життєвого циклу додатку. Перспективи переходу до Room забезпечують потенціал для розширення та використання додатку на практиці.

Для підвищення точності планування технічного обслуговування в

мобільному застосунку Car Service App реалізовано окремий компонент — ServiceScheduleProvider, який відповідає за збереження та надання типових регламентів обслуговування транспортного засобу. Цей підхід дозволяє користувачу не вводити вручну інтервали обслуговування, а використовувати заздалегідь визначений перелік сервісних операцій із рекомендованими пробігами.

Призначення компонента:

- ServiceScheduleProvider є утилітним класом, який генерує список об’єктів Reminder, кожен з яких містить інформацію про певну сервісну процедуру. Ці дані використовуються у двох основних підсистемах:

- вкладка “Нагадування” – для виведення користувачу типових сервісних операцій;

- система локальних сповіщень (NotificationHelper) – для автоматичного інформування при досягненні контрольного пробігу.

Кожен запис у стандартному регламенті включає дані з таблиці 3.2.

Таблиця 3.2 - Структура регламенту

Тип обслуговування	Рекомендований пробіг
Заміна моторної оливи	10 000 км
Заміна повітряного фільтра	15 000 км
Перевірка ходової частини	20 000 км
Заміна гальмівної рідини	30 000 км
Заміна ременя ГРМ	60 000 км

Усі значення задаються у кілометрах та зберігаються як числові значення (типу Int), що дозволяє зручно проводити порівняння з поточним пробігом автомобіля (лістинг 3.10).

Лістинг 3.10 - Приклад реалізації класу

```
object ServiceScheduleProvider {
```

```
fun getDefaultSchedule(): List<Reminder> {
    return listOf(
        Reminder("Заміна моторної оливи", 10000),
        Reminder("Заміна повітряного фільтра", 15000),
        Reminder("Перевірка ходової частини", 20000),
        Reminder("Заміна гальмівної рідини", 30000),
        Reminder("Заміна ременя ГРМ", 60000)
    )
}
```

Клас реалізовано у вигляді object (Singleton), що дозволяє забезпечити централізований доступ до даних без потреби створення екземплярів.

Інтеграція з іншими модулями:

- ReminderAdapter – використовує список, наданий ServiceScheduleProvider, для виведення на екран користувачу у вигляді нагадувань;
- NotificationHelper – аналізує пробіг кожного авто та порівнює його з рекомендованими значеннями із регламенту для автоматичної генерації сповіщень.

Модель Reminder – є об’єктом передачі даних (DTO), що містить поля назви, пробігу та статусу.

Хоча у поточній версії регламент обслуговування є уніфікованим для всіх транспортних засобів, програмна структура класу ServiceScheduleProvider дозволяє реалізувати індивідуальні графіки обслуговування в майбутньому. Зокрема:

- персоналізація регламентів для конкретної марки або моделі авто;
- можливість редагування типових інтервалів користувачем;
- збереження кастомізованих профілів ТО у базі даних або хмарному сервісі;
- прив’язка регламенту до VIN-коду.

Компонент ServiceScheduleProvider у застосунку Car Service App відіграє ключову роль у формуванні автоматизованих нагадувань про технічне обслуговування. Він забезпечує стандартизований підхід до обліку сервісних інтервалів, інтегрується з іншими підсистемами додатку та має

потенціал до подальшого розширення для персоналізованого обслуговування автомобілів.

4 ІНТЕРФЕЙС КОРИСТУВАЧА

4.1 Структура XML-макетів активностей

Інтерфейс користувача мобільного застосунку Car Service App побудовано на основі XML-розміток, що зберігаються у директорії res/layout/. Для кожної активності застосунку створено окремий XML-файл, який описує просторову структуру компонентів інтерфейсу – таких як заголовки (TextView), поля вводу (EditText), кнопки (Button), списки (RecyclerView), меню та інші елементи керування (таблиця 4.1).

Це відповідає концепції розділення логіки (Kotlin-код) та представлення (XML-макети), що є стандартною практикою в Android-розробці та сприяє зручності масштабування, тестування й підтримки інтерфейсу (лістинг 4.1).

Таблиця 4.1 - Основні макети активностей

XML-файл	Призначення
activity_login.xml	Екран входу користувача з полями для email і паролю
activity_register.xml	Екран реєстрації нового користувача
activity_main.xml	Головна панель користувача зі списком зареєстрованих авто
activity_add_car.xml	Форма для додавання нового транспортного засобу
activity_service_log.xml	Список сервісних записів для вибраного авто
activity_add_service.xml	Форма створення нового запису обслуговування
activity_reminder.xml	Вкладка технічне обслуговування технічне

У більшості макетів використано сучасні контейнери компонування, такі як:

- LinearLayout – для вертикального або горизонтального розміщення елементів у спрощених інтерфейсах;

- ConstraintLayout – як гнучкий і оптимізований спосіб компонування, що дозволяє розміщувати елементи відносно один одного та меж контейнера;

- ScrollView – у формах із великою кількістю полів (наприклад, при реєстрації чи додаванні сервісного запису), що забезпечує вертикальне прокручування.

Також використано FloatingActionButton для виклику додаткових дій (наприклад, додавання нового запису чи авто), CardView для обрамлення елементів списку, а також Toolbar як верхню панель управління.

Лістинг 4.1 - Фрагмент макету activity_add_car.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextMake"
        android:hint="Марка"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <EditText
        android:id="@+id/editTextModel"
        android:hint="Модель"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

  <!-- Інші поля -->

  <Button
    android:id="@+id/buttonSaveCar"
    android:text="Зберегти авто"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

</LinearLayout>
```

Завдяки використанню dp-одиниць виміру, відносних відступів та ConstraintLayout, інтерфейси зберігають коректний вигляд на різних типах пристроїв – як на смартфонах, так і на планшетах. Це відповідає принципам адаптивного дизайну, рекомендованим Google.

Структура XML-макетів у застосунку Car Service App побудована відповідно до принципів модульності, масштабованості та інтерфейсної доступності. Використання окремих макетів для кожної активності сприяє зрозумілості структури проєкту, а вибір компонентів компонування дозволяє забезпечити узгоджений зовнішній вигляд на пристроях із різними параметрами дисплея.

4.2 Адаптери списків (RecyclerView Adapters)

Для відображення динамічних списків у мобільному застосунку Car Service App використано компонент RecyclerView, що є рекомендованим рішенням Android для ефективної візуалізації колекцій даних. Цей компонент забезпечує високу продуктивність, повторне використання елементів інтерфейсу (view recycling) та зручність реалізації кастомної логіки взаємодії.

Для кожного типу списку в застосунку створено окремий адаптер, реалізований як Kotlin-клас, який відповідає за прив’язку даних до елементів інтерфейсу та керування відображенням (таблиця 4.2).

Таблиця 4.2 - Основні адаптери

Клас адаптера	Призначення
CarAdapter	Відображення автомобілів користувача на головному екрані
ServiceRecordAdapter	Список записів технічного обслуговування у сервісній книжці
ReminderAdapter	Вивід списку рекомендацій з технічного обслуговування
MasterCarAdapter	Список авто, які доступні для перегляду майстром (за VIN)

Кожен адаптер використовує шаблон XML-розмітки для виводу одного елемента списку, що розташовується в директорії res/layout/ (таблиця 4.3).

Таблиця 4.3 - XML-шаблони елементів списку

XML-файл	Призначення
item_car.xml	Відображення одного авто у списку користувача
item_service_record.xml	Один запис у сервісній книжці
item_reminder.xml	Окремий елемент у списку нагадувань
item_master_car.xml	Автомобіль у списку для перегляду майстром

Ці шаблони містять компоненти TextView, ImageView, Button або ImageButton, об’єднані через ConstraintLayout або LinearLayout, що забезпечує зручну адаптацію під екран будь-якого розміру (лістинг 4.2).

Кожен адаптер реалізує шаблон ViewHolder, який відповідає за зберігання посилань на елементи інтерфейсу та їх повторне використання. Це знижує навантаження на систему при прокручуванні списку.

Лістинг 4.2 – Структура типової реалізації адаптера

```
class CarAdapter(
    private val carList: List<Car>,
    private val onDeleteClick: (Car) -> Unit
) : RecyclerView.Adapter<CarAdapter.CarViewHolder>() {

    inner class CarViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        val carNameTextView: TextView =
itemView.findViewById(R.id.textViewCarName)
        val deleteButton: ImageButton =
itemView.findViewById(R.id.buttonDelete)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): CarViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_car, parent, false)
        return CarViewHolder(view)
```

```
    }

    override fun onBindViewHolder(holder: CarViewHolder,
position: Int) {
        val car = carList[position]
        holder.carNameTextView.text = "${car.make} ${car.model}"
        holder.deleteButton.setOnClickListener {
onDeleteClick(car) }
    }

    override fun getItemCount(): Int = carList.size
}
```

У кожному адаптері передбачено обробку подій користувача, зокрема:

- натискання на елемент списку (для перегляду детальної інформації);
- натискання кнопки «видалити» або «редагувати»;
- передача дій через функціональні інтерфейси або callback-функції.

Цей підхід дозволяє зробити адаптери гнучкими та незалежними, що спрощує повторне використання компонентів інтерфейсу у різних контекстах (наприклад, один і той самий адаптер може використовуватись у різних активностях).

Завдяки використанню адаптерів RecyclerView у застосунку Car Service App реалізовано ефективне, масштабоване і продуктивне відображення динамічних даних. Такий підхід дозволяє повністю контролювати вигляд і поведінку кожного елемента списку, спрощує обробку взаємодій та забезпечує високий рівень UX навіть на пристроях із невеликим обсягом оперативної пам’яті.

4.3 Стилі, кольори та переклади

Для забезпечення єдиного стилістичного оформлення, а також реалізації багатомовного інтерфейсу, в мобільному застосунку Car Service App використано механізми ресурсів Android, які зберігаються у відповідних підкаталогах директорії res/. Це дозволяє централізовано керувати виглядом додатку, спрощувати підтримку локалізації та забезпечувати адаптивність до

користувацьких мовних уподобань (лістинг 4.3).

Лістинг 4.3 – Стилі та кольори

```
res/values/themes.xml
res/values/colors.xml
res/values/styles.xml (за потреби)
```

Основні положення.

В якості основної теми застосовано Theme.AppCompat.Light.NoActionBar, яка забезпечує світлий інтерфейс із можливістю ручного управління Toolbar у кожній активності (таблиця 4.4).

Таблиця 4.4 - Кольорова палітра

Назва кольору	Значення	Призначення
colorPrimary	#2196F3	Основний акцент (кнопки, заголовки)
colorAccent	#FF9800	Додаткові елементи (FAB, іконки)
colorBackground	#FFFFFF	Фон екранів
textColorPrimary	#000000	Основний текст

У файлі themes.xml також визначено (лістинг 4.4):

- шрифти та розміри тексту;
- стилі для EditText, Button, Toolbar;
- кути заокруглення елементів (якщо використовується shape-ресурс);
- загальні параметри materialThemeOverlay.

Лістинг 4.4 - Приклад

```
<resources>
    <style name="AppTheme"
parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorAccent">@color/colorAccent</item>
```

```
        <item
name="android:textColor">@color/textColorPrimary</item>
    </style>
</resources>
```

Це забезпечує візуальну послідовність між екранами, знижує вартість підтримки інтерфейсу та дає змогу швидко змінювати дизайн (лістинг 4.5).

Лістинг 4.5 - Переклади та локалізація

```
res/values/strings.xml (англійська мова)
res/values-uk/strings.xml (українська мова)
```

Для підтримки багатомовного інтерфейсу всі текстові елементи винесено у вигляді ключів у файли strings.xml. Це дозволяє динамічно завантажувати мовні ресурси в залежності від локалі пристрою або налаштувань користувача (лістинг 4.6).

Лістиг 4.6 - Для підтримки багатомовного інтерфейсу

```
<!-- Англійський варіант -->
<string name="add_car">Add Car</string>

<!-- Український переклад -->
<string name="add_car">Додати авто</string>
```

Механізм перемикання мови.

Перемикання мови реалізовано програмно, за допомогою спеціального утилітного класу LocaleHelper, що дозволяє оновити локаль додатку без повного перезапуску процесу (лістинг 4.7).

Лістинг 4.7 - Користувач вибирає мову через меню у верхньому правому куті екрана

```
LocaleHelper.setLocale(context, "uk")
```

Поточна активність перезавантажується, а ресурси підтягуються з відповідної директорії локалізації.’Усі надписи в інтерфейсі автоматично

оновлюються відповідно до обраної мови.

Переваги реалізованого підходу:

- централізоване керування стилем і текстами;
- миттєве оновлення інтерфейсу без перезапуску додатку;
- просте додавання нових мов у майбутньому (наприклад, values-pl для польської);
- повна відповідність практикам інтернаціоналізації (i18n) та локалізації (l10n) у Android.

Завдяки використанню системи ресурсів Android застосунок Car Service App підтримує повноцінну локалізацію та єдине стилістичне оформлення. Це забезпечує високу якість користувацького досвіду, адаптацію під аудиторію з різними мовними вподобаннями та гнучкість у майбутній розробці інтерфейсних компонентів.

ВИСНОВКИ

У даній кваліфікаційній роботі було виконано повний цикл досліджень та розробки програмного засобу для реалізації цифрової сервісної книжки автомобіля, який забезпечує облік технічного обслуговування транспортного засобу в цифровому форматі.

Проведено аналіз сучасних підходів до організації обліку сервісного обслуговування транспортних засобів, виявлено основні недоліки паперових сервісних книжок та переваги цифрових рішень.

Обґрунтовано вибір технологій для реалізації цифрової сервісної книжки, включаючи архітектуру клієнт-серверного застосунку, мову програмування, базу даних та інтерфейсні рішення.Розроблено програмний засіб, що дозволяє:

- реєструвати та ідентифікувати автомобіль і власника;
- вести журнал технічного обслуговування з фіксацією дати, типу робіт, виконавця та пробігу;
- надавати доступ до інформації для третіх сторін за дозволом користувача.

Реалізовано механізми зберігання даних, захисту інформації та контролю доступу, що відповідають вимогам сучасних інформаційних систем. Проведено тестування програмного забезпечення, підтверджено його працездатність, зручність інтерфейсу та відповідність функціоналу поставленим вимогам.Запропоновано напрямки подальшого розвитку системи, зокрема:

- розширення функціоналу для сервісних центрів;
- реалізація мобільного додатку;
- інтеграція з державними базами даних і системами електронного документообігу;
- використання QR-ідентифікаторів для швидкого доступу до історії

обслуговування.

Таким чином, поставлене в роботі завдання виконано повністю. Розроблене програмне забезпечення має прикладну цінність і може бути впроваджене в реальних умовах як окремий сервіс або як модуль у складі комплексних інформаційних систем автомобільного обслуговування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jemerov D., Isakova S. Kotlin in Action. Shelter Island: Manning Publications, 2017.

2. Phillips B., Stewart K. Android Programming: The Big Nerd Ranch Guide (Kotlin Edition). Atlanta: Big Nerd Ranch, 2019.

3. Subramaniam V. Programming Kotlin. Dallas: Pragmatic Bookshelf, 2019. .

4. Eckel B., Isakova S. Atomic Kotlin. Mindview, 2021. .

5. Griffiths D., Griffiths D. Head First Android Development: A Brain-Friendly Guide. 3rd ed. Sebastopol: O'Reilly Media, 2021. .

6. Burton M. Android App Development for Dummies. 3rd ed. Hoboken: John Wiley & Sons, 2022.

7. Murphy M. L. The Busy Coder's Guide to Android Development. – CommonsWare, 2021. .

8. Олефір М.О., Єрошенко О.А. Програмні засоби розробки цифрової сервісної книжки автомобіля. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей 15 міжнародної науково-технічної конференції 24-25 квітня 2025 року. Баку, Харків, Жиліна. 2025. С. 31.

9. Олефір М.О., Єрошенко О.А. Цифрова сервісна книжка для автомобіля. Проблеми інформатизації: тези доповідей 12-тої міжнародної науково-технічної конференції. Т.2. 2024. С. 124.