Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка мобільного додатку взаємодії клієнта з СТО»

Студента 2м курсу, 6з групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Горбуля Михайла Олексійовича

Науковий керівник кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Палагута Катерина Олексіївна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Горбулю Михайлу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка мобільного додатку взаємодії клієнта з СТО»
Затверджена наказом ректора від "24" грудня 2019 р. № 4440

2. Строк здачі студентом закінченої проекту 01 грудня 2020 р.

3. Цільова установка та вихідні дані до проекту
Мета проекту розробка мобільного додатка, що містить інструменти взаємодії зі споживачами послуг автосервісу.
Об'єкт дослідження процес розробки мобільного додатка
Предмет дослідження розробка мобільного додатка, який має поліпшити взаємодію співробітників станції технічного обслуговування і клієнтів.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ВЗАЄМОДІЇ СТО З КЛІЄНТАМИ

1.1. Обслуговування ТЗ

1.2. Взаємодія СТО з клієнтом

1.3. Аналіз існуючих додатків для діагностики і обслуговування

1.4. Висновок до розділу 1

РОЗДІЛ 2. ОСОБЛИВОСТІ РОЗРОБКИ ANDROID ДОДАТКУ

2.1. Архітектура Android платформи

2.2. .Середовище розробки Android Studio

2.3. Система автоматичної сбірки Gradle

2.4. Хмарна база даних Firebase

2.5. Висновок до розділу 2

РОЗДІЛ 3. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ANDROID ДОДАТКУ

3.1. Інтерфейс програмного модулю

3.2. Архітектура

3.3. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проект*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Аналіз існуючих рішень взаємодії сто з клієнтами*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Особливості розробки android додатку*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Особливості реалізації android додатку*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускного кваліфікаційного проекту*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускного кваліфікаційного проекту*	*10.12.2020-11.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Палагута К. О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Горбуль М.О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист _____Палагута К. О._____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _Горбуль М. О._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Криворучко О. В._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Робота присвячена створенню мобільного додатку для взаємодії клієнта з СТО.

Додаток дозволяє: моніторити інформацію про ремонтні роботи та їх вартість; робити запис на обслуговування; збирати дані з різних апаратних модулів автомобіля використовуючи протокол OBDII; здійснювати передачу даних бездротовими каналами зв'язку. В процесі розробки була використана технологія бездротового зв'язку Bluetooth. Серверна частина розгорнута за допомогою сервісу Firebase, а клієнтська створена в Android Studio. Мова програмування – Java. Проєкт зібрано за допомогою Gradle.

Упровадження цього додатку дозволить станціям технічного обслуговування автомобілів зручно і швидко взаємодіяти з клієнтом, надаваінформацію про виконані роботи, а також дистанційно проводити моніторинг автомобіля. Це зменшить витрати коштів та часу для аналізу поломки автомобіля як зі сторони користувача так і зі сторони СТО.

Ключові слова: Android, мобільний додаток, станція технічного обслуговування, Bluetooth, OBDII.

ABSTRACT

The work is devoted to the creation of a mobile application for customer interaction with the service station.

The application allows you to: monitor information about repairs and its cost; make an entry for service; collect data from various hardware modules of the car using the OBDII protocol; transmit data via wireless communication channels. Bluetooth wireless technology was used in the development process. The server part is deployed using the Firebase servis, and the client part is created in Android Studio. Programming language - Java. The project was compiled using Gradle.

The introduction of this application will allow car service stations to conveniently and quickly interact with the customer, provide information about the work performed, as well as remotely testing the car. This will reduce the cost and time to analyze the breakdowns of the car for both the user and the service station.

Keywords: Android, mobile application, service station, Bluetooth, OBDII.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СТО – станція технічного обслуговування

БД – база даних

OBD (англ. On-Board Diagnostics) – комп'ютерна діагностика автомобіля

ALDL (англ. Assembly Line Diagnostic Link) – діагностична система автомобілів

ART (англ. Android Runtime) – осередок виконання Android додатків

API (англ. Application Programming Interface) – прикладний програмний інтерфейс

HAL (англ. Hardware Abstraction Layer) – шаблон абстракції обладнання

DEX – (англ. Dalvik Executable) – розширення файлів для ініціалізації і виконання додатків

AOT (англ. Ahead-Of-Time) – завчасна компіляція

JIT (англ. Just-In-Time) – компіляція та компіляція під час роботи програми

GC (англ. Garbage Collection) – оптимізована збірка сміття

NDK (англ. Native Development Kit) – необхідний набір інструментарію для розробки компонентів програмного забезпечення

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки

JDK (англ. Java Development Kit) – комплект розробника на мові Java

JRE (англ. Java RuntimeEnviroment) – виконавча система Java безпеки

DSL (англ. Domain-Specific Language) - предметно-орієнтована мова програмування

XML (англ. Extensible Markup Language) - розширювана мова розмітки

CI (англ. Continuous Integration) – безперервна інтеграція

HTTP (англ. Hyper Text Transfer Protocol) – протокол передачі даних

HTTPS (англ. HyperText Transfer Protocol Secure) - розширення протоколу HTTP для підтримки шифрування з метою підвищення

SDK (англ. Software Development Kit) - набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи

JSON (англ. JavaScript Object Notation) - текстовий формат обміну даними між комп'ютерами

NoSQL (англ. Not Only SQL) - база даних, яка забезпечує механізм зберігання та видобування даних відмінний від підходу таблиць-відношень в реляційних базах даних

CSV (англ. Comma-Separated Values) - файловий формат для представлення табличних даних

ЗМІСТ

					КНТЕУ 121 06з-02.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*19.10.20*	Розробка мобільного додатку взаємодії клієнта з СТО	*Стадія*	*Арку*	*Аркушів*
Керівник		Палагута К.О.		*19.10.20*		*Зміст*	*4*	*43*
Гарант		Криворучко О.В.		*19.10.20*		*Факультет інформаційних технологій 2м курс, 6з група*		
Розробив		Горбуль М.О.		*19.10.20*	*Зміст*			

ВСТУП

Актуальність. Актуальність роботи обумовлена тим, що в даний час інформаційні технології та телекомунікації є елементом отримання конкурентної переваги. Необхідність організації таких систем пов'язана з постійним оновленням потреб бізнесу, а також є природною реакцією на зміну зовнішніх умов.

Кожен автомобіль потребує обслуговування. Регулярний огляд і плановий ремонт допомагають істотно зменшити витрати на утримання авто і значно продовжити йому життя. А найголовніше - убезпечити життя водія, його пасажирів, інших транспортних засобів та пішоходів на дорогах.

Одним з основних понять в області сервісу є поняття послуги. Під послугою автосервісу розуміється комплекс робіт по обслуговуванню автомобіля, спрямованих на задоволення потреб клієнта, що надаються за певну плату. Тобто підприємства автосервісу є клієнтоорієнтованими. Керівництво і співробітники таких підприємств повинні докладати максимум зусиль не тільки для своєчасного і якісного виконання своїх зобов'язань, але і для забезпечення більшої зручності і комфорту для споживачів, що забезпечить залучення нових клієнтів і зведе до мінімуму втрату давніх, постійних клієнтів.

Розробка мобільних додатків є актуальним для підприємств автосервісу. У мобільному додатку зручно вказати перелік послуг і цін, види оплати, адресу та графік роботи, щоб клієнти могли легко знайти компанію. Його дуже легко підтримувати в актуальному стані, і воно охоплює значне коло потенційних клієнтів.

Мета дослідження: розробка мобільного додатка, що містить інструменти взаємодії зі споживачами послуг автосервісу.

Об'єкт дослідження: процес розробки мобільного додатка.

*Предмет дослідження***:** розробка мобільного додатка, який має поліпшити взаємодію співробітників станції технічного обслуговування і клієнтів.

У відповідності з метою дослідження поставлені наступні *завдання*:

- визначення функцій програми;
- вибір програмних засобів для проєктування мобільного додатка;
- розробка структури інтерфейсу;
- розробка додатку.

Методи дослідження. У ході дослідження задля вирішення поставлених завдань було використано принципи системно-комплексного та процесного підходів; методи: аналізу та синтезу, логічного узагальнення, аналогій, порівняльного співставлення; прийоми групування, схематичного зображення даних.

Наукова новизна дослідження. Наукова новизна роботи полягає в дослідженні методів впровадження digital платформи в роботі станції технічного обслуговування.

Практичне значення дослідження. Практична цінність роботи полягає у аналізі методів та засобів для взаємодії клієнта з автосервісом за допомогою мобільного додатка.

РОЗДІЛ 1.
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ВЗАЄМОДІЇ СТО З КЛІЄНТАМИ

1.1. Обслуговування ТЗ

Автомобільний транспорт в усьому світі, і зокрема в Україні, розвивається бурхливими темпами. В даний час виробництво автомобілів в світі неухильно зростає, збільшуючись в середньому на 2 млн на рік. Кожні чотири з п'яти автомобілів загального світового парку - легкові, і на їх частку припадає понад 60% пасажирів, що перевозяться всіма видами транспорту. Рівень автомобілізації в Україні сягає позначки в 232 ТЗ на 1000 чоловік, що налічує майже 10 млн. автомобілів в країні і цей показник постійно зростає. В такій ситуації закономірно зростає попит на технічне обслуговування, адже кожен водій прагне забезпечити безпеку пересування та подовжити термін експлуатації авто.

Технічне обслуговування і ремонт транспортних засобів та їх складових виконують з метою підтримання їх у належному стані та забезпечення встановлених виробником технічних характеристик під час використання, зберігання або утримання протягом періоду експлуатації.

Виконавцями технічного обслуговування і ремонту транспортних засобів є суб'єкти господарювання, які відповідають таким вимогам:

- мають власні або орендовані засоби технічного обслуговування і ремонту, що відповідають установленим законодавством вимогам;

					КНТЕУ 121 06з-02.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		25.06.20	Розробка мобільного додатку взаємодії клієнта з СТО	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Палагута К.О.		25.06.20		*Р1*	*7*	*43*
Гарант		Криворучко О.В.		25.06.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Горбуль. М.О.		25.06.20	*Аналіз існуючих рішень взаємодії сто з клієнтами*			

- роботи з технічного обслуговування і ремонту здійснює персонал необхідного рівня професійної кваліфікації відповідно до видів цих робіт;
- мають виробничі споруди, засоби технічного обслуговування і ремонту, що відповідають встановленим законодавством вимогам.

Вимоги до виконавця технічного обслуговування і ремонту транспортних засобів та надаваних ним послуг (виконуваних робіт) встановлюються технічним регламентом з підтвердження відповідності, затвердженим у встановленому законодавством порядку.

Станція технічного обслуговування (СТО) - підприємство, яке надає послуги населенню та організаціям по плановому технічному обслуговуванню, поточного і капітального ремонтів, усунення поломок, встановлення додаткового обладнання, відновного ремонту автомобілів.

СТО являє собою комплекс споруд та механізмів (підйомники, рихтувальні стенди, шиномонтажний верстат, балансування, стенд розвалу-сходження, установка для заміни масла, промивання паливної системи, рихтувальне і фарбувально-сушильне устаткування, стенди для діагностики електроніки автомобіля), а також ручний і пневматичний інструмент, зібрані в одному місці для комплексного ремонту і обслуговування автомобілів.

1.2. Взаємодія СТО з клієнтом

Для підвищення якості обслуговування автомобіля і спрощення взаємодії СТО з клієнтом необхідно розробити мобільний додаток, який буде містити інструменти маркетингової взаємодії з користувачами послуг автосервісу. Такий додаток має містити наступний функціонал:

- Авторизація;

- Інформація про транспортний засіб;
- Сервісна книжка;
- Запис на сервіс;
- Статус ремонту;
- Швидка комп’ютерна діагностика;
- Контакти.

Із переліченого функціоналу мобільного додатку найбільшим важелем є комп’ютерна діагностика, яка дає можливість клієнту автосервісу продіагностувати свій автомобіль на наявність помилок та несправностей віддалено, без відвідування станції. При цьому результати діагностики будуть оброблені кваліфікованими співробітниками СТО і клієнт, відповідно, отримає доступну інформацію про технічний стан авто.

Комп’ютерна діагностика автомобіля (OBD, on-board diagnostics) - це діагностика різноманітних систем автомобіля, яка проводиться блоком керування автомобіля. Результати діагностики відображаються для користувача автомобіля, наприклад на панелі приладів, а також використовуються автомеханіками і діагностами. Системи OBD впроваджуються з 1980-х років, OBDII - з 1996 року. Сучасні варіанти використовують стандартизовані цифрові порти для надання поточних даних і видачі ряду стандартних кодів проблем (DTC, diagnostic trouble code).

Стандарти інтерфейсів:

1. ALDL (Assembly Line Diagnostic Link) - діагностична система автомобілів, розроблена компанією General Motors, яка є попередником стандарту OBD-I. Ця система не являла собою чіткий стандарт і через це була введена як специфікація забезпечення зв’язку з транспортним засобом. Існує три різні роз’єми ALDL: 5-контактний роз'єм, 10- контактний і 12-контакний, - останній має широке поширення на автомобілях GM.

Більш ранні версії використовували швидкість передачі 160 біт / с, в той час як пізніші - 8192 біт / с і використовували двосторонній зв'язок з Power-train Control Module (PCM).

2. OBD-I (On-Board Diagnostic Link) - бортова діагностика, що спонукала автовиробників розробляти надійні системи контролю за викидами (Emission control system).
3. OBD 1.5 - часткова реалізація OBD-II, яку General Motors використовував на деяких автомобілях в 1994 і 1995 роках (General Motors не використовували термін OBD 1.5 в документації на ці автомобілі, вони просто називалися OBD і OBD-II секції в інструкції по експлуатації).
4. OBD-II - бортова діагностика, стандарт якої був спроєктований у середині 90-х років, що дозволяє отримувати повний контроль над двигуном автомобіля. Дозволяє здійснювати діагностування частин кузова і додаткових пристроїв, а також проводити моніторинг мережі управління транспортним засобом. В даному інтерфейсі виробники використовують такі протоколи для з'єднання з транспортним засобом:
    - ISO 9141-2;
    - ISO 14230 Keyword Protocol 2000;
    - SAE J1850 VPW;
    - SAE J1850 PWM;
    - ISO 15765-4 CAN (Controller Area Network).
5. EOBD - європейська бортова діагностична система, заснована на специфікації OBD-II. Ця система була введена під час розробки вимог моніторингу та скорочення викидів від автомобілів EURO 3, відповідно до «Directive 98/69 / EC of the European Parliament» від 13.10.1998 р.

6. EOBD2 - Термін EOBD2 є маркетинговим терміном, який використовувався деякими виробниками транспортних засобів щоб звернути увагу на наявність специфічної функції від виробника, яка фактично не є частиною OBD або EOBD стандарту. В даному випадку “E” розшифровується як “Розширений” (Enhanced).
7. JOBD (Japan On-Board Diagnostic) - версія OBD-II для автомобілів, які були продані у Японії.

1.3. Аналіз існуючих додатків для діагностики і обслуговування автомобіля

В Україні не розповсюджені додатки від автосервісів з широким спектром функціоналу та послуг, тому для аналізу було розглянуто три найпопулярніші додатки в Android Play Store для компютерні діагностики: Torque, CarScanner, OBD Auto Doctor.

Torque

1. Графічний інтерфейс. Додаток має застарілий дизайн, який не відповідає теперішнім дизайнерський рішенням та вимогам Material Design від Google.
2. Користувацький досвід. Важкий в освоєнні інтерфейс. Іконки з параметрами можна кастомізувати, але дуже не зрозуміло як і де це робити. Новому користувачеві важко розібратися з елементами керування.
3. Функціональні можливості. Додаток вміє робити базові речі: зчитувати коди помилок та робити логування параметрів системи.

Швидкодія. Додаток має задовільну швидкодію. Під час користування додатком ніяких нарікань не виявлено.

Рис. 1.1. Скріншот Torque

Рис. 1.2. Скріншот Torque

Car Scanner

Графічний інтерфейс. Додаток має приємний, лаконічний дизайн, який відповідає вимогам Material Design. До головних функцій є легкий доступ на головному екрані у вигляді кнопок з іконками і надписами.

1. Користувацький досвід. Всі основні функції винесені на головний екран, що є зручним у користуванні і легким в освоєнні для нових користувачів.
2. Функціональні можливості. Додаток вміє зчитувати коди помилок та робити логування параметрів системи та показвати їх у зручному для аналізу вигляді (графіки, спідометри і таблиці). Також, додаток може читати коди помилок.
3. Швидкодія. Додаток має задовільну швидкодію. Під час користування додатком ніяких нарікань не виявлено.

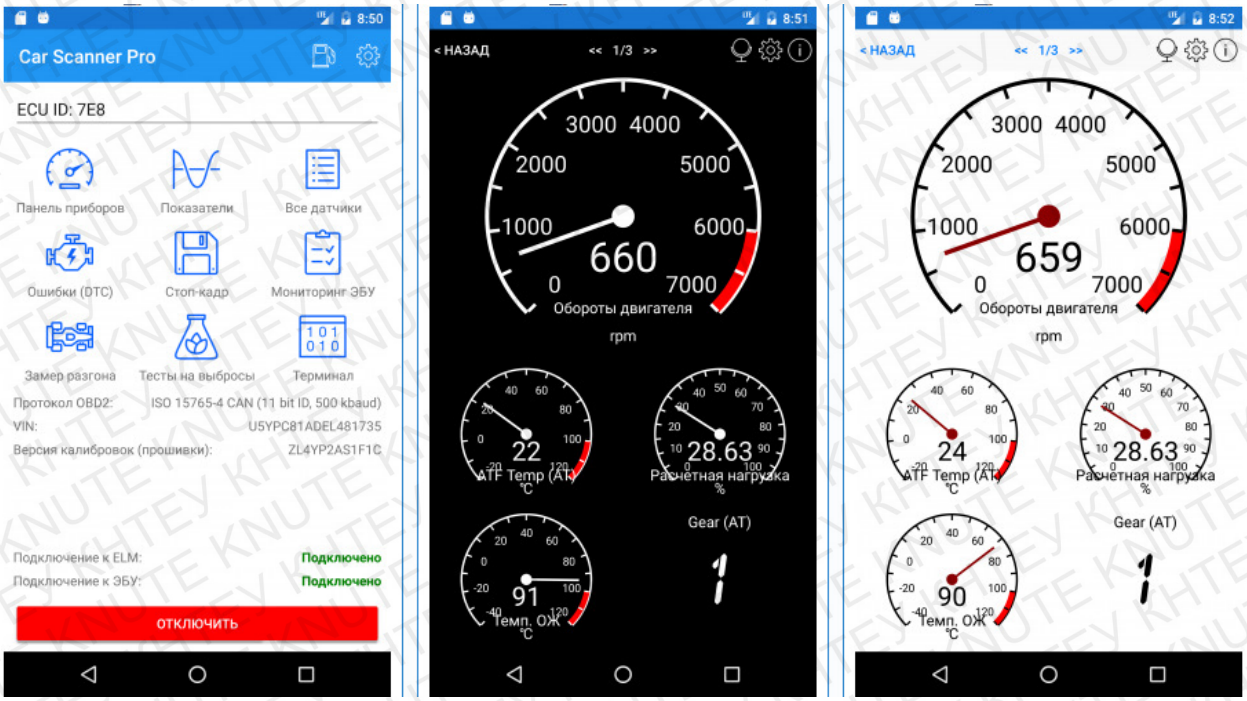

Рис. 1.3. Скріншот Car Scanner

OBD Auto Doctor

1. Графічний інтерфейс. Додаток відповідає вимогам Material Design від Google і має приємний користувацький інтерфейс.
2. Користувацький досвід. Додаток має головне меню внизу екрану, що є дуже зручним рішенням. Але, спершу не зрозуміло як користуватися додатком, адже багато елементів не мають підпису про те, яку саме дію вони виконують.

3. Функціональні можливості. Додаток вміє зчитувати основні параметри, а також читати та скидати помилки.

Швидкодія. Додаток має задовільну швидкодію. Під час користування додатком ніяких нарікань не виявлено.

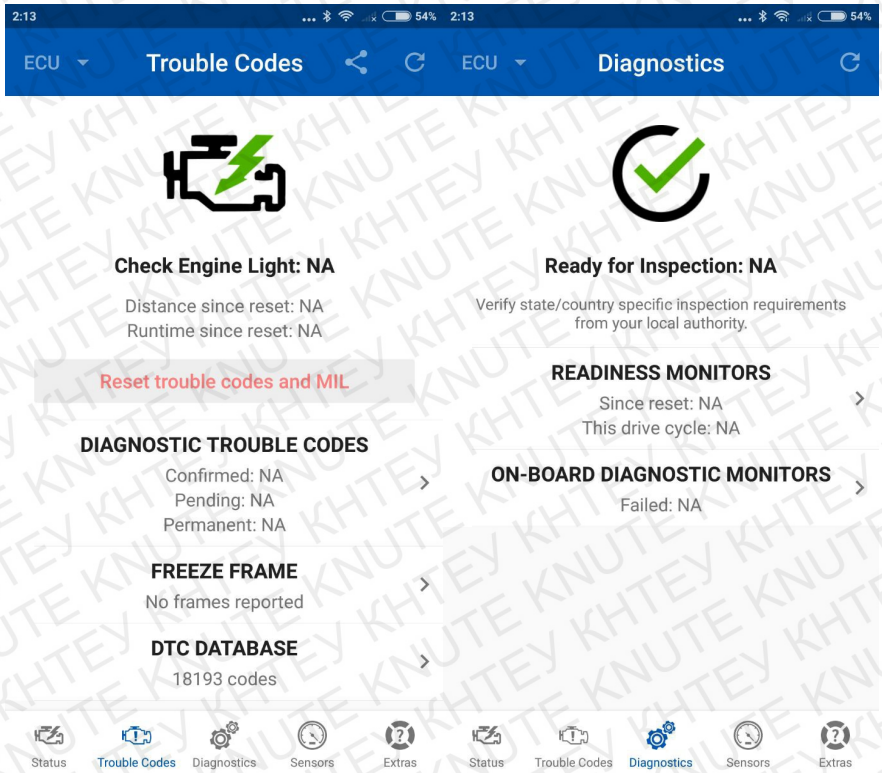

Рис. 1.4. OBD Auto Doctor

1.4. Висновки до розділу 1

Комп'ютерна діагностика є важливим елементом первинної діагностики автомобіля в світлі сучасної діджиталізації, що дає змогу економити час клієнта та фінансові витрати. А можливість дистанційно обробляти діагностичні данні кваліфікованим спеціалістом і надавати повну та доступну інформацію про стан авто клієнту відображає значну перевагу розробленого додатка перед готовими мобільними програмами діагностики.

Серед існуючих рішень найкращим додатком по сукупності всіх параметрів виявився "Car Scanner", адже він надає великий функціонал у

поєднанні з красивим і логічним дизайном. За допомогою “Car Scanner” можна зчитувати коди помилок, проводити моніторинг основних параметрів у реальному часі а також робити їх логування. Але, все ж таки, основним недоліком всіх розглянутих додатків є те, що користувач без досвіду в автомобільній механіці не матиме достатньо знань для діагностики автомобіля без допомоги спеціаліста.

РОЗДІЛ 2.
ОСОБЛИВОСТІ РОЗРОБКИ ANDROID ДОДАТКУ

2.1. Архітектура Android платформи

Android є системою з відкритим кодом, що побудована на основі програмного стеку системи Linux для підтримки широкого спектру пристроїв різних форм-факторів.

Linux ядро

Основою платформи Android є ядро Linux. Наприклад, Android Runtime (ART) спирається на ядро Linux для основних функціональних можливостей, таких як потоки та управління низьким рівнем пам'яті.

Використання ядра Linux дає змогу Android використовувати переваги ключових функцій безпеки і дозволяє виробникам пристроїв розробляти драйвери апаратного забезпечення для вже добре відомого ядра.

Шаблон абстракції обладнання (HAL)

Шаблон абстракції обладнання (Hardware Abstraction Layer, HAL) надає стандартні інтерфейси, які розширюють апаратні можливості пристрою до більш високого рівня Java API. HAL складається з декількох модулів, кожен з яких реалізує інтерфейс для конкретного типу апаратного компонента, такого як камера або модуль Bluetooth. Коли API платформа здійснює виклик для доступу до апаратного забезпечення пристрою, система Android завантажує модуль бібліотеки для цього компонента обладнання.

Android Runtime

Для пристроїв, що працюють під Android версією 5.0 (рівень API 21 або вище), кожна програма працює у своєму власному процесі та з власним екземпляром Android Runtime (ART).

ART написаний для запуску декількох віртуальних машин на пристроях з низькою пам'яттю, виконуючи файли DEX, формат байт-коду, розроблений спеціально для Android, оптимізований для мінімального споживання пам'яті. Збірка інструментів, наприклад, Jack, компілює джерела Java в байт-код DEX, який може працювати на платформі Android.

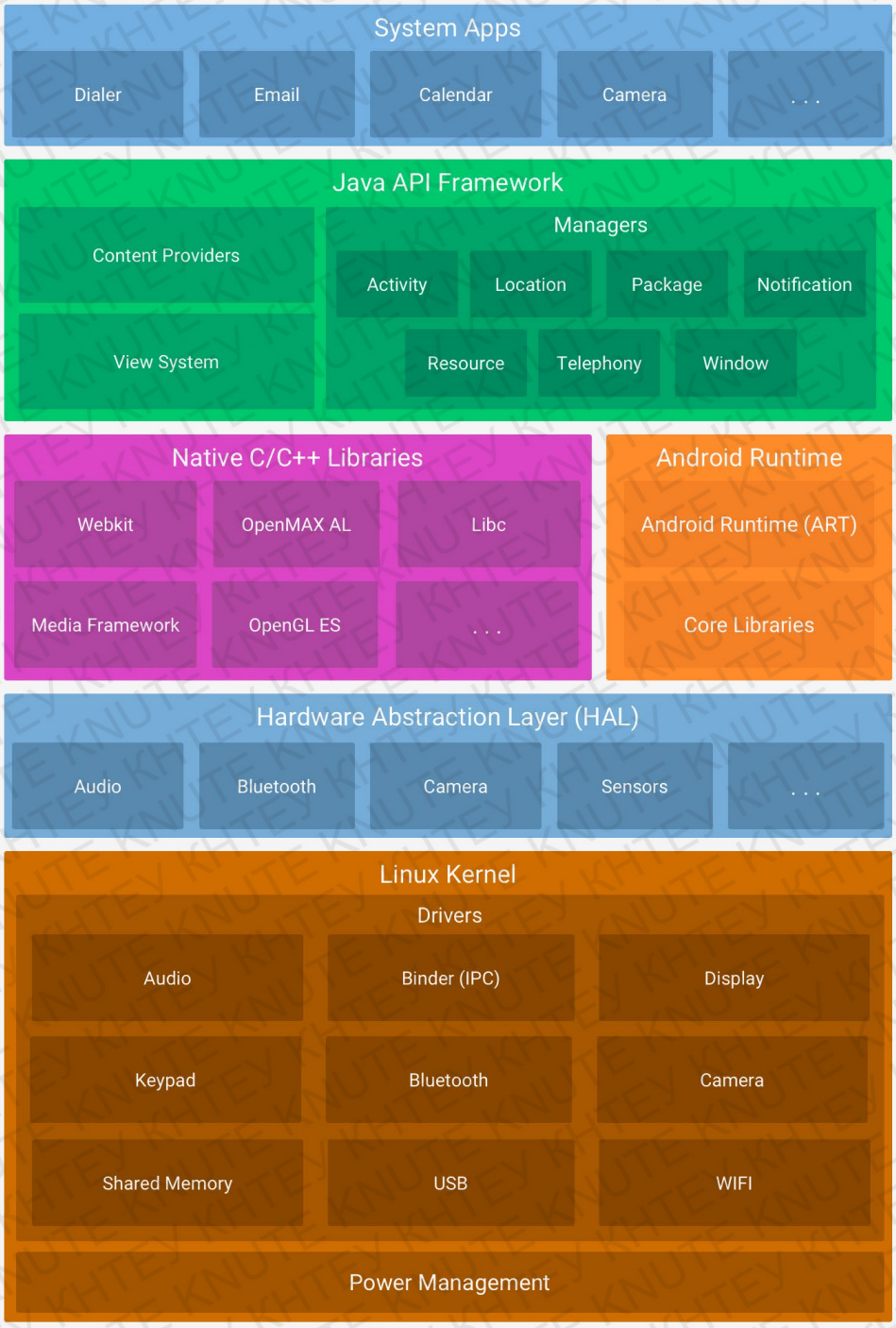

Рис. 2.1. Структура Android системи

Деякі з ключових можливостей ART включають наступне:

- завчасна (Ahead-of-time, AOT) компіляція та компіляція під час роботи програми (just-in-time, JIT);
- оптимізована збірка сміття (garbage collection, GC);
- на Android 9 (рівень API 28) і вище, перетворення файлів у форматі Dalvik Executable (DEX) пакета програм на більш компактний машинний код;
- підтримка кращого налагодження (анг. debug), включаючи спеціальний профайлер вибірки, докладні діагностичні виключення та звіти про аварійне завершення роботи, а також можливість встановити точки спостереження для моніторингу конкретних полів.

До версії Android 5.0 (рівень API 21), Android був оснований на віртуальній машині Dalvik. Якщо програма працює добре на ART, то вона буде працювати і на Dalvik, але зворотне не стверджується.

Android також включає в себе набір основних бібліотек, які забезпечують більшу частину функціональності мови програмування Java, включаючи деякі мовні особливості Java 8, які використовує Java API.

Нативні (Native) C/C++ бібліотеки

Багато основних системних компонентів і служб Android, таких як ART і HAL, побудовані з коду, який вимагає нативних бібліотек, написаних на мовах C і C++. Платформа Android надає API Java-платформи для розкриття функціональних можливостей деяких з цих нативних бібліотек для програм. Наприклад, можна отримати доступ до OpenGL ES за допомогою API OpenGL для платформи Android, щоб додати підтримку для малювання та маніпулювання 2D і 3D графікою у додатку.

Якщо розробляється програма, яка потребує коду C або C++, можна скористатися NDK для Android, щоб отримати доступ до деяких з цих

бібліотек платформи безпосередньо.

Java API Framework

Весь набір функцій ОС Android доступний через API, написане на мові Java. Ці API формують “будівельні” блоки, необхідні для створення додатків Android, спрощуючи повторне використання основних компонентів і служб модульної системи, які включають наступне:

- велику та розширювану систему перегляду (View System), яку можна використовувати для створення користувацького інтерфейсу програми, включаючи списки, сітки, текстові поля, кнопки та навіть вбудований веб-переглядач;
- менеджер ресурсів (Resource Manager), що надає доступ до ресурсів системи, таких як локалізовані рядки, графічні файли та файли компонування;
- менеджер сповіщень (Notification Manager), який дозволяє програмам відображати спеціальні сповіщення в рядку стану;
- менеджер активності (Activity Manager), який керує життєвим циклом програм і надає загальний стек навігації;
- постачальники вмісту (Content Providers), які дозволяють програмам отримувати доступ до даних з інших програм, наприклад, програми «Контакти», або обмінюватися власними даними.

Розробники мають повний доступ до тих самих API інтерфейсів, які використовують системні програми для Android.

Системні додатки

Android поставляється з набором основних програм для електронної пошти, SMS-повідомлень, календарів, веб-перегляду, контактів тощо. Програми, включені до платформи, не мають спеціального статусу серед програм, які користувач вибирає для встановлення. Таким чином, додаток

стороннього виробника може стати користувальницьким веб-браузером за промовчанням, SMS-переглядачем або навіть клавіатурою за замовчуванням (але, все ж, є деякі винятки, такі як додаток налаштувань системи).

Системні програми функціонують як програми для користувачів, а також надають ключові можливості, до яких розробники можуть отримати доступ з власних програм. Наприклад, якщо програма хоче надіслати SMS-повідомлення, то не потрібно самостійно створювати цю функцію, а, натомість, можна скористатися вже встановленим SMS додатком для відправки за вказаною адресою.

2.2. Середовище розробки Android Studio

Додаток для взаємодії станції технічного обслуговування з клієнтами було розроблено на мові програмування Java у середовіщі розробки Android Studio.

Android Studio – інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

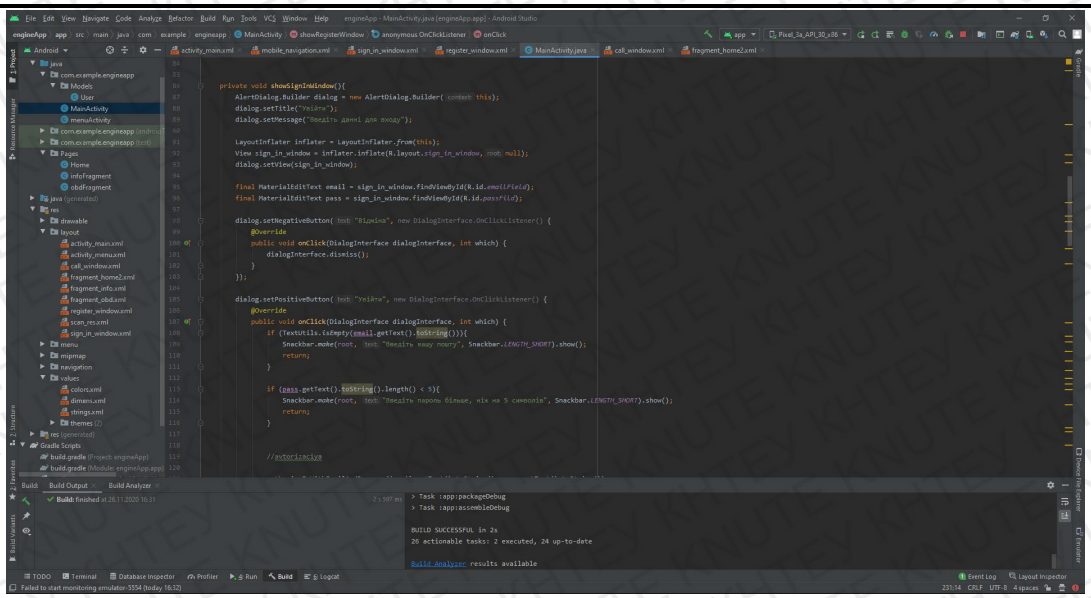

Рис. 2.2. Інтерфейс Android Studio

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

У Android Studio передбачені такі функції:

- Живі макети (layout): редагувальник WYSIWYG — живе кодування — подання (rendering) програми в реальному часі.
- Консоль розробника: підказки по оптимізації, допомога по перекладу, стеження за напрямком, агітації та акції — метрики Google аналітики.
- Базування на Gradle.
- Android-орієнтований рефакторинг та швидкі виправлення.
- Lint утиліти для охоплення продуктивності, юзабіліті, сумісності версій та інших проблем.
- Використання можливостей ProGuard та підписів до програм.
- Багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag-and-drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів.

Розробка реалізована на мові програмування Java за допомогою JDK (англ.Java Development Kit) - комплекту розробника на мові Java, який включає компілятор Java, стандартні бібліотеки, документацію, утиліти і виконавчу систему Java (англ. JRE - Java RuntimeEnviroment).

2.3. Система автоматичної сбірки Gradle

У проєкті вікористана система Gradle. Це система автоматичного складання, що надає предметноорієнтовану мову (DSL) на мові Groovy замість традиційної XMLподібної форми подання конфігурації проекту. Вона також надає можливості управління залежностями.

Система управління залежностями є простим способом завантаження сторонніх бібліотек в проект, без зберігання бібліотек у вихідному проекті.

Система управління залежностями заснована на файлі, який визначає імена і версії бібліотек, і які необхідно включити в додаток. Додавання, видалення та зміна версії сторонніх бібліотек виконується так само просто, як і зміна декількох рядків у файлі конфігурації. Система управління залежностями буде завантажувати зазначені бібліотеки з центрального сховища та зберігати їх в директорії за межами вашого проекту.

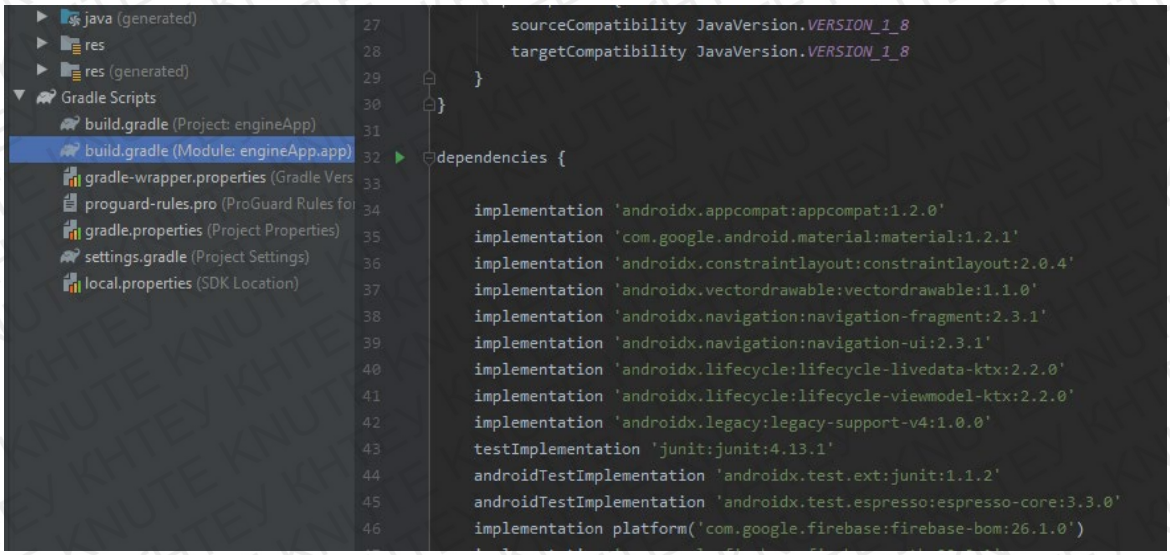

Рис. 2.3. Файл конфігурації збірки

Система збирання допомагає зі складанням і упаковкою додатки без прив'язки до певного середовища розробки. Це особливо корисно, якщо використовується сервер для складання або безперервної інтеграції (CI), де середовища розробки не завжди доступні. Замість цього сервер збірки може викликати систему збирання, надаючи конфігурації для збірки так, що система знає як зібрати додаток для різних платформ.

У випадку з Gradle, управління залежностями і система збирання йдуть разом. Вони налаштовані однаковим набором файлів. Gradle використовує спрямований ациклічний граф для визначення порядку виконання завдань.

Gradle був розроблений для розширюваних багатопроєктних збірок, і підтримує інкрементальні збірки, визначаючи, які компоненти дерева збірки не змінилися і які завдання, що залежні від цих частин, не вимагають перезапуску.

2.4. Хмарна база даних Firebase

База даних в проєкті реалізована за допомогою Firebase. Firebase – це платформа для розробки мобільних та веб-додатків, розроблена FirebaseInc у 2011 році, потім придбана Google у 2014 році.

Більшість баз даних вимагають здійснення HTTP-дзвінків для отримання та синхронізації даних. Більшість баз даних надають дані лише тоді, коли виконано запит. Додаток Firebase не підключається через звичайний HTTP. Підключення відбувається через WebSocket. WebSockets набагато швидше ніж HTTP. Усі дані синхронізуються автоматично через один WebSocket так швидко, як мережа клієнта може переносити його. Firebase надсилає нові дані, як тільки вони оновлюються. Коли клієнт зберігає зміну даних, усі підключені клієнти отримують оновлені дані майже миттєво.

Firebase Storage – це потужна, проста та економічно вигідна послуга зберігання об’єктів, створена для масштабів Google. Пакет FirebaseSDK для хмарного збереження додає правила безпеки Google для завантаження програм Firebase, незалежно від якості мережі. Можна використовувати SDK для збереження зображень, аудіо, відео чи іншого вмісту, створеного користувачем.

На сервері є можливість використовувати Google Cloud Storage для доступу до файлів. Розробники використовують пакети SDK Firebase для Cloud Storage для завантаження та вивантаження файлів безпосередньо від клієнтів. Якщо мережевий зв’язок поганий, клієнт може повторити операцію прямо там, де він припинився, економлячи час та пропускну здатність користувачів.

Хмарне збереження зберігає файли для зберігання в хмарі Google, роблячи їх доступними через Firebase та Google Cloud. Це дозволяє гнучко завантажувати файли з мобільних клієнтів через пакети SDK Firebase, а

також виконувати обробку на стороні сервера, наприклад, фільтрування зображень або перекодування відео за допомогою Google Cloud Platform. Хмарне збереження масштабується автоматично, це означає, що немає необхідності переходити до будь-якого іншого постачальника.

SDK-файли Firebase для хмарного збереження легко інтегруються з аутентифікацією Firebase для ідентифікації користувачів. Також, можливе використання декларативної мови безпеки, яка дозволяє встановлювати контроль доступу для окремих файлів або груп файлів, щоб могли встановлювати файли як з публічним доступом, так і з приватним. Більшість додатків повинні знати особу користувача. Знання ідентичності користувача дозволяє додатку надійно зберігати дані користувачів у хмарі та надавати однаковий персоналізований досвід на всіх пристроях.

Перевірка автентичності Firebase надає сервіси, прості у використанні SDK та готові бібліотеки інтерфейсу для автентифікації користувачів в програмі. Він підтримує автентифікацію за допомогою паролів, номерів телефонів, популярних постачальників ідентифікаційних даних, таких як Google, Facebook та Twitter тощо. Автентифікація Firebase тісно інтегрується з іншими службами Firebase і використовує такі галузеві стандарти, як OAuth 2.0 та OpenID Connect, тому вона може бути легко інтегрована в окремий сервер. Щоб увійти в додаток користувача, спочатку отримуються облікові дані автентифікації від користувача. Ці облікові дані можуть бути електронною адресою користувача та паролем користувача або маркером OAuth від постачальника ідентифікаційних даних. Потім, ці повноваження передаються в SDK для автентифікації Firebase. Потім, сервіси Firebase перевірять ці дані та повернуть відповідь клієнту. Після успішного входу, відкривається доступ до основної інформації користувача.

Також, можна контролювати доступ користувача до даних, що зберігаються в інших продуктах Firebase. Є можливість використовувати

наданий маркер автентифікації для перевірки особи користувачів у сервісах сервера. За замовчуванням, автентифіковані користувачі можуть читати та записувати дані в базу даних Firebase Realtime та GoogleCloud. Можете контролювати доступ цих користувачів, модифікувавши бази даних та правила безпеки хмарного зберігання.

Cloudfunction для Firebase дозволяють автоматично запускати резервний код у відповідь на події, викликані функціями Firebase та HTTPS-запитами. Код зберігається в хмарі Google і працює в керованому середовищі. Не потрібно керувати та масштабувати сервери. Після написання та розгортання функцій сервера, Google починають керувати функцією. Є можливість запустити функціонування за допомогою HTTP-запису, або, використовуючи фонові функції, сервіси Google проводять моніторинг програми та використовують цю функцію, коли вона запущена. Коли завантаження зменшується – Google реагує, швидко зменшуючи кількість екземплярів віртуального сервера, необхідних для запуску функції. Кожна функція функціонує у власному середовищі зі своєю власною конфігурацією.

Realtime Databaseє хмарною базою даних типу NoSQL. Дані синхронізуються для всіх клієнтів у режимі реального часу та залишаються доступними, коли програма переходить у режим офлайн. Дані зберігаються в форматі JSON і синхронізуються в режимі реального часу з кожним підключеним клієнтом. При створенні крос-платформової програми з пакетами SDK для iOS, Android або JavaScript, всі клієнти діляться одним екземпляром бази даних і автоматично отримують оновлення з найновішими даними.

База даних Firebase Realtime дозволяє створювати спільні програми, дозволяючи безпечний доступ до бази даних безпосередньо з коду клієнта. Дані зберігаються на локальному рівні, і навіть в режимі офлайн події в режимі реального часу продовжують працювати, надаючи кінцевому

користувачеві результат. Коли пристрій відновлює з'єднання, база даних синхронізує зміни локальних даних із віддаленими оновленнями, які відбулися під час відключення клієнта, автоматично ліквідуючи будь-які конфлікти.

Firebase Realtime забезпечує гнучку мову, засновану на виразах, правилах, щоб визначити, як мають бути структуровані дані та коли дані можуть зчитуватись або бути збереженими в сховищі. При інтеграції з Firebase Authentication розробники можуть визначити, хто має доступ до даних, і як вони можуть отримати доступ до них. База даних у реальному часі – це база даних NoSQL і як така має різні можливості для оптимізації та функціональні можливості порівняно з реляційною базою даних.

Firebase Realtime API розроблений так, щоб дозволяти виконувати операції, які можна швидко виконати. Це дає змогу створити зручну роботу в режимі реального часу, яка може обслуговувати мільйони користувачів без шкоди для роботи та даних.

2.5. Висновки до розділу 2

Отже, орієнтованість на розробку додатку для платформи Android стала основною причиною вибору Android Studio як основного середовища розробки. Це, легке в опануванні, середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків. У тому числі включені у середовище засоби для спрощення тестування програм і автоматизації збірки полегшують роботу розробника і значно скорочують витрати часу на створення додатку. Використання автоматизації збірки має характерні переваги, такі як:

- Поліпшення якості продукту;
- Прискорення процесу компіляції;
- Позбавлення зайвих дій;

- Мінімізація «поганих» збірок;
- Ведення історії збірок для розбору;
- Економія часу.

Використання Firebase також сприяє скороченню використання часу на розробку, адже не потрібно відволікатися на створення БД, написання API прийому та отримання даних. Уся серверна частина виконується сервісом Firebase майже без втручання.

РОЗДІЛ 3.
ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ANDROID ДОДАТКУ

3.1. Інтерфейс програмного модулю

При запуску додатку відображається стартовій екран з назвою та кнопками для наступної взаємодії. Після запуску відбувається перевірка на те, чи був користувач раніше аутентифікований. Якщо так, то додаток дістає зі сховища збережений унікальний ідентифікатор користувача та надсилає його на сервер для аутентифікації. Якщо ж користувач раніше не був аутентифікований, або була стерта вся інформація додатку то показується вікно для вводу особистого унікального ідентифікатора користувача та його паролю.

Рис. 3.1. Стартовий екран додатку

					КНТЕУ 121 06з-02.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		19.10.20	Розробка мобільного додатку взаємодії клієнта з СТО	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Палагута К.О.		19.10.20		*РЗ*	*29*	*43*
Гарант		Криворучко О.В.		19.10.20		*Факультет інформаційних технологій 2м курс, 6з група*		
Розробив		Горбуль М.О.		19.10.20	*Особливості реалізації android додатку*			

Для того, щоб аутентифікуватися, користувач повинен натиснути кнопку «УВІЙТИ» та ввести свій унікальний email та пароль до свого облікового запису. Якщо одне з полів буде порожнім, то аутентифікація не почнеться, а користувачеві буде показано повідомлення про помилку.

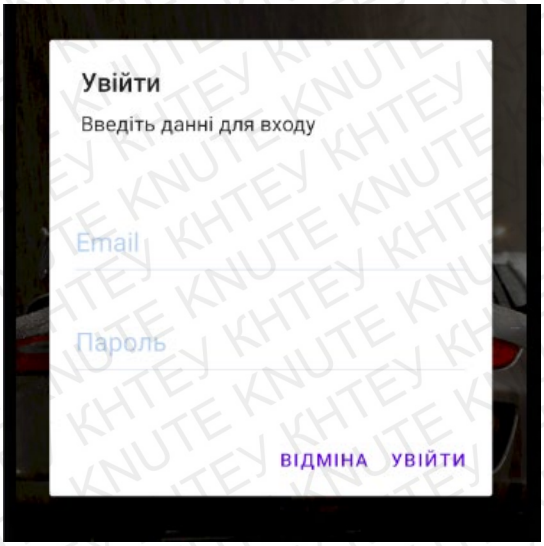

Рис. 3.2. Форма входу

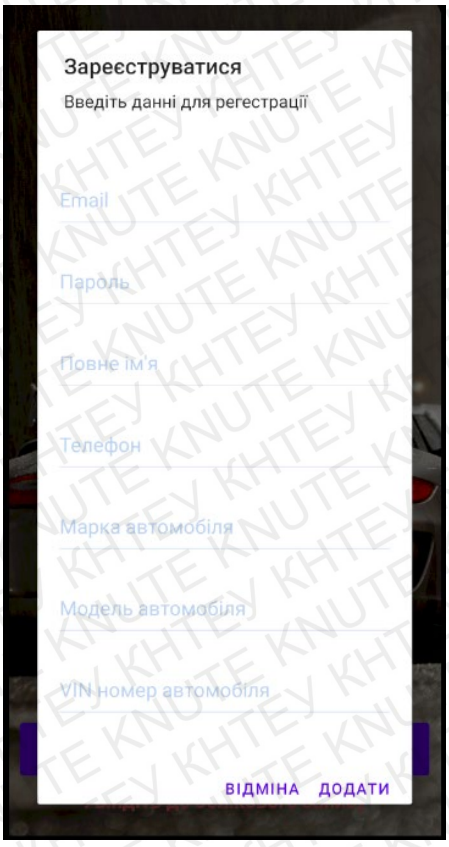

Рис. 3.3. Форма реєстрації

Якщо користувач ще не зареєстрований у системі, потрібно натиснути на кнопку «ЗАРЕЄСТРУВАТИСЯ», яка покаже вікно для реєстрації користувача. Для того, щоб користувач зареєструвався, він повинен ввести наступні дані:

- Email;
- Номер телефону;
- Ім’я;
- Пароль;
- Марка авто;
- Модель авто;
- VIN номер авто.

Якщо якесь поле буде порожнім, користувач отримає повідомлення про помилку. Поле паролю не буде показуватися як звичайний текст, а буде замінено на крапки, щоб ніхто не підглянув пароль користувача.

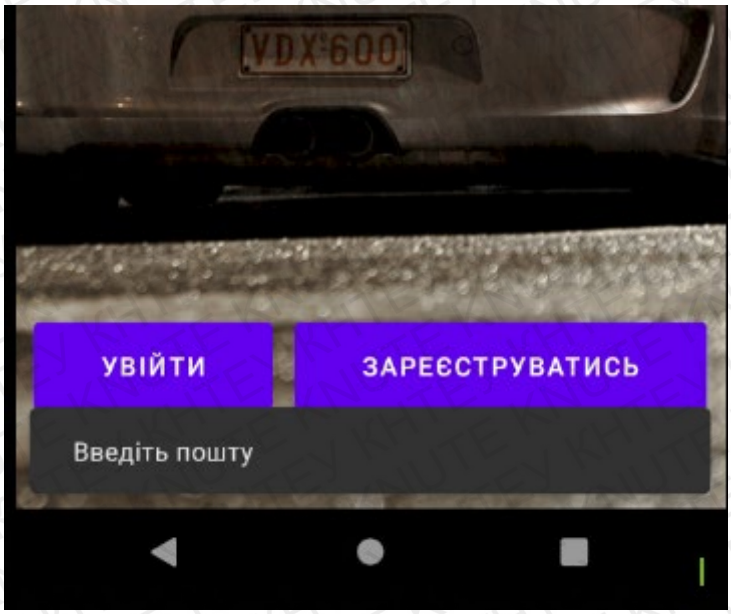

Рис. 3.4. Помилка заповнення поля пошти

Коли користувач увів всі потрібні дані, він повинен натиснути на кнопку «УВІЙТИ», якщо користувач вже був зареєстрований. На даному етапі встановлюється з’єднання з сервером. Після того, як з’єднання з

сервером встановлене і аутентифікація пройшла успішно буде показано основне меню додатку.

В основному меню знаходиться нижнє меню, яке відповідає за перемикання основного контенту на головному екрані. Для цього реалізовано контейнер, вміст якого може динамічно змінюватися під час роботи додатку. Таким чином, користувач може перемикати вміст головного екрану, не втрачаючи історію роботи з цим екраном. Тобто, якщо користувач щось робив у вікні 1, переключився на вікно 2, а потім повернувся назад на 1 вікно, то стан вікна 1 буде той, який був на момент, коли користувач залишив це вікно.

Головне меню налічує такі пункти як:

- Мої данні;
- OBD сканер;
- Контакти СТО.

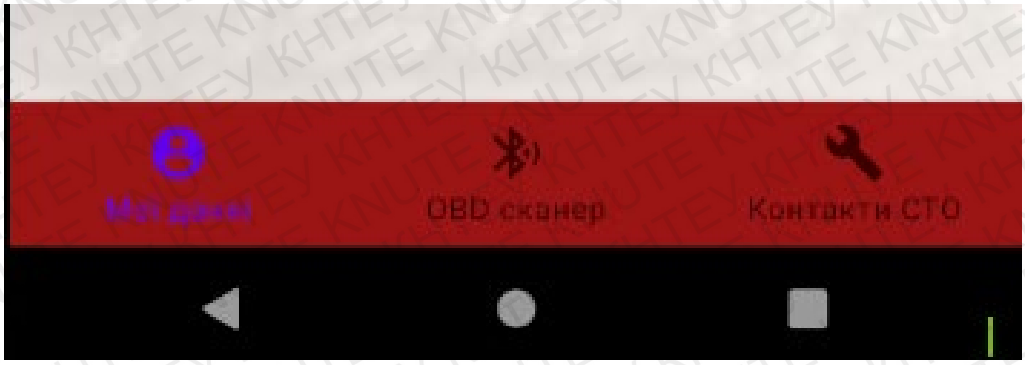

Рис. 3.5. Пункти меню

Перший пункт меню «Мої данні» відображає всю актуальну інформацію про автомобіль та поточні і минулі роботи, що виконувались з автомобілем. Інформація про роботи синхронізується з базою даних. При виконанні робіт інформація додається до БД співробітниками станції і відображається в додатку в таблиці під назвою «Роботи і витрати». Якщо роботи не проводились то відображається сповіщення «Інформація відсутня». Також є кнопка «РЕЗУЛЬТАТИ СКАНУВАННЯ». Вона викликає

вікно, яке відображає інформацію про результати сканування OBD сканеру, оброблені кваліфікованим спеціалістом станції. Після відправлення результатів сканування на перевірку біля кнопки «РЕЗУЛЬТАТИ СКАНУВАННЯ» з'являється позначка про повідомлення.

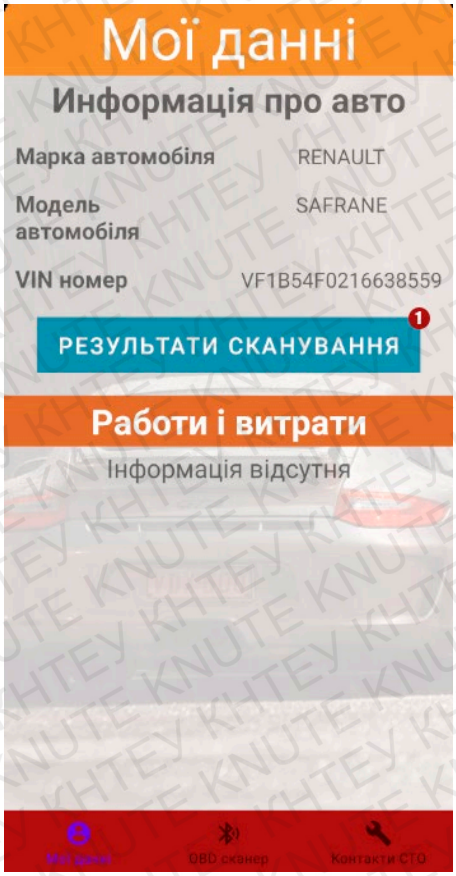

Рис. 3.6. Пункт меню «Мої данні»

Другий пункт меню надає можливості комп'ютерної діагностики автомобіля в реальному часі для відображення основних параметрів і читання помилок, який називається «OBD сканер». При переході у «OBD сканер» відкриється екран пошуку Bluetooth пристроїв, що є в зоні видимості. Після вибору пристрою запускається сканування автомобіля і відображення даних у вигляді таблиці.

Дані зберігаються у форматі CSV у зовнішнє сховище додатку, тому їх можна вільно переглядати чи редагувати за межами додатку. Також ці данні, за бажанням користувача, відправляються для обробки отриманої інформації кваліфікованим співробітником станції кнопкою «Перевірити результати».

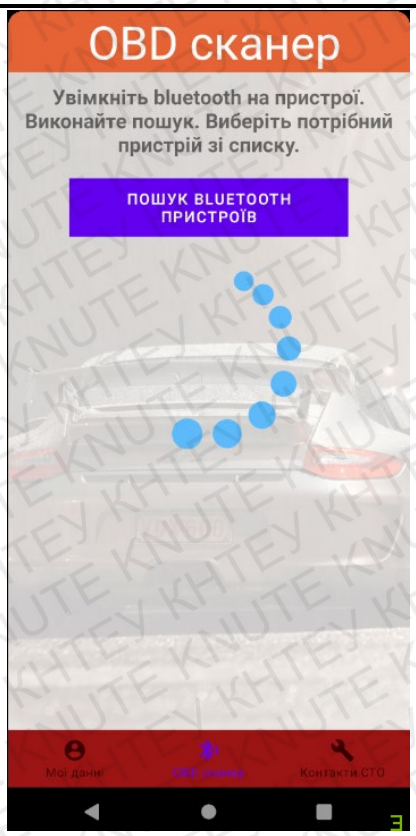

Рис. 3.7. Пошук пристроїв

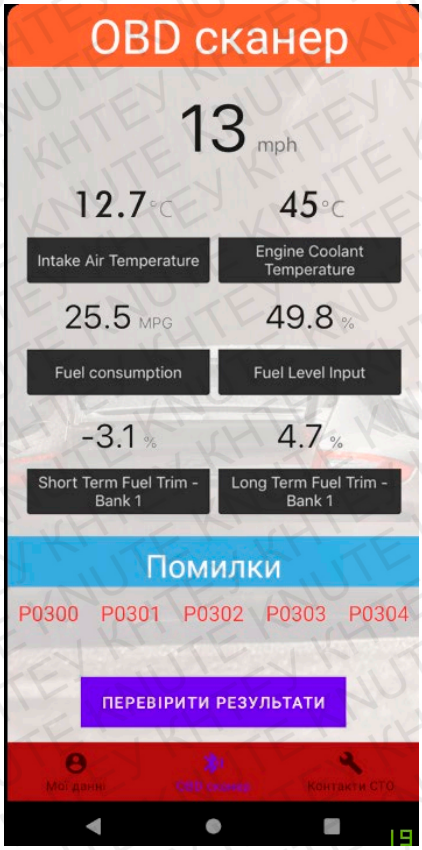

Рис. 3.8. Данні комп’ютерного сканування

Останній пункт меню «Інформація про СТО» налічує контактну інформацію станції технічного обслуговування. Там вказано адресу сервісу, контактні номери та час роботи. Також присутня кнопка «ЗАПИС НА ДІАГНОСТИКУ» яка дозволяє зробити запис. При натисканні відображається форма, що пропонує вибрати дату, час запису і поле, де можна вказати причину записи, а саме орієнтовну несправність.

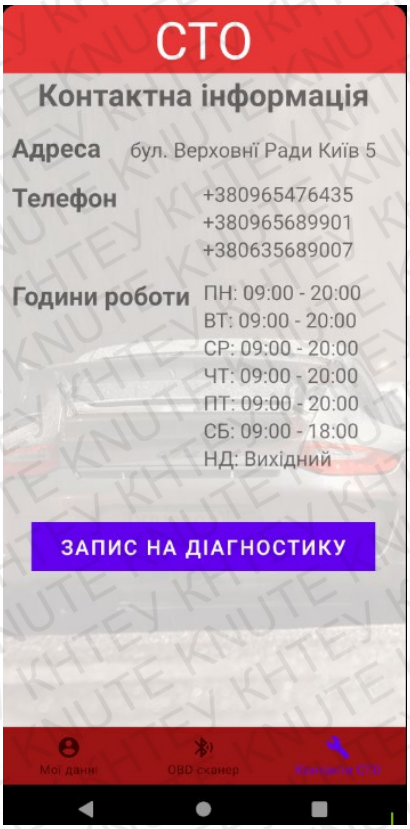

Рис. 3.9. Пункт меню «Контакти СТО»

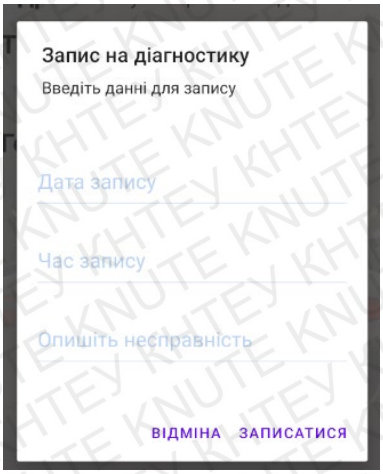

Рис. 3.10. Форма запису на діагностику

3.2. Архітектура

Додаток складається з декількох Android модулів, кожен з яких має своє призначення:

- App;
- Obd;
- Bluetooth.

Основний модуль додатку – «app». Цей модуль містить користувацьку логіку додатку, весь графічний інтерфейс, і зв'язує між собою інші модулі. Основою графічного інтерфейсу в системі Android є Activity (операція).

Activity – це компонент, з яким користувач може взаємодіяти для того, щоб виконати ту чи іншу дію, як наприклад, набрати номер телефону чи зробити фотографію. Для кожної Activity надається вікно для відображення відповідного користувацького інтерфейсу. Зазвичай, вікно відображається на весь екран, проте, бувають випадки коли вікна поділяють екран між собою. Додаток складається з багатьох Activity, кожна з яких грає окрему роль. У додатку для Android оов'язково повинна існувати головна Activity, яка показується одразу після запуску додатку. В свою чергу, операція може викликати інші операції для виконання різних дій. Кожен раз, коли викликається нова операція, попередня зупиняється, але не зникає, система зберігає її в стеку ("стек переходів назад"). Стек переходів назад працює за принципом "останнім зайшов – першим вийшов". Тому, коли користувач закінчує поточну операцію, поточна операція видаляється зі стеку, і відновлюється попередня операція, яка була на вершині стеку.

Не менш важливою складовою графічного інтерфейсу Android є фрагменти (Fragment). Фрагмент являє собою частину користувацького інтерфейсу в операції. Вони створені для того, щоб декілька фрагментів компонувати в одну операцію, для побудови багатофункціонального користувацького інтерфейсу з можливістю перевикористання різних

фрагментів. Таким чином, фрагмент можна розглядати як частину операції. Кожен фрагмент має свій окремий життєвий цикл і самостійно обробляє дії користувача. Ще одною важливою особливістю фрагментів є те, що їх можна додавати і видаляти прямо під час роботи програми.

У розробленому додатку основною операцією є користувацьке меню. Тобто, головна операція містить у собі інтерфейс меню, а також контейнер для основних пунктів меню, які представлені у вигляді фрагментів. За допомогою меню, яке розташоване внизу екрану, користувач має змогу переключатися між фрагментами, які не перестворюються щоразу, як користувач їх вмикає, що забезпечує високу швидкодію і плавний інтерфейс додатку.

Модуль «obd» містить у собі засоби для роботи з інтерфейсом OBDII. У цьому модулі реалізовані константи, які відповідають різним командам інтерфейсу OBDII, а також класи для обробки відповіді цих команд. Наприклад, для запиту значення обертів двигуна (яке відображається на тахометрі автомобіля), потрібно використати команду «01 0C». А для обчислення відповіді скористаємося формулою для обрахунку обертів двигуна ((A*256)+B)/4.

Даний модуль також містить так званий OBDManager (менеджер ОБД), задача якого полягає у взаємодії з пристроєм через інтерфейс OBDII. Для того, щоб OBDManager почав зчитувати дані, йому на вхід потрібно передати вхідний (input) і вихідний (output) потоки даних. Це можуть бути як і потоки з Bluetooth сокету, так і з Internet сокету. ОБД менеджер може зчитувати дані як одинично, так і циклічно. Для цього, йому потрібно вказати необхідні команди для зчитування, і запустити його. Після успішного запуску, OBDManager проводить ініціалізацію OBDII контролера, для того, щоб прибрати зайві символи, дублювання, а також, задати необхідні параметри

передачі. Після успішної ініціалізації менеджер починає зчитувати вказані дані.

Як було визначено емпіричним шляхом, зчитування даних відбувається на різних машинах з різною швидкістю. Велику роль в швидкості передачі грає комп'ютер машини. В середньому, на машині економ класу 2010-х років виробництва, зчитування відбувається зі швидкістю 3-4 параметри в секунду. Отже, мінімальна затримка якої вдавалося досягнути складала 0.25с.

Таким чином, можна безперервно зчитувати декілька параметрів з затримкою кадру в:

N * 0.25

де N – кількість параметрів, або зчитувати один параметр з затримкою в 0.25с.

Модуль "bluetooth" містить у собі застосунки для роботи з Bluetooth в мобільній платформі Android.

Після з'єднання, модуль "bluetooth" повертає модулеві "app" сокет для передачі даних через Bluetooth. Управління Bluetooth з'єднанням реалізовано за допомогою шаблону проектування «Одинак» (анг. Singleton). Одинак — це породжувальний шаблон проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього. Це надає зручний та уніфікований доступ до управління Bluetooth з'єднанням з різних частин програми.

BluetoothManager – основний клас модуля "bluetooth". Цей клас надає основний інструментарій для роботи з Bluetooth. За допомогою цього класу можна писати у сокет, отримувати повідомлення з сокета, з'єднуватися з пристроєм, перевіряти чи з'єднання сформоване і закривати з'єднання. Також, цей клас надає можливість пошуку пристроїв з увімкненим Bluetooth, які знаходяться в межах видимості Bluetooth приймача.

3.3. Висновок до розділу 3

Завдяки створеним програмним модулям і налагодженню обміну данними між ними вдалося створити мобільний додаток на платформі Android для забезпечння взаємодії клієнта з СТО. Додаток має зручний і легкий у використанні інтерфейс, можливість слідкувати за витратами на обслуговування автомобіля та може виконати швидку комп'ютерну діагностику автомобіля. Також, отріману діагностичну інформацію можна відправити на перевірку кваліфікованому співробітнику станції, що є перевагою цього додатку перед аналогами.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Кількість автомобілів в Україні з кожним днем зростає. Також зростає попит на обслуговування і ремонт цих авто. Станція технічного обслуговування – дуже корисний і потрібний сервіс в наш час, тому, рано чи пізно, повинна відбутися диджиталізація цього виду бізнесу. Для розвитку і отримання конкурентної переваги у СТО повинні з'явитися мобільні додатки, як, наприклад, у компаній доставки чи таксі.

Головною метою даного дипломного проекту було створення додатку, який дозволить віддалено взаємодіяти клієнту з СТО. Це дозволить клієнту зручно, просто, а головне доступно відслідковувати ремонтну історію свого авто і витрати на обслуговування. А наявність комп'ютерної діагностики дозволить виконати кваліфікований аналіз віддалено і без зайвих витрат.

Усі автомобілі, які зараз виробляються, підтримують діагностику на базі OBDII інтерфейсу. В деяких країнах, на законодавчому рівні, зобов'язують автовиробників влаштовувати підтримку OBDII інтерфейсу у свої автомобілі. Дуже багато автомобілістів вже добре знайомі з даним інтерфейсом, і, навіть, мають свої власні OBDII адаптери. Майже всі адаптери, які існують на масовому ринку базуються на основі чіпів ELM, адже ці чіпи підтримують всі існуючі протоколи для роботи з OBDII інтерфейсом.

У результаті аналізу існуючих рішень для діагностування автомобілів на мобільній платформі Android, було визначено, що їх основний недолік – необхідність мати навички діагноста для розуміння результатів діагностики.

Особливістю розробленого додатку є те, що він надає можливість віддаленого діагностування автомобіля. Для цього потрібно лише під'єднати телефон до OBDII адаптера, і надсилати дані на СТО для перевірки кваліфікованим спеціалістом. Таким чином, професійний діагност зможе віддалено проводити діагностування та моніторинг автомобіля, без прив'язки до місцезнаходження транспортного засобу.

Для того, щоб використовувати додаток, що розроблений у даному дипломному проекті, достатньо мати Android телефон, з доступом в Internet, і OBDII адаптер, який підтримує передачу даних, використовуючи технологію Bluetooth.

У перспективі, до даного додатка можна додати велику кількість корисного функціоналу, який покращить якість і корисність додатка. Такими функціями можуть бути:

- Нагадування заміни мастила;
- Виклик евакуатора;
- Акційні пропозиції від СТО;
- Оплата;
- Відстеження стану ремонту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Технічній регламент з технічного обслуговування і ремонту колісних транспортних засобів [Електронний ресурс]. - Режим доступу: https://zakon.rada.gov.ua/laws/show/643-2013-%D0%BF#Text
2. Станція технічного обслуговування [Електронний ресурс]. - Режим доступу:https://uk.wikipedia.org/wiki/Станція технічного обслуговування
3. OBD-II (Check Engine Light) Trouble Codes [Електронний ресурс]. - Режим доступу: https://www.obd-codes.com/trouble_codes/
4. OBD-II PIDs [Електронний ресурс]. - Режим доступу: http://obdcon.sourceforge.net/2010/06/obd-ii-pids/
5. Torque lite [Електронний ресурс]. - Режим доступу: https://play.google.com/store/apps/details?id=org.prowl.torquefree
6. Car Scanner [Електронний ресурс]. - Режим доступу: https://play.google.com/store/apps/details?id=com.ovz.carscanner
7. OBD Auto Doctor [Електронний ресурс]. - Режим доступу: https://play.google.com/store/apps/details?id=com.obdautodoctor
8. Griffiths D. Head First Android Development: A Brain-Friendly Guide / D. Griffiths, D. Griffiths., 2015.
9. Lake I. Professional Android / I. Lake, L. Meier., 2018.
10. Филлипс Б. Android. Программирование для профессионалов / Б. Филлипс, К. Стюарт, К. Марсикано., 2016. – (Питер).
11. Android Studio [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Android_Studio

					КНТЕУ 121 06з-02.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	Розробка мобільного додатку взаємодії клієнта з СТО	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Палагута К.О.		24.01.20		*СВД*	*42*	*43*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Горбуль М.О.		24.01.20	*Список використаних джерел*			

12. Android Studio User Guide [Електронний ресурс]. - Режим доступу: https://developer.android.com/studio/build
13. Gradle Build Tool [Електронний ресурс]. - Режим доступу: https://gradle.org/
14. Автоматизація збірки за допомогою Gradle [Електронний ресурс]. - Режим доступу: https://studfile.net/preview/6845260/
15. Firebasedatabases [Електронний ресурс]. - Режим доступу: https://firebase.google.com/docs/database
16. Firebase Cloud Functions, (or running code on Firebase Servers!) [Електронний ресурс]. - Режим доступу: https://javebratt.com/firebase-cloud-functions/

ТЕХНІЧНЕ ЗАВДАННЯ

Положення технічного завдання для розробки Android додатку взаємодії клієнта з СТО.

1. Загальні відомості

Назва проекту: EngineApp.

Планові строки розробки: травень 2020 – вересень 2020.

Потенційні користувачі: користувачі мобільних пристроїв з встановленою операційною системою Android.

Призначення програмного рішення: надати можливість клієнту автосервісу слідкувати за витратами на технічне обслуговування свого автомобіля, а також робити дистанційну компьютерну діагностику через мобільній додаток.

Мета створення програмного рішення: освоєння сучасних технологій розробки Android.

2. Структура проекту

Додаток складається з клієнтської і серверної частин.

Клієнтська частина реалізована у вигляді додатку для платформи Android.

Серверна частина розгорнута за допомогою сервісу Firebace.

3. Функціональні вимоги до додатку

Функціональними можливостями додатку є: реєстрація, авторизація, запис на діагностику, відображення списку робіт і витрат, швидка комп'ютерна діагностика.

					КНТЕУ 121 06з-02.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		28.02.20	Розробка мобільного додатку взаємодії клієнта з СТО	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Палагута К.О.		28.02.20		*ТЗ*	*44*	*43*
Гарант		Криворучко О.В.		28.02.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Горбуль М.О..		28.02.20	*Технічне завдання*			

ТЕСТУВАННЯ ДОДАТКА ENGINEAPP

Щоб упевнитися у працездатності додатка було проведено тестування основної функції комп'ютерної діагностики автомобіля.

Оскільки розроблений додаток передбачає взаємодію з автомобілем тестування в Android-емуляторі є неможливим. Для тестування був використаний смартфон MEIZU M3 NOTE з наступними характеристиками:

- Операційна система Android 5.1;
- Оперативна пам'ять 3Gb;
- Підтримка Bluetooth 4.0

У якості OBDII сканера о обрано пристрій Viecar. Сканер Viecar має наступні переваги:

- Висока швидкодія;
- Наявність Bluetooth 4.0;
- Підтримка всіх протоколів OBDII.

Передік автомобілів на яких було протестовано додаток:

- Renault Safrane 1997 бензин 2.5;
- Citroen C5 2009 дизель 2.0.

Під час тестування були протестовані всі функції, на обох автомобілях. Зчитування даних OBD відбувається без помилок і великих затримок. В середньому на цих автомобілях час зчитування одного параметру складає 0.3 секунди.

ДОДАТКИ

Додаток А

Код класу MainActivity

```
public class MainActivity extends AppCompatActivity {
    Button btnSignIn, btnRegister, btnCall, enter;
    FirebaseAuth auth;
    FirebaseDatabase db;
    DatabaseReference users;
    RelativeLayout root;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnSignIn = findViewById(R.id.btnSignIn);
        btnRegister = findViewById(R.id.btnRegister);
        btnCall = findViewById(R.id.call_button);
        auth = FirebaseAuth.getInstance();
        db = FirebaseDatabase.getInstance();
        users = db.getReference("Users");
        root = findViewById(R.id.root_element);
        btnRegister.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showRegisterWindow();
            }
        });
        btnSignIn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showSignInWindow();
            }
        });}
    private void showSignInWindow(){
        AlertDialog.Builder dialog = new AlertDialog.Builder(this);
        dialog.setTitle("Увійти");
        dialog.setMessage("Введіть данні для входу");
        LayoutInflater inflater = LayoutInflater.from(this);
        View sign_in_window = inflater.inflate(R.layout.sign_in_window,null);
        dialog.setView(sign_in_window);
        final MaterialEditText email = sign_in_window.findViewById(R.id.emailField);
        final MaterialEditText pass = sign_in_window.findViewById(R.id.passFild);
        dialog.setNegativeButton("Відміна", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int which) {
                dialogInterface.dismiss();
            }
        });
        dialog.setPositiveButton("Увійти", new DialogInterface.OnClickListener() {
            @Override
```

Продовження дод. А

```
        public void onClick(DialogInterface dialogInterface, int which) {
            if (TextUtils.isEmpty(email.getText().toString())){
                Snackbar.make(root, "Введіть вашу пошту", Snackbar.LENGTH_SHORT).show();
                return;
            }
            if (pass.getText().toString().length() < 5){
                Snackbar.make(root, "Введіть пароль більше, ніж на 5 символів",
Snackbar.LENGTH_SHORT).show();
                return;
            }
auth.signInWithEmailAndPassword(email.getText().toString(), pass.getText().toString())
                    .addOnSuccessListener(new OnSuccessListener<AuthResult>() {
                        @Override
                        public void onSuccess(AuthResult authResult) {
                            startActivity(new Intent(MainActivity.this, menuActivity.class));
                            finish();
                        }
                    }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Snackbar.make(root, "Помилка авторизації. Спробуйте пізніше" +
e.getMessage(),Snackbar.LENGTH_SHORT).show();
                }
            });
        }
    });
    dialog.show();
}
private void showRegisterWindow() {
    AlertDialog.Builder dialog = new AlertDialog.Builder(this);
    dialog.setTitle("Зареєструватися");
    dialog.setMessage("Введіть данні для регестрації");
    LayoutInflater inflater = LayoutInflater.from(this);
    View register_window = inflater.inflate(R.layout.register_window,null);
    dialog.setView(register_window);
    final MaterialEditText email = register_window.findViewById(R.id.emailField);
    final MaterialEditText pass = register_window.findViewById(R.id.passFild);
    final MaterialEditText name = register_window.findViewById(R.id.namelField);
    final MaterialEditText phone = register_window.findViewById(R.id.phoneField);
    final MaterialEditText mark = register_window.findViewById(R.id.markField);
    final MaterialEditText model = register_window.findViewById(R.id.modelField);
    final MaterialEditText vin = register_window.findViewById(R.id.vinField);
    dialog.setNegativeButton("Відміна", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int which) {
            dialogInterface.dismiss();
        }
    });
    dialog.setPositiveButton("Додати", new DialogInterface.OnClickListener() {
        @Override
```

Продовження дод. А

```
        public void onClick(DialogInterface dialogInterface, int which) {
            if (TextUtils.isEmpty(email.getText().toString())){
                Snackbar.make(root, "Введіть пошту", Snackbar.LENGTH_SHORT).show();
                return;
            }
            if (TextUtils.isEmpty(name.getText().toString())){
                Snackbar.make(root, "Введіть ім'я", Snackbar.LENGTH_SHORT).show();
                return;
            }
            if (TextUtils.isEmpty(phone.getText().toString())){
                Snackbar.make(root, "Введіть телефон", Snackbar.LENGTH_SHORT).show();
                return;
            }
            if (pass.getText().toString().length() < 5){
                Snackbar.make(root, "Пароль має бути більше 5 символів",
Snackbar.LENGTH_SHORT).show();
                return;
            }
            if (TextUtils.isEmpty(mark.getText().toString())){
                Snackbar.make(root, "Введіть марку авто", Snackbar.LENGTH_SHORT).show();
                return;
            }if (TextUtils.isEmpty(model.getText().toString())){
                Snackbar.make(root, "Введіть модель авто", Snackbar.LENGTH_SHORT).show();
                return;
            }if (TextUtils.isEmpty(vin.getText().toString())){
                Snackbar.make(root, "Введіть vin номер автомобіля", Snackbar.LENGTH_SHORT).show();
                return;
            }
auth.createUserWithEmailAndPassword(email.getText().toString(), pass.getText().toString())
                    .addOnSuccessListener(new OnSuccessListener<AuthResult>() {
                        @Override
                        public void onSuccess(AuthResult authResult) {
                            User user = new User();
                            user.setEmail(email.getText().toString());
                            user.setName(name.getText().toString());
                            user.setPass(pass.getText().toString());
                            user.setPhone(phone.getText().toString());
                            user.setMark(mark.getText().toString());
                            user.setModel(model.getText().toString());
                            user.setVin(vin.getText().toString());
users.child(FirebaseAuth.getInstance().getCurrentUser().getUid())
                                    .setValue(user)
                                    .addOnSuccessListener(new OnSuccessListener<Void>() {
                                        @Override
                                        public void onSuccess(Void aVoid) {
                                            Snackbar.make(root, "Користувача додано",
Snackbar.LENGTH_SHORT).show();
                                        }
                                    });
                        }
```

Продовження дод. А

```
                    }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Snackbar.make(root, "Такий email вже існує!" +
e.getMessage(),Snackbar.LENGTH_SHORT).show();
                }
            });
        }
    });
    dialog.show();
  }
}
```

Додаток Б

Код класу menuActivity

```
public class menuActivity extends AppCompatActivity {
    private BottomNavigationView bottomNavigationView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
        bottomNavigationView = findViewById(R.id.bottomNav);
        bottomNavigationView.setOnNavigationItemSelectedListener(bottomNavMethod);
        getSupportFragmentManager().beginTransaction().replace(R.id.container, new Home()).commit();
    }
    private final BottomNavigationView.OnNavigationItemSelectedListener bottomNavMethod = new
BottomNavigationView.OnNavigationItemSelectedListener() {
        @SuppressLint("NonConstantResourceId")
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
          Fragment fragment=null;
            switch (menuItem.getItemId()){
                case R.id.navigation_home:
                    fragment = new Home();
                    break;
                case R.id.navigation_dashboard:
                    fragment = new obdFragment();
                    break;
                case R.id.navigation_notifications:
                    fragment = new infoFragment();
                    break;
            }
            getSupportFragmentManager().beginTransaction().replace(R.id.container, fragment).commit();
            return true;
        }
    };
}
```

Додаток В

Код класу User

```
package com.example.engineapp.Models;
public class User {
        private String name, email, pass, phone, mark, model, vin;
        public User(){}
        public User(String name, String email, String pass, String phone, String mark, String model,
String vin) {
            this.name = name;
            this.email = email;
            this.pass = pass;
            this.phone = phone;
            this.mark = mark;
            this.model = model;
            this.vin = vin;
        }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getMark() {
        return mark;
    }
    public void setMark(String mark) {
        this.mark = mark;
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
```

Продовження дод. В

```
        this.model = model;
    }
    public String getVin() {
        return vin;
    }
    public void setVin(String vin) {
        this.vin = vin;
    }
}
```

Код класу BluetoothManager

```
@SuppressLint("NewApi")
public class BluetoothManager extends CommService{
   private BtConnectThread mBtConnectThread;
   private BtWorkerThread mBtWorkerThread;
   private final StreamHandler ser = new StreamHandler();
   BtCommService(Context context, Handler handler)
   {
      super(context, handler);
      BluetoothAdapter mAdapter = BluetoothAdapter.getDefaultAdapter();
      mAdapter.cancelDiscovery();
      elm.addTelegramWriter(ser);
      ser.setMessageHandler(elm);
   }
  @Override
   public synchronized void start()
   {
      log.log(Level.FINE, "start");
      if (mBtConnectThread != null)
      {
         mBtConnectThread.cancel();
         mBtConnectThread = null;
      }
      if (mBtWorkerThread != null)
      {
         mBtWorkerThread.cancel();
         mBtWorkerThread = null;
      }
      setState(STATE.LISTEN);
   }
   @Override
   public synchronized void connect(Object device, boolean secure)
   {
      log.log(Level.FINE, "connect to: " + device);

      if (mState == STATE.CONNECTING)
      {
         if (mBtConnectThread != null)
         {
            mBtConnectThread.cancel();
            mBtConnectThread = null;
         }
      }

      if (mBtWorkerThread != null)
      {
         mBtWorkerThread.cancel();
         mBtWorkerThread = null;
      }
```

Продовження дод. Г

```
    setState(STATE.CONNECTING);
    mBtConnectThread = new BtConnectThread((BluetoothDevice)device, secure);
    mBtConnectThread.start();
}
private synchronized void connected(BluetoothSocket socket, BluetoothDevice
device, final String socketType)
{
    log.log(Level.FINE, "connected, Socket Type:" + socketType);
    if (mBtConnectThread != null)
    {
        mBtConnectThread.cancel();
        mBtConnectThread = null;
    }
    if (mBtWorkerThread != null)
    {
        mBtWorkerThread.cancel();
        mBtWorkerThread = null;
    }
    mBtWorkerThread = new BtWorkerThread(socket, socketType);
    mBtWorkerThread.start();
      connectionEstablished(device.getName());
 }
@Override
public synchronized void stop()
{
    log.log(Level.FINE, "stop");
    elm.removeTelegramWriter(ser);
    if (mBtConnectThread != null)
    {
        mBtConnectThread.cancel();
        mBtConnectThread = null;
    }
    if (mBtWorkerThread != null)
    {
        mBtWorkerThread.cancel();
        mBtWorkerThread = null;
    }
    setState(STATE.OFFLINE);
}
@Override
public synchronized void write(byte[] out)
{
    mBtWorkerThread.write(out);
}
private class BtConnectThread extends Thread
{
    private final BluetoothDevice mmDevice;
    private BluetoothSocket mmSocket;
    private final String mSocketType;
    BtConnectThread(BluetoothDevice device, boolean secure)
```

Продовження дод. Г

```
{
   mmDevice = device;
   BluetoothSocket tmp = null;
   mSocketType = secure ? "Secure" : "Insecure";
   final UUID SPP_UUID = UUID
      .fromString("00001101-0000-1000-8000-00805F9B34FB");
   try{
      if (secure)
      {
         tmp = device.createRfcommSocketToServiceRecord(SPP_UUID);
      } else
      {
         tmp = device.createInsecureRfcommSocketToServiceRecord(SPP_UUID);
      }
   } catch (IOException e)
   {
      log.log(Level.SEVERE, "Socket Type: " + mSocketType + "create() failed", e);
   }
   mmSocket = tmp;
   logSocketUuids(mmSocket, "BT socket");
}
private void logSocketUuids(BluetoothSocket socket, String msg)
{
   if(log.isLoggable(Level.INFO))
   {
      StringBuilder message = new StringBuilder(msg);
      message.append(" - UUIDs:");
      ParcelUuid[] uuids = socket.getRemoteDevice().getUuids();
      if(uuids != null)
      {
         for (ParcelUuid uuid: uuids)
         {
            message.append(uuid.getUuid().toString()).append(",");
         }
      }
      else
      {
         message.append("NONE (Invalid BT implementation)");
      }
      log.log(Level.INFO, message.toString());
   }
}

public void run()
{
   log.log(Level.INFO, "BEGIN mBtConnectThread SocketType:" + mSocketType);
   try
   {
      log.log(Level.FINE, "Connect BT socket");
      mmSocket.connect();
```

Продовження дод. Г

```
            }
            catch (IOException e)
            {
                log.log(Level.FINE, e.getMessage());
                cancel();

                log.log(Level.INFO, "Fallback attempt to create RfComm socket");
                BluetoothSocket sockFallback;
                Class<?> clazz = mmSocket.getRemoteDevice().getClass();
                Class<?>[] paramTypes = new Class<?>[]{Integer.TYPE};
                try {
                    Method m = clazz.getMethod("createRfcommSocket", paramTypes);
                    Object[] params = new Object[]{1};
                    sockFallback = (BluetoothSocket) m.invoke(mmSocket.getRemoteDevice(), params);
                    mmSocket = sockFallback;
                    logSocketUuids(mmSocket, "Fallback socket");
                    mmSocket.connect();
                }
                catch (Exception e2)
                {
                    log.log(Level.SEVERE, e2.getMessage());
                    connectionFailed();
                    return;
                }
            }
            synchronized (BtCommService.this)
            {
                mBtConnectThread = null;
            }
            connected(mmSocket, mmDevice, mSocketType);
        }
        synchronized void cancel()
        {
            try
            {
                log.log(Level.INFO, "Closing BT socket");
                mmSocket.close();
            } catch (IOException e)
            {
                log.log(Level.SEVERE, e.getMessage());
            }
        }
    }
    private class BtWorkerThread extends Thread
    {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;
        BtWorkerThread(BluetoothSocket socket, String socketType)
        {
```

Продовження дод. Г

```
            log.log(Level.FINE, "create BtWorkerThread: " + socketType);
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
            try
            {
               tmpIn = socket.getInputStream();
               tmpOut = socket.getOutputStream();
            } catch (IOException e)
            {
               log.log(Level.SEVERE, "temp sockets not created", e);
            }
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
            ser.setStreams(mmInStream, mmOutStream);
         }
         public void run()
         {
            log.log(Level.INFO, "BEGIN mBtWorkerThread");
            try
            {
               ser.run();
            } catch (Exception ex)
            {
                  log.log(Level.SEVERE, "Comm thread aborted", ex);
            }
            connectionLost();
         }
         synchronized void write(byte[] buffer)
         {
            ser.writeTelegram(new String(buffer).toCharArray());
         }
         synchronized void cancel()
         {
            try
            {
               log.log(Level.INFO, "Closing BT socket");
               mmSocket.close();
            } catch (IOException e)
            {
               log.log(Level.SEVERE, e.getMessage());
            }
         }
      }
   }
```

Код класу OBDManager

```
public class OBDManager
   extends CommService
   implements Runnable
{
   private Socket mSocket;
   private final StreamHandler ser = new StreamHandler();
   private Thread serThread;
   public OBDManager()
   {
      super();
   }
   public OBDManager(Context context, Handler handler)
   {
      super(context, handler);
      ser.setMessageHandler(elm);
   }
   @Override
   public void start()
   {
      log.fine("start");
      // set up protocol handlers
      elm.addTelegramWriter(ser);
      // create communication thread
      serThread = new Thread(this);
      serThread.start();
   }
   @Override
   public void stop()
   {
      log.fine("stop");
      elm.removeTelegramWriter(ser);
      // close socket
      try
      {
         mSocket.close();
      } catch (Exception e)
      {
         log.severe(e.getMessage());
      }
      setState(STATE.OFFLINE);
   }
   @Override
   public void write(byte[] out)
   {
      ser.writeTelegram(new String(out).toCharArray());
   }
   @Override
   public void run()
```

Продовження дод. Д

```
    {
        ser.run();
        connectionLost();
    }
    class ConnectThread extends Thread
    {
        final CommService svc;
        final String device;
        int portNum;

        ConnectThread(CommService svc, String device, int portNum)
        {
            this.svc = svc;
            this.device = device;
            this.portNum  = portNum;
        }
        @Override
        public void run()
        {
            log.info(String.format("Connecting to %s port %d", device, portNum));
            setState(STATE.CONNECTING);
            try
            {
                mSocket = new Socket();
                InetSocketAddress addr = new InetSocketAddress(device, portNum);
                mSocket.connect(addr);
                ser.setStreams(mSocket.getInputStream(), mSocket.getOutputStream());
                connectionEstablished(device);
                svc.start();
            } catch (Exception e)
            {
                log.severe(e.getMessage());
                connectionFailed();
            }
        }
    }
    public void connect(Object device, int portNum)
    {
        new ConnectThread(this, String.valueOf(device), portNum).start();
    }
    @Override
    public void connect(Object device, boolean secure)
    {
        connect(device, secure ? 23 : 22);
    }
private transient static final String FID_GAUGE_SERIES = "GAUGE_SERIES";
private final transient LayoutInflater mInflater;
private static int resourceId;
private final transient int minWidth;
private final transient int minHeight;
```

Продовження дод. Д

```
private final DisplayMetrics mDisplayMetrics;
private static final NumberFormat labelFormat = new DecimalFormat("0;-#");
static class ViewHolder
{
   AwesomeSpeedometer gauge;
   TextView tvDescr;
}
public ObdGaugeAdapter(Context context, int resource, int minWidth, int minHeight, DisplayMetrics metrics)
{
   super(context, resource);
   mInflater = (LayoutInflater) context
      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
   resourceId = resource;
   this.minWidth = minWidth;
   this.minHeight = minHeight;
   mDisplayMetrics = metrics;
}
@Override
public void remove(EcuDataPv object)
{
   object.remove(FID_GAUGE_SERIES);
   object.removePvChangeListener(this);
   object.setRenderingComponent(null);
   super.remove(object);
}

/* (non-Javadoc)
@Override
public void add(EcuDataPv currPv)
{
   try
   {
      CategorySeries category = (CategorySeries) currPv.get(FID_GAUGE_SERIES);
      if (category == null)
      {
         category = new CategorySeries(String.valueOf(currPv.get(EcuDataPv.FID_DESCRIPT)));
         category.add(String.valueOf(currPv.get(EcuDataPv.FID_UNITS)),
               ((Number)currPv.get(EcuDataPv.FID_VALUE)).doubleValue());
         currPv.put(FID_GAUGE_SERIES, category);
         currPv.setRenderingComponent(null);
      }
   }
   finally
   {
      currPv.addPvChangeListener(this, PvChangeEvent.PV_MODIFIED);
   }
   super.add(currPv);
}
@Override
public View getView(int position, View convertView, ViewGroup parent)
```

Продовження дод. Д

```
{
   ViewHolder holder;
   EcuDataPv currPv = getItem(position);
   int pid = Objects.requireNonNull(currPv).getAsInt(EcuDataPv.FID_PID);
   if (convertView == null)
   {
      convertView = mInflater.inflate(resourceId, parent, false);
      holder = new ViewHolder();
      holder.gauge = convertView.findViewById(R.id.chart);
      holder.tvDescr = convertView.findViewById(R.id.label);
      convertView.setTag(holder);
   }
   else
   {
      holder = (ViewHolder)convertView.getTag();
   }
   int pidColor = ChartActivity.getItemColor(pid!=0?pid:position);
   convertView.setBackgroundColor(pidColor & 0x10FFFFFF);
   holder.tvDescr.setText(String.valueOf(currPv.get(EcuDataPv.FID_DESCRIPT)));
   Number minValue = (Number) currPv.get(EcuDataPv.FID_MIN);
   Number maxValue = (Number) currPv.get(EcuDataPv.FID_MAX);
   Number value =    (Number) currPv.get(EcuDataPv.FID_VALUE);
   if (minValue == null) minValue = 0f;
   if (maxValue == null) maxValue = 255f;
   holder.gauge.setTrianglesColor(pidColor);
   holder.gauge.setUnit(currPv.getUnits());
   holder.gauge.setMinSpeed(minValue.floatValue());
   holder.gauge.setMaxSpeed(maxValue.floatValue());
   holder.gauge.speedTo(value.floatValue());
   return convertView;
}
@Override
public void pvChanged(PvChangeEvent event)
{
   if(event.getKey().equals(EcuDataPv.FIELDS[EcuDataPv.FID_VALUE])
         && event.getValue() instanceof Number)
   {
      ProcessVar currPv = (ProcessVar) event.getSource();
      CategorySeries series = (CategorySeries) currPv.get(FID_GAUGE_SERIES);
      series.set(0,
            String.valueOf(currPv.get(EcuDataPv.FID_UNITS)),
            ((Number)event.getValue()).doubleValue());
   }
  }
}
```