МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____
"____"_____2022 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: " Мобільний додаток діагностики автомобіля по протоколу OBD II "

Виконавець: студент групи СП-435 Волошин Григорій Володимирович

Керівник: викладач Нечипурук Віталій Володимирович

Нормоконтролер: Тупота Є.В

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітнього ступеня бакалавр
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
О.Є. Литвиненко
"___" _____ 2022 р

ЗАВДАННЯ
на виконання дипломного проекту студента
Волошина Григорія Володимировича

1. Тема проекту: «Мобільний додаток діагностики автомобіля по протоколу OBD II»
затверджена наказом ректора від 20.04.2022 р. № 512/ст

2. Термін виконання проекту: з 05.05.2022 р. до 05.05.2022 р.

3. Вихідні данні до проекту: програмний продукт розробити за допомогою середовища інтегрованого середовища розробки PyCharm.

4. Зміст пояснювальної записки:
   1. Аналіз існуючих систем діагностики автомобілів
   2. Вимоги до мобільного додатку діагностування автомобілів
   3. Структура додатку діагностики автомобілів
   4. Реалізація додатку діагностики автомобілів
5. Перелік обов'язкових слайдів презентації:
   1. Тема, виконавець, керівник.
   2. Існуючи методики, аналіз недоліків, постановка завдання.
   3. Вимоги до програмного засобу.
   4. Структура засобу, діаграма класів
   5. Інтерфейс програмного засобу

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та вивчення літератури. Складання графіку роботи	11.05.22 – 12.05.22	
2.	Написання 1 розділу, представлення керівнику	12.05.22 – 12.05.22	
3.	Написання 2 розділу, представлення керівнику	14.05.22 – 16.05.22	
4.	Написання 3 розділу, представлення керівнику	17.05.22 – 19. 05.22	
5	Написання 4 розділу, представлення керівнику	20.05.22 – 25.05.22	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	26.05.22 – 30.05.22	
7.	Проходження нормо-контролю, перепліт пояснювальної записки.	01.06.22	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	02.06.22	
9.	Отримання відгуку керівника, рецензії.	03. 06.22	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 1 папка, 1 конверт)	10.06.22 – 20.06.22	

7. Дата видачі завдання 16.05.2022

Керівник: викл. Нечипурук В. В.

Завдання прийняв до виконання: Волошин Г. В.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Мобільний додаток діагностики автомобіля по протоколу OBD II».

Кількість сторінок 45.

Кількість таблиць: 1.

Кількість рисунків: 10.

МОБІЛЬНИЙ ДОДАТОК, ДИЗАЙН, ІНТЕРФЕЙС, ПРОЕКТУВАННЯ, РОЗРОБКА СИСТЕМИ , ДІАГНОСТИКА АВТОМОБІЛЯ.

Об'єкт дослідження – мобільний додаток діагностики автомобіля по протоколу OBD II.

Мета дипломної роботи – створення мобільного додатку діагностики автомобіля по протоколу OBD II, що дасть змогу покращити існуючі системи для зручнішої діагностики автомобіля.

Метод розробки – ООП об’єктно-орієнтований підхід, та функціональний метод.

Технічні та програмні засоби – ПК з ОС Windows 7, 8 або 10; інтегроване середовище розробки PyCharm.

Результати даної роботи можна переглядати, вивчати, та використовувати у вільному доступі.

Прогнозні припущення щодо розвитку об'єктів розроблення – даний додаток можна розширити шляхом реалізації нового функціоналу для діагностики авто.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ООП – об’єктно-орієнтовне програмування

ОС – операційна система

ТО – технічне обслуговування

OBD (On-Board Diagnostics) – бортова діагностика

ЕБК – електронний блок керування

DTC (Dynamic Traction Control) – система динамічного контролю тяги

ДТП – дорожньо-транспортна пригода

HTML – HyperText Markup Language

ОС – операційна система

ВСТУП

У сучасності автомобільний вид транспорту найбільший по кількості перевезень і пасажирів, і вантажу, ще він найбільш гнучкий, та універсальний, саме завдяки йому, забезпечується своєчасність доставки.

Одною з головних задач автомобільної індустрії є забезпечення безпеки при перевезенні вантажів та пасажирів. На безпеку перевезення впливає чимало факторів, таких як: порушення правил дорожнього руху, недотримання дистанції, водіння у нетверезому стані, виїзд на смугу протилежного руху, перевищення швидкості, неуважність та поломки автомобіля.

Резюмуючи, можна стверджувати, що основним чинником ДТП є людський фактор. Проаналізовавши світову статистику автокатастроф, світове товариство захотіло прибрати цей фактор. Саме тому, на сьогоднішній день, всі провідні автовиробничі компанії, такі як BMW, General Motors, Mercedes, Ford, та інші, не шкодуючи гроші вкладають у створення новітніх систем для виробництва безпілотних автомобілів.

На даний момент, компанія яка більше за інших досягла успіху в розробці автопілота, це Tesla. З кожним роком автопілот на базі нейронних мереж все покращує свої результати, адже кількість навчальних даних збільшується. Так за даними третього кварталу 2021 року, авто Tesla що використовували автопілот в 3.1 рази рідше потрапляли в ДТП, аніж автівки з ручним керуванням.

Проте, це не останнє, що можуть поліпшити автовиробники, за для зменшення кількості автокатастроф. Ще залишаються поломки автомобілів, чи їх окремих деталей.

В цьому напрямку, за останнє десятиріччя виробники авто також зробили великі кроки. За допомогою суперкомп’ютерів модулюються аварії, після чого конструктори вносять правки в проекти, винаходяться новітні матеріали, що в свою чергу покращують показники нових моделей.

Крім цього, системи самодіагностики є в кожній новій автівці, та навіть, все частіше, їх вмонтовують в бортові комп’ютери.

АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ДІАГНОСТИКИ АВТОМОБІЛІВ

1.1. Аналіз статистики автокатастроф та їх причини

Безпечні транспортні засоби мають вирішальну роль у запобіганні та зниження ймовірності серйозних травм. Існує ряд правил ООН з безпеки транспортних засобів. Якщо вони інтегровані в стандарти виробництва країн, вони потенційно можуть спасти безліч життів. До них відноситься: вимога до виробників транспортних засобів, дотримування правил лобового і бічного зіткнення, включання електронного контролю стійкості (для запобігання надмірного повороту керма) і забезпечування установкою подушок безпеки і ременів безпеки у всіх транспортних засобах. Без цих основних стандартів ризик дорожньо-транспортних пригод вище, як для тих, хто знаходиться в автомобілі, так і для тих, хто поза ним.

Дуже важливим чинником запобігання аварій є ТО. Інтервали якого вказані законом, чи підзаконним актом в кожній державі. Проте часто трапляється, що автомобіль потребує ТО до зазначеного у законі терміну. Для цього, у новітніх автомобілях передбачені системи самодіагностики. Переважна більшість сучасних автомобілів більш комп'ютеризовані, ніж будь-коли раніше.

Фактично, ця комп'ютеризація дозволяє ретельно перевіряти автомобілі за допомогою автоматичних діагностичних тестів, щоб підтримувати автомобіль в оптимальному робочому стані.

За допомогою датчиків, які можуть сканувати компоненти, та інтегровані системи автомобіля вмонтованими тестами, на наявність помилок або несправностей, перш ніж вони стануть дійсно небезпечними, що значно знижує потенційну загрозу автокатастроф.

1.2 Діагностика як засіб необхідності

Діагностика автомобіля – це комплекс робіт з огляду, перевірки автомобіля на спеціалізованій СТО, щодо його технічної справності і нормальної працездатності як всього авто, так і окремих механізмів. А вмонтована діагностика автомобіля - це цифровий аналіз різних комп'ютерних систем і компонентів автомобіля, визначення технічного стану автомобіля, не розбираючи його.

Нижче наведено деякі види діагностики в автомобілі:

- Діагностика гальмівної системи: проводжується огляд і перевірка в якому стані гальмівні диски, гальмівні колодки і шланги
- Діагностика систем рульового управління: проводжується огляд і перевірка стану рульових тяг, пиляків, рульових наконечників.
- Діагностика ходової частини: проводжується огляд і перевірка амортизаторів, пружини, кульової опори, сайлентблоків, втулки і ступичних підшипників.
- Перевірка рівнів і стану технічних рідин: перевірка стану масла в двигуні, що охолоджує, гальмівний і рідини гідропідсилювача, масла в коробці передач (АКПП), масла в редукторах, мостах, роздавальних коробках.
- Діагностика трансмісії: огляд і перевірка стану піввісі, карданів і хрестовин, пиляків.
- Огляд і перевірка стану колісних дисків, коліс і тиску в шинах.
- Діагностика освітлювальних приладів
- Діагностика електрообладнання
- Діагностика АКБ (акумулятор)

Хоча ці результати можуть допомогти водіям оцінити стан свого автомобіля, діагностичні тести не є абсолютно точними. У багатьох випадках вони не можуть точно сказати фахівцеві або випробувачу, в чому полягає проблема. Замість цього, вони використовуються для визначення місця

розташування проблеми або потенційної помилки, щоб механіки могли швидше визначити і усунути проблему для водіїв.

Завдяки цим новітнім діагностичним методам тестування, можна відслідковувати, та виявляти помилки до того, як вони призведуть до катастрофічних наслідків.

1.3 ODB-II як система діагностики

OBD - це стандартний протокол, який використовується в більшості малотоннажних автомобілів для отримання діагностичної інформації. Інформація генерується блоками управління двигуном в автомобілі. Вони являють собою вмонтований комп'ютер автомобіля. Порт OBD може бути використаний для збору безлічі важливих даних і надання користувачеві додаткових можливостей.

Система OBD з'явилася, в основному, із-за двох факторів: необхідність регулювання викидів і масове впровадження електронного уприскування палива автовиробниками, починаючи з 1980-х років.

На відміну від карбюраторів, або попередніх механічних систем уприскування палива, електронне уприскування палива (EFI) вимагає комп'ютерного управління. Як і його попередники, EFI регулює подачу палива в двигун, але робить це за допомогою електронних сигналів, а не механічних елементів. Це викликало першу серйозну потребу в установці комп'ютерів в автомобілі.

Кілька автовиробників представили комп'ютерні інтерфейси для своїх власних автомобілів до 1990-х років, але поштовх до стандартизації почався лише в 1991 році, коли Каліфорнійська рада з повітряних ресурсів (CARB) зобов'язала всі автомобілі, що продаються в Каліфорнії, мати можливість використання OBD в тій чи іншій формі. Однак CARB не випускав стандарти для цих систем до 1994 року. Відомий як OBD II, цей стандарт був введений в 1996 модельному році і використовується до цих пір. Попередні ітерації OBD були класифіковані як OBD I.

Кожен новий автомобіль, проданий в США за останні 20 років, та в Європі за останні 15 років, відповідає стандарту OBD II, зображеному на рис 1.1.

Рис.1.1. Система діагностування OBD II

1.4 Принципи та протоколи системи діагностування OBD II

Як випливає з назви, діагностика є основною метою OBD. Коли датчики автомобіля вирішують, що щось не так, вони видають повідомлення, відоме як "код несправності", який може проявлятися у вигляді лампочки" check engine" або іншого попередження на приладовій панелі, або передавати данні за допомогою Bluetooth Сканери OBD можуть перевірити ці коди несправностей, щоб визначити, що саме не так, і очистити їх з пам'яті комп'ютера після усунення проблеми.

Коди несправностей - це лише коди: коди. Замість діагнозу типу" нещільно закрита кришка бензобака " ви побачите рядок букв і цифр, яка незрозуміла без довідника. Коди несправностей починаються з букви і включають чотири або п'ять цифр, які разом вказують на конкретну підсистему і проблему, з якою вона стикається. На додаток до загальних кодів, які застосовуються до всіх автомобілів, окремі виробники, мають свої власні спеціальні коди, що трохи ускладнює зчитування, та обробку одержуваної інформації.

Діагностика може бути найважливішою функцією обладнання OBD, але ці інструменти також можуть бути використані для того, щоб змусити ваш автомобіль їхати швидше.

Кілька брендів вторинного ринку пропонують порт OBD II, як реєстратори даних, так і продуктивні тюнери, які отримують доступ до найважливіших систем автомобіля через порт приладової панелі. Реєстратори даних можуть використовуватися для відстеження таких повсякденних речей, як економія палива, але вони також можуть записувати такі речі, як час проходження кола і вихідну потужність. Професійні гонщики покладаються на ці дані, щоб побачити, як вони поводяться на трасі, і налаштувати свої автомобілі. Варто зазначити, що ці оновлення можуть мати негативні наслідки в інших областях, таких як надійність або економія палива.

Автомобіль, сумісний з OBD2, може використовувати будь-який з п'яти протоколів зв'язку: SAE J1850 PWM, SAE J1850 VPW, ISO9141-2, ISO14230-4

(KWP2000), а з 2003 року також ISO 15765-4 /SAE. Схема порту OBD2 зображена на рис. 1.2.

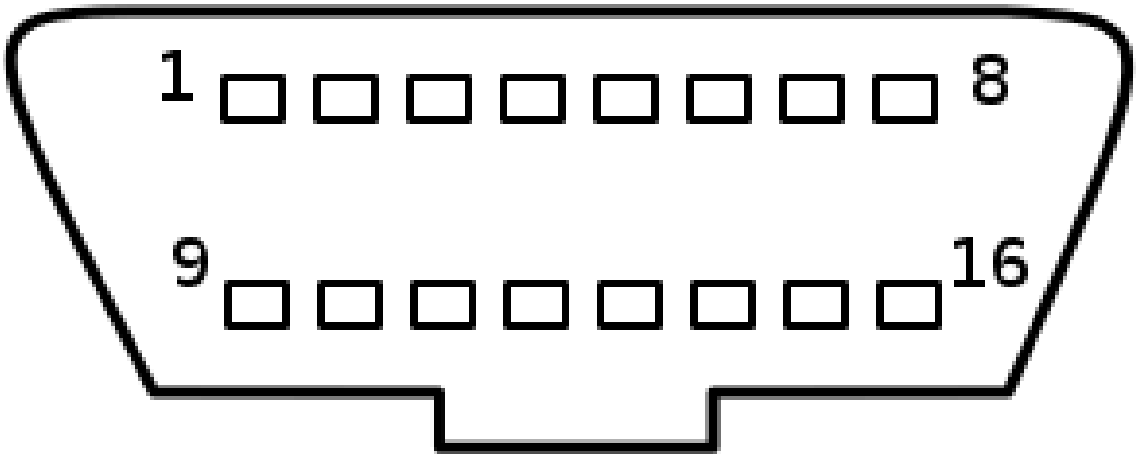

Рис. 1.2. Схема розташування виводів порту OBD2

J2480. ISO15765-4 – це найсучасніший протокол, обов'язковий для всіх автомобілів 2008 року випуску і старше, що продаються в США. Використовує шостий, та чотирнадцятий контакти, зв'язок диференційний. Існує чотири варіанти стандарту ISO15765. Вони відрізняються тільки довжиною і швидкістю шини:

- ISO 15765-4 CAN (11-бітний ідентифікатор, 500 кБ/сек)
- ISO 15765-4 CAN (29-бітний ідентифікатор, 500 кБ/сек)
- ISO 15765-4 CAN (11-бітний ідентифікатор, 250 кБ/сек)
- ISO 15765-4 CAN (29-бітний ідентифікатор, 250 кБ/сек)

ISO14230-4 (KWP2000) - це дуже поширений протокол для автомобілів 2003+, що використовують ISO9141 K-Line. Використовує сьому жилу.
Існує два варіанти стандарту ISO14230-4. Вони відрізняються тільки способом ініціалізації зв'язку. Всі вони використовують 10400 біт в секунду.

- ISO 14230-4 KWP (введення 5 кБ/сек, 10,4 кБ/сек)
- ISO 14230-4 KWP (швидке введення, 10,4 кБ/сек)

ISO9141-2 - це найбільш старий протокол використовувався в основному на європейських автомобілях в період з 2000 по 2004 рік. Використовує сьомий, та, інколи, п’ятнадцятий контакти.

SAE J1850 VPW - це діагностична шина, яка використовується в основному на автомобілях General Motors. Використовує першу жилу. Швидкість передачі даних становить 10,4 кБ/сек.

SAE J1850 ШІМ, це діагностична шина, чи протокол, який використовується в основному на Ford. Використовує перший, та другий контакти, сигнал зв'язку диференціальний, а його швидкість становить 41,6 кБ/сек.

1.5 Існуючі мобільні додатки для діагностики автомобілів

Ознайомившись з відомими сервісами діагностики, було складено таблицю 1.1, ознайомившись з якою користувач зможе підібрати найбільш зручний варіант.

У таблиці можна дізнатися, чи підтримує додаток Bluetooth з’єднання, чи показує параметри ЕБК, що відповідає за усі електричні системи, такі як, електричний підсилювач керма, або дверні контролери. Таблиця відображає наявність системи моніторингу пального для даного автомобіля.

Звісно, саме наявністю, та зручністю зчитування кодів помилок, та моніторингом параметрів реального часу, яких як, швидкість автомобіля, кількість обертів двигуна, температура, та тиск, та інші обумовлена популярність того, чи іншого додатку. Також, у таблиці, можна дізнатись якими мовами надає додаток інформацію, та кількість людей, що користується ним.

Саме покриттям і кількістю що відслідковуються літаків обумовлена популярність того чи іншого сервісу авіарадарів в певному регіоні.

Таблиця 1.1

Найпопулярніші мобільні додатки для діагностики.

Назва	**Car Scanner**	**inCarDoc**	**Torque**

Назва	**Car Scanner**	**inCarDoc**	**Torque**
Мовні версії	Англійська, українська, німецька, арабська, чеська, іспанська, французька, італійська, російська, польська, шведська, японська, китайська, корейська.	Англійська, російська	Англійська, російська
Bluetooth	+	+	+
Моніторинг ЕБК	+	+	-
Моніторинг кодів помилок (DTC)	+	+	+
Параметри реального часу	+	+	+
Безкоштовне використання	+	У безкоштовній версії доступні не всі функції	У безкоштовній версії доступні не всі функції
Моніторинг даних про витрату	+	+	+

Назва	**Car Scanner**	**inCarDoc**	**Torque**
пального			
Кількість завантажень	Більше 10 млн.	Більше 5 млн.	10 млн.

Додаток inCarDoc має широкий функціонал, що підтримує моніторинг ЕБК, проте, на відміну від Car Scanner, у безкоштовному доступі немає всього функціоналу, а саме, моніторингу більшості параметрів реального часу, та моніторингу даних про витрату пального. Torque немає ніяких переваг серед своїх конкурентів, проте він був випущений першим, що зумовлює його популярність.

Також з табличних даних видно, що найбільшу мовну підтримку користувачів (на 14 мовах) надає додаток Car Scanner, тоді як інші додатки відображають дані переважно англійською, та російською. Безперечними перевагами перед іншими додатками володіє Car Scanner, бо окрім підтримки найбільшої кількості мов, він має у безкоштовному доступі увесь функціонал, що й зумовлює його популярність. Car Scanner відображає найактуальнішу, на поточний момент, інформацію про параметри реального часу, а при критичній помилці, він швидко повідомляє користувачеві код помилки, з її ретельним описом.

Висновок

У першому розділі дипломного проекту було проаналізовано статистику автомобільних катастроф, та аргументовано їх передумови. Також, проілюстровано актуальність діагностики автомобіля, за допомогою якої вирішується проблема несвоєчасної поломки, чи зносу певного елемента автомобіля.

На додаток до цього, було описано систему діагностики ODB-II. Проаналізовано її принципи роботи, та протоколи.

Також, розглянуті додатки діагностики автомобіля, що вже існують на ринку. Порівняно їх функціонал, та проаналізовано плюси, та мінуси, кожного з них.

РОЗДІЛ 2
ВИМОГИ ДО МОБІЛЬНОГО ДОДАТКУ ДІАГНОСТУВАННЯ АВТОМОБІЛІВ

2.1. Вимоги проектування мобільного додатку

Мобільний додаток діагностування з точки зору розробника, це сервіс, який забезпечує користувача даними низьку даних про поточний стан автомобіля, а саме коди помилок, що зчитуються датчиками системи, та передається на смартфон. Також надаються додаткові данні реального часу, які залежать від конкретної моделі авто. На рис. 2.1 показано схему вимог, що будуть описані у поточному розділі.

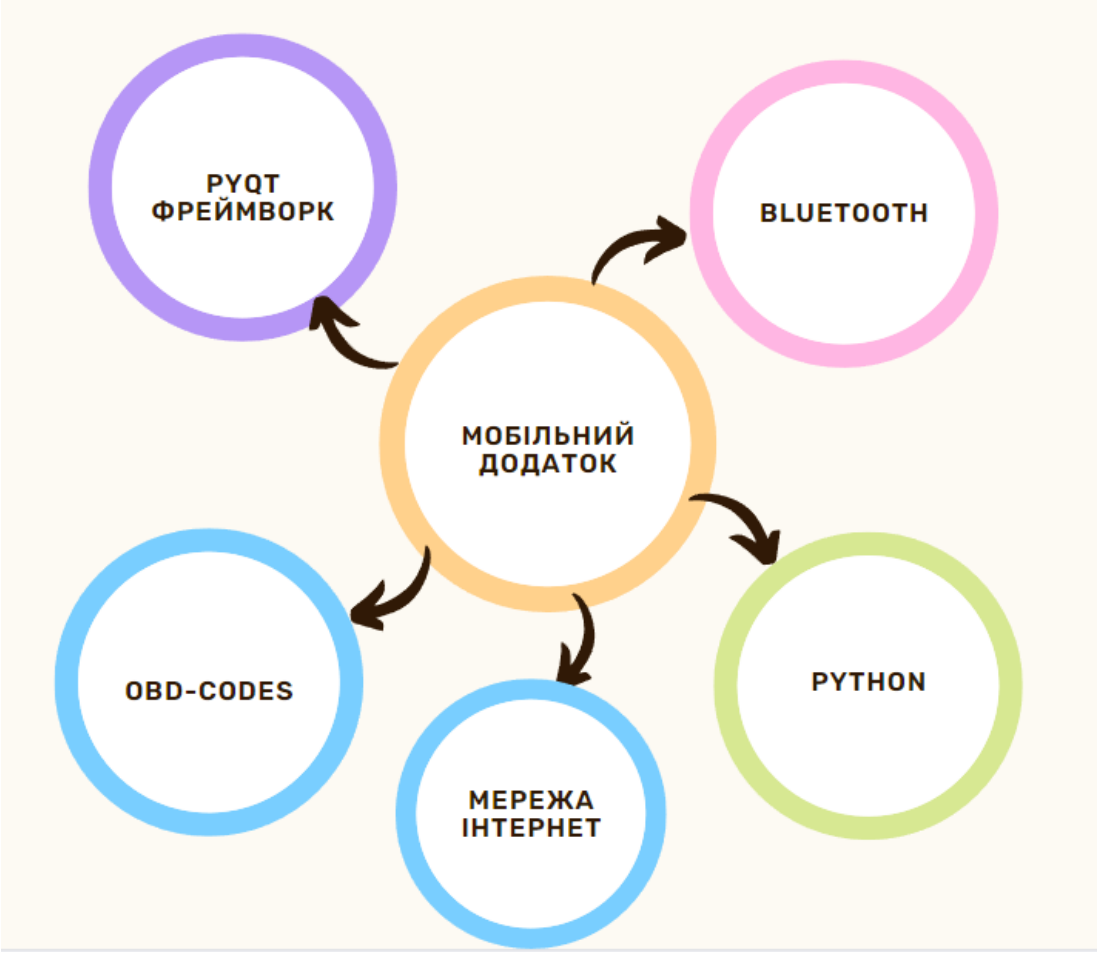

Рис.2.1. Схема вимог до мобільного додатку

2.2. Функціональні вимоги до мобільного додатку

Функціональні вимоги - це вимоги, та функції, які користувач використовує в якості основних можливостей, які повинні пропонувати система. Всі ці функціональні можливості повинні бути обов'язково включені в систему, та описані в технічному завданні. Функціональні вимоги представлені або вказані у формі вхідних даних, які повинні бути надані системі, виконуваної операції і очікуваного результату. В основному це вимоги, які можна побачити безпосередньо в кінцевому продукті, на відміну від нефункціональних вимог.

Розроблений мобільний додаток діагностики авто повинен задовольняти наступні функціональні вимоги :

1) користувач повинен включити Bluetooth
2) під’єднати смартфон до відповідного пристрою через Bluetooth
3) якщо користувач не включив Bluetooth, повідомити про це
4) при підключенні до неправильного пристрою повідомити, цю інформацію користувачеві
5) після підключення до модуля зв’язку повідомити про успішне підключення
6) після успішного підключення, користувач має потрапити на головну сторінку
7) на головній сторінці він бачить назву діагностичної системи, а саме “Car diagnostic”, та логотип системи.
8) також на головній сторінці має бути навігаційна кнопка “error monitoring”
9) натиснувши кнопку “error monitoring” користувач має потрапити на сторінку з панеллю моніторингу помилок

10) на сторінці моніторингу користувач може побачити хронологію помилок, що зафіксував додаток
11) біля коду помилки користувач має бачити помилку в текстовій формі
12) на кожній сторінці користувач має бачити зверху кнопку виходу
13) на кожній сторінці користувач має бачити назву додатку
14) користувач повинен мати змогу вийти з системи
15) якщо гаджет користувача від'єднався від модуля зв'язку, повинне прийти сповіщення
16) якщо на гаджеті користувача вимкнувся Bluetooth, повинне прийти сповіщення

2.3. Нефункціональні вимоги до проекту

Нефункціональні вимоги – це, в основному, вимоги якості, які система повинна задовольняти відповідно до технічного завдання. Пріоритет, або ступінь реалізації цих факторів варіюються в залежності від проекту, та розробників. Вони також включають обмеження на процес розробки системи.

Програмні вимоги.

З програмних вимог можна виділити кросплатформеність, додаток має працювати на Android , Linux , macOS та Windows, оскільки обраний фреймворк має підтримку цих операційних систем.

Продуктивність. Час отримання даних через протокол зв'язку Bluetooth не має бути довшим за 2-3 секунди.

Доступність. Повинен бути миттєвий відгук на будь-яку дію користувача з інтерфейсом.

Надійність. Через протокол зв’язку Bluetooth, сторонній пристрій не має мати доступ до будь-яких персональних даних.

Вимоги до дизайну.

Оскільки якість процесу взаємодії користувача із системою (швидкість, зручність, інтуїтивність) пов’язана з такими психологічними характеристиками людини, як короткострокова, та середньострокова пам’ять, час реакції, можливості сприйняття візуальної інформації, вимоги до дизайну мають велике значення.

Інтерфейс, це найважливіша частина застосунку з точки зору його реклами з метою популяризації, і з точки зору безпосереднього користувача системи, який буде працювати з нею. Інтерфейс впливає на характер рішень, які приймає додаток, він може прискорювати час прийняття рішення, та покращувати, або погіршувати їх якість. Вимоги до дизайну мають давати можливість керувати додатком будь-якому користувачеві, навіть без розуміння програмування, та діагностування автомобілів.

Список вимог до дизайну інтерфейсу виглядає наступним чином:

1) адаптивність дизайну під різні пристрої, та системи;
2) не перевантажити кожний з екранів (сторінок) елементами дизайну
3) орфографічно, та пунктуаційно правильний контент екранів;
4) чітке розділення шапки, основного контенту та низу сторінки;
5) шрифти повинні бути в одному стилі, та якомога менша використана кількість розмірів шрифтів;
6) привабливий, запам'ятний, та унікальний логотип;
7) повнота дизайну;
8) інтуїтивна, та зрозуміла для користувача навігація;
9) зрозумілі кнопки, іконки, тексти, та позначення;

Використовувати традиційні або загальноприйняті кнопки, та іконки, для прискорення навчання роботи з даним додатком.

Висновок

В цьому розділі ретельно сформульовано функціональні, та нефункціональні вимоги до розроблюваного застосунку. Описана їх доречність, та необхідність.

Перелік функціональних вимог було згруповано в єдиний список, в якому представлені функціональних вимоги до всіх розроблюваних екранів, та контенту.

Нефункціональні вимоги кластеризовано в окремі блоки, завдяки яким, взаємодія із застосунком буде відбуватися продуктивно, швидко, та зручно.

РОЗДІЛ 3
СТРУКТУРА ДОДАТКУ ДІАГНОСТИКИ АВТОМОБІЛІВ

3.1. Архітектура застосунку діагностики автомобілів

Застосунок діагностики автівок спроектований, за допомогою шаблону проектування, або так званого паттерна проектування Model-View-Presenter(MVP), який, у свою чергу похідний від Model-view-controller.

Цей паттерн проектування відокремлює додаток на три взаємопов'язані функціональні частини.

Перша з яких, це модель (model), яка являє собою набір класів для даних, що відображаються, або з якими будуть відбуватися інші процеси у інтерфейсі. Друга функціональна частина, це інтерфейс, або представлення (view), яка управляє всіма елементами на екрані, та відсилає події до пред'явника. Пред'явник (Presenter), реагує на події, та оновлює модель, що змінює дані, або бізнес логіку, після чого, може повідомити представленню про зміни в моделі. Завдяки такому шаблону, зміни, що відбулися в одному з компонентів, мінімізують втручання в інші частини, та також їх можна легко відслідкувати, та протестувати.

Оскільки, в діагностичній системі немає мінливої бізнес логіки, то модель можна було б представити у вигляді незалежного блоку, використовуючи більш простий шаблон проектування Model-View-ViewModel, проте, щоб зберегти можливість масштабування застосунку, було прийнято рішення використати паттерн MVP.

На рис. 3.1 представлена схема MVP.

Рис. 3.1 Схема шаблону проектування MVP

3.2. Діаграма компонентів шаблону проектування MVP

Діаграма компонентів (Component Diagram) вказує на структурні частини системи і взаємозв'язки між ними. Елементами діаграми виступають різні об'єкти, такі як: бібліотеки, модулі, файли, пакети. Ця діаграма дає розуміння поведінки кожної частини програмного коду.

На рис 3.2 представлена діаграма компонентів розроблюваного додатку.

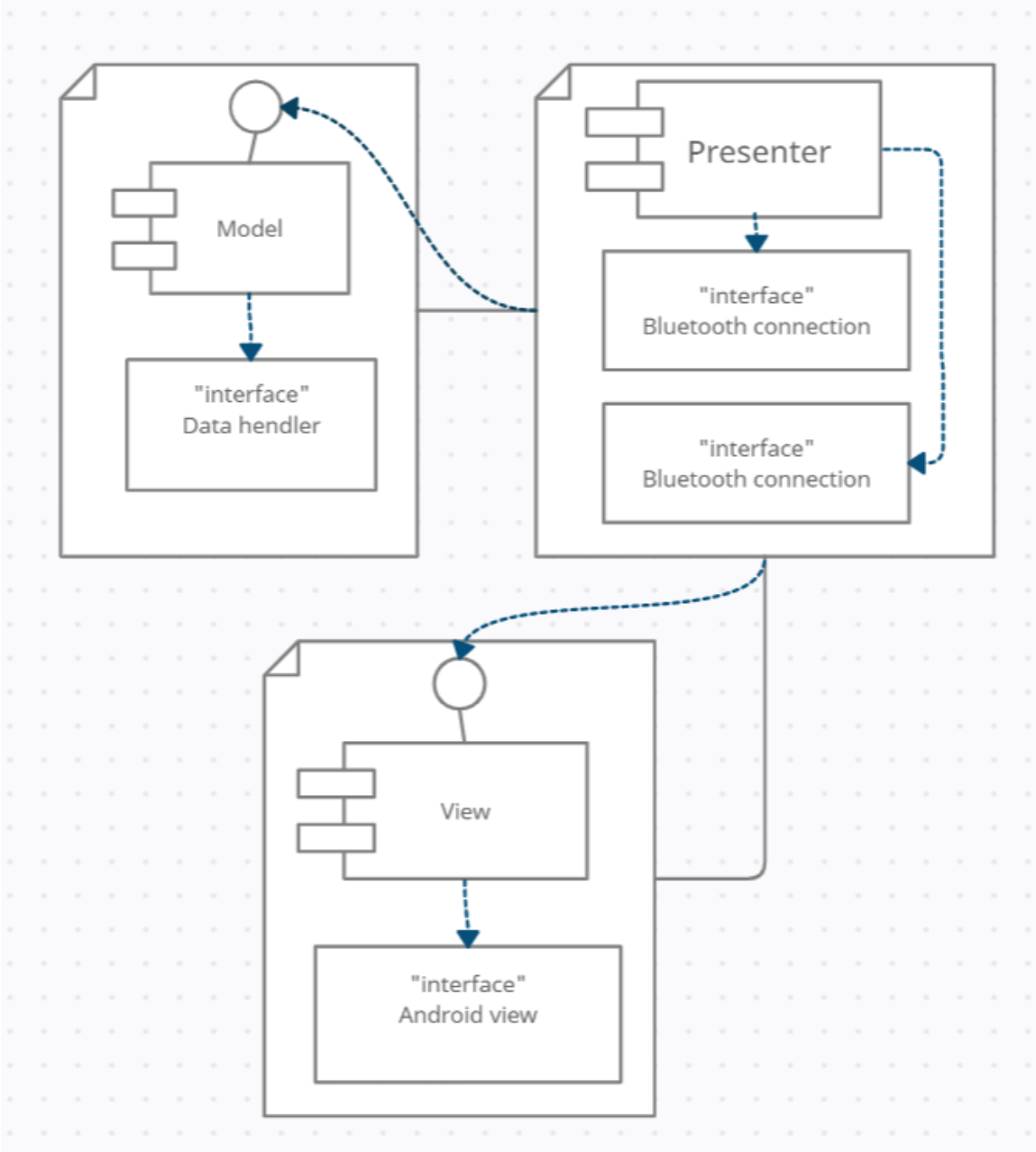

Рис.3.2. Діаграма компонентів додатку згідно шаблону проектування MVP

3.3. Діаграма структури класів

Діаграма класів – це UML-діаграма, яка описує систему, візуалізуючи різні типи об'єктів усередині системи та види статичних зв'язків, що існують між ними. Також вона ілюструє атрибути класів. Суттєвий момент в діаграмі класів в тому, що це первинне представлення архітектури програмного коду, на основі якого формуються реальні об’єкти.

На рис. 3.3. представленні класи програми.

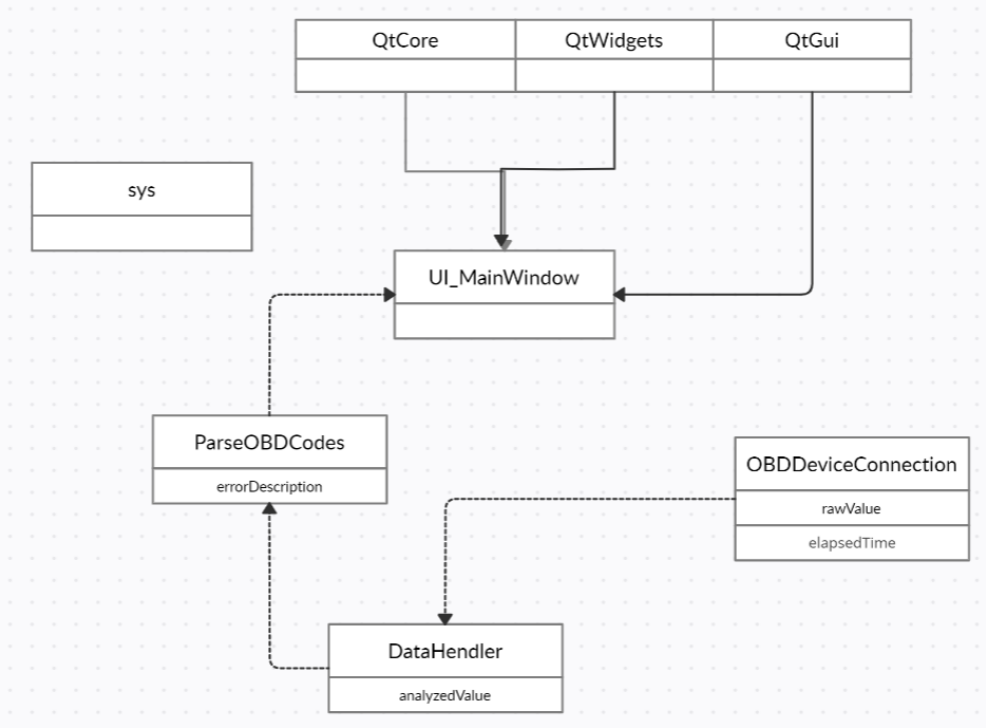

Рис. 3.3. Діаграма класів додатку діагностики авто.

На рисунку відображено структуру класів, яку створено за допомогою паттерна MVP. Відображено 4 основні класси:

1) UI_MainWindow – клас, який відображає весь інтерфейс та дані, описує взаємодію з користувачем.
2) DataHendler – клас, який обробляє дані і знаходить код помилки;
3) OBDDeviceConnection – клас, котрий дістає необроблені дані через протокол Bluetooth;

4) ParseOBDCodes – клас, який відправляє запит, щодо знайденого коду помилки.

Висновок

В цьому розділі було проілюстровано схему, та дві діаграми, котрі відображають систему.

У схемі MVP відображено, підхід до проектування мобільного застосунку.

Діаграма компонентів продемонструвала структурний зміст шаблону проектування Model-View-Presenter, котрий є основою всього додатку.

На діаграмі класів нарисовано всі зв’язки класів системи, та описано значення кожного з них.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ДОДАТКУ ДІАГНОСТИКИ АВТОМОБІЛІВ

4.1. Розробка додатку діагностики автомобілів

Прототипи та мінімально життєздатні продукти (Minimal Viable Product) є важливою частиною розробки продукту.

Коли розробник надає форму своїй ідеї продукту, та починає роботу над нею, виходить елемент з низьким рівнем інженерії та низькою функціональністю, це називають прототипом продукту. Прототип — це спосіб швидкої перевірки основних ідей та припущень, що лежать в основі продукту. Завдяки йому, можна внести правки, та переосмислити деякі припущення, мінімальною кількістю затрачених зусиль і ресурсів.

Прототипи бувають різноманітні, вони відрізняються візуалізації та процентом реалізації. Одна з головних задач прототипів, це вкласти розуміння бізнес частини проекту з виконавцем і узгодити їх уявлення про кінцевий чи проміжний результат.

Прототип - це базовий макет засобу, який візуалізує розташування всіх елементів і функцій. Він дозволяє наочно проілюструвати всі задумки, а також внести правки ціною мінімальних зусиль і витрат.

Часто навіть професійні, й повноцінні команди починають аналіз та проектування прототипу з прикидки на папері. За допомогою цього можна об ґрунтувати ідею, та подивитися на неї під іншим кутом, перш ніж починати реалізовувати, оскільки це не займає багато часу і ресурсів.

Прототип мобільного додатку діагностики автомобілів було реалізовано об’єктно орієнтовано, на мові програмування Python інтегрованому середовищі розробки PyCharm. Прототип було створено з метою подальшою реалізацією мінімально життєздатного продукта.

4.2. Реалізація розробленого застосунку діагностики

Починається реалізація з написання зовнішнього вигляду початкового екрану, за допомогою фреймворку PyQt, створено класс Ui_MainWindow, та метод “setupUi(self, MainWindow)”, в якому ініціалізуємо усі елементи, за допомогою класів пакетів фреймворку PyQt. В цьому класі реалізовано всі зміни інтерфейсу при роботі з користувача з додатком, та відображаються дані, що надходять з інших класів. Нижче, у лістінгу 4.1 представлена частина класу Ui_MainWindow.

Лістинг 4.1

Частина класу Ui_MainWindow та метод setupUi

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):

    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(360, 740)
        MainWindow.setStyleSheet("background-color: rgb(255, 255,
255);")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(-10, -10, 391,
741))
        self.groupBox.setObjectName("groupBox")
        self.label = QtWidgets.QLabel(self.groupBox)
```

```
        self.label.setGeometry(QtCore.QRect(50, 150, 291, 141))
        self.label.setStyleSheet("font: 63 16pt \"Segoe UI
Semibold\";")
        self.label.setFrameShadow(QtWidgets.QFrame.Raised)
        self.label.setLineWidth(1)
        self.label.setTextFormat(QtCore.Qt.AutoText)
        self.label.setObjectName("label")
        self.pushButton = QtWidgets.QPushButton(self.groupBox)
        self.pushButton.setGeometry(QtCore.QRect(50, 380, 281,
71))
        self.pushButton.setMouseTracking(False)

self.pushButton.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
        self.pushButton.setAcceptDrops(False)
        self.pushButton.setStyleSheet("background-color: rgb(0, 0,
96);\n"
"font: 63 16pt \"Segoe UI Semibold\";\n"
"color: rgb(255, 255, 255);")

self.pushButton.setLocale(QtCore.QLocale(QtCore.QLocale.Ukrainian,
QtCore.QLocale.Ukraine))
        self.pushButton.setIconSize(QtCore.QSize(30, 30))
        self.pushButton.setDefault(False)
        self.pushButton.setObjectName("pushButton")
        self.groupBox_2 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_2.setGeometry(QtCore.QRect(0, -11, 381,
111))
        self.groupBox_2.setStyleSheet("background-color: rgb(0, 0,
96);")
        self.groupBox_2.setObjectName("groupBox_2")
        self.label_2 = QtWidgets.QLabel(self.groupBox_2)
```

Потім транслюємо інтерфейс користувача у методі “retranslateUi(self, MainWindow)”.

Лістинг 4.2

Метод retranslateUi

```
    def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
    self.groupBox.setTitle(_translate("MainWindow", "GroupBox"))
    self.label.setText(_translate("MainWindow",
"<html><head/><body><p><span style=\" font-size:14pt;\">For the
program to work properly,</span></p><p><span style=\" font-
size:14pt;\">you need to turn on
Bluetooth</span></p></body></html>"))
    self.pushButton.setText(_translate("MainWindow", "Turn on
Bluetooth"))
```

```
    self.groupBox_2.setTitle(_translate("MainWindow", "GroupBox"))
    self.label_2.setText(_translate("MainWindow",
"<html><head/><body><p><span style=\" font-size:10pt;
color:#ffffff;\">Car diagnostic</span></p></body></html>"))
```

Після натиску кнопки активації Bluetooth, за допомогою бібліотеки jnius реалізовано включення на операційній системі Android та під'єднання до сокету Bluetooth, після чого починаємо отримувати необроблені дані з під'єднаного OBD2 девайсу, що реалізовано в класі OBDDeviceConnection.

Лістинг 4.3

Клас OBDDeviceConnection

```
from jnius import autoclass

class OBDDeviceConnection(object):

      def pushButtonCliced(self, clicked, autoclass):
            if (clicked == true):
    BluetoothAdapter =
autoclass('android.bluetooth.BluetoothAdapter')
    BluetoothDevice =
autoclass('android.bluetooth.BluetoothDevice')
    BluetoothSocket =
autoclass('android.bluetooth.BluetoothSocket')
    UUID = autoclass('java.util.UUID')

    def get_socket_stream(name):
        global send_stream, recv_stream
        paired_devices =
BluetoothAdapter.getDefaultAdapter().getBondedDevices().toArray()
        socket = None
        for device in paired_devices:
            if device.getName() == name:
                socket = device.createRfcommSocketToServiceRecord(
                    UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB"))
                recv_stream = socket.getInputStream()
                break
```

Далі у класі DataHendler оброблюється всі отримувані дані, що надходять з класу OBDDeviceConnection. За допомогою регулярних виразів виділяються статуси, коди помилок. В цьому допомагає вмонтований в Python пакет re з регулярними виразами.

Лістинг 4.1

Клас DataHendler

```
import re

class DataHendler (object):
    def handle(self):
        WHITESPACE_PATTERN = re.search("\\s", rawValue)
        BUS_INIT_PATTERN = re.search("(BUS INIT)|(BUSINIT)|(\\.)",
rawValue)
        SEARCHING_PATTERN = re.search("SEARCHING", rawValue)
        CARRIAGE_PATTERN = re.search("[\r\n]", rawValue)
        CARRIAGE_COLON_PATTERN = re.search("[\r\n].:", rawValue)
        COLON_PATTERN = re.search(":", rawValue)
        STARTS_WITH_ALPHANUM_PATTERN = re.search("[^a-z0-9 ]",
rawValue)
```

Оскільки немає ніякого прикладного програмного інтерфейса (Application Programming Interface) з кодами помилок, було прийнято рішення діставати, або парсити ці дані з офіційного сайту OBD2 . Оброблена помилка з класу DataHendler передається до класу ParseOBDCodes, в якому запитами, за допомогою бібліотеки requests дістаємо необхідну інформацію щодо конкретної помилки. Відокремлюємо її від html тегів.

Лістинг 4.5

Клас ParseOBDCodes

```
from bs4 import BeautifulSoup
import requests
import re

url = "https://www.obd-codes.com/p"
class ParseOBDCodes(object):

    def parse(self, analizedValue, url):
        url = url + analizedValue
        requestForAllData = requests.get(url)
        soup = BeautifulSoup(requestForAllData.text, 'lxml')
        raw_header = str(soup.find('h1'))
        errorDescription = re.sub(r'<h1>', ' ', raw_header)
```

4.3. Результати тестування додатку діагностики автомобілів

1. Перший вітальний екран, та він же екран включення Bluetooth. Нажавши кнопку, вмикається Bluetooth, якщо у додатка є необхідні доступи.

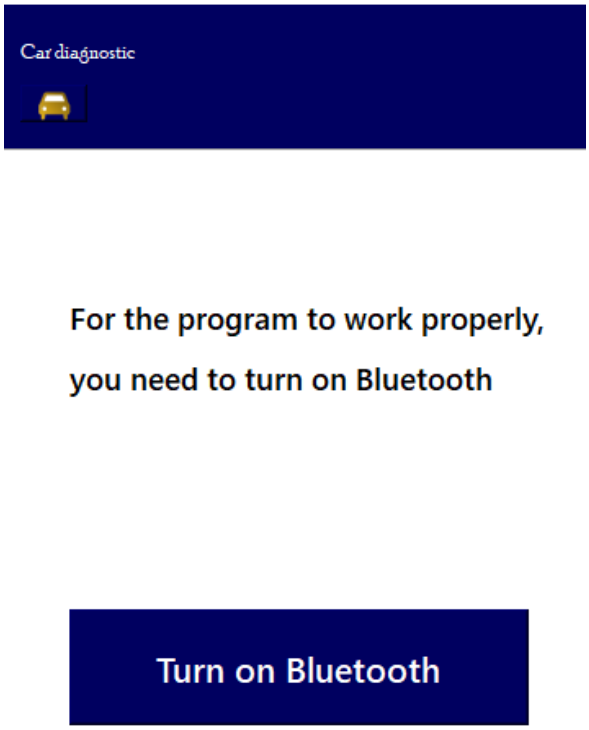

Рис.4.1 Екран включення Bluetooth

Після вдалого включення користувач потрапляє на домашній екран. На якій зображено інформація про даний додаток. Натиснувши кнопку йде перехід до наступного екрану.

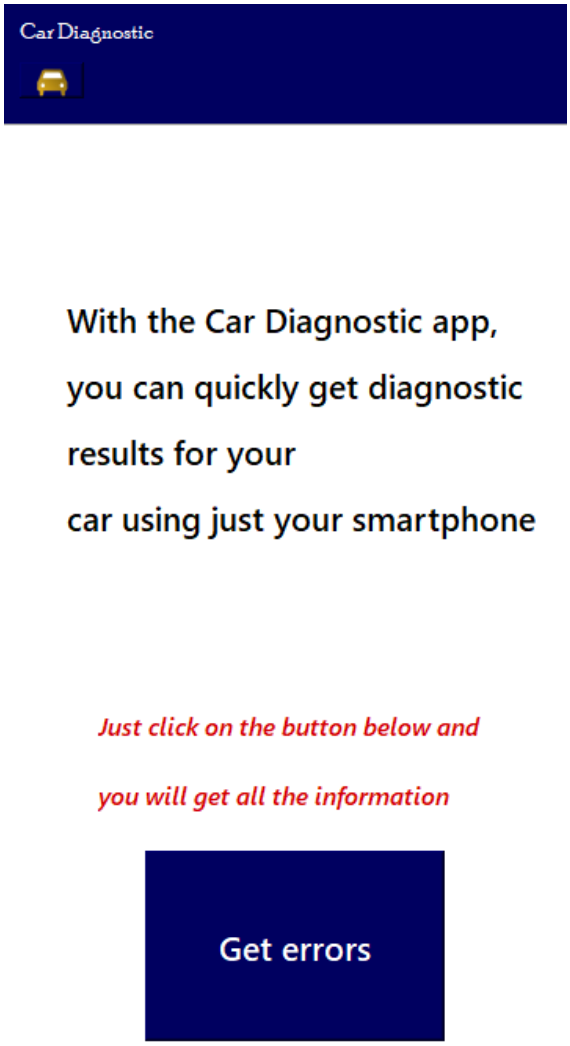

Рис.4.2. Домашній екран програми

Наступний екран із списком помилок підключеної автівки. Натиснувши відповідну до помилки кнопку, йде перехід до екрану з описанням помилки.

Errors in your car

	New Column	Show details
1	P0410	Show details
2	P0410	Show details
3	P0410	Show details
4	P0410	Show details
5	P0410	Show details
6	P0410	Show details
7	P0217	Show details
8	P0410	Show details
9	P0410	Show details
10	P0410	Show details
11	P0410	Show details
12	P0410	Show details

Рис.4.3 Список помилок під’єднаного авто

4. Останній екран із назвою та описом помилки.

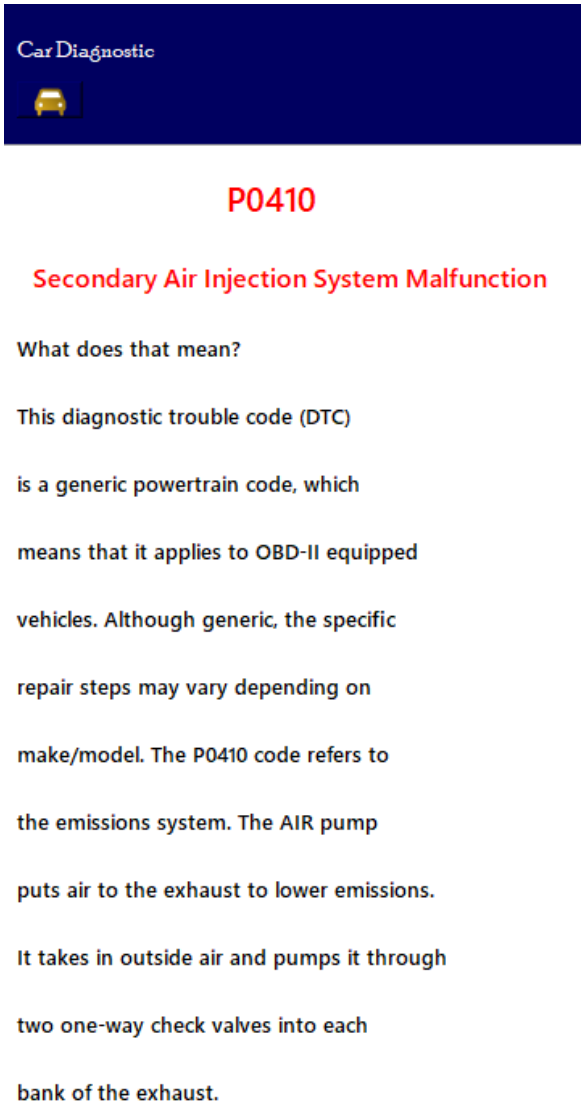

Рис.4.4. Екран опису помилки

Висновки

В результаті реалізації четвертої частини даної роботи було створено прототип додатку діагностування автомобілів за допомогою протоколу OBD2, який відправляє інформацію по протоколу Bluetooth. Розроблений засіб дозволяє моніторити помилки автомобіля за допомогою будь-якого гаджета на ОС Android. За допомогою даного продукту ви можете дізнаватися всю необхідну інформацію про ці помилки.

У даному розділі також було продемонстровано лістинг основних пакетів програми. В ньому представлено всі необхідні класи для відображення шаблона проектування Model-View-Presenter.

ВИСНОВКИ

В ході даної роботи були вивчені технології Python, PyQt framework, MVP, робота з Bluetooth, на ОС Android підхід до розробки через тестування, основи функціонального та ООП програмування.

Основна мета даної роботи була досягнена - був розроблений додаток діагностики автомобілів.

Завдяки реалізованій системі власники автівок зможуть слідкувати за станом свого авто у зручній для них формі. Адже у сьогоденні гаджет є у кожного.

В майбутньому планується розробка додатку з більшою кількістю функціоналу, та підтримкою декількох мов.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Б. Хендерсон Дж. та Х. Хейнес «OBD-II и электронные системы управления двигателем», 2009. -87с.
2. Р.В. Єфименко «АНАЛІЗ ПРИЧИН ВИНИКНЕННЯ ДТП І РОЗРОБКАЗАХОДІВ ІЗ ПІДВИЩЕННЯ БЕЗПЕКИ ДОРОЖНЬОГО РУХУ» 2015 -28с.
3. Трейсі Мартін «How To Use Automotive Diagnostic Scanners» 2008. - 124с.
4. Боб Грегорі та Гарі Персиваль «Architecture Patterns with Python» 2020. - 41с.
5. Роберт Мартін «Чистый код. Создание, анализ и рефакторинг» 2008. - 73с.
6. Валентіно Лі, Роббі Шелл, та Хетер Шнейдер «Mobile Applications: Architecture, Design, and Development» 2008. - 146с.
7. Дронов Володимир та Прохоренок Миколай «Python 3 и PyQt 5. Разработка приложений.» 2019. - 211с.
8. Эрик Маттес «Python Crash Course» 2015. - 92с.

ДОДАТОК А.

ЛІСТІНГ ПРОГРАМИ

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):

    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(360, 740)
        MainWindow.setStyleSheet("background-color: rgb(255, 255,
255);")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(-10, -10, 391,
741))
        self.groupBox.setObjectName("groupBox")
        self.label = QtWidgets.QLabel(self.groupBox)
        self.label.setGeometry(QtCore.QRect(50, 150, 291, 141))
        self.label.setStyleSheet("font: 63 16pt \"Segoe UI
Semibold\";")
        self.label.setFrameShadow(QtWidgets.QFrame.Raised)
        self.label.setLineWidth(1)
        self.label.setTextFormat(QtCore.Qt.AutoText)
        self.label.setObjectName("label")
        self.pushButton = QtWidgets.QPushButton(self.groupBox)
        self.pushButton.setGeometry(QtCore.QRect(50, 380, 281,
71))
        self.pushButton.setMouseTracking(False)

self.pushButton.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
        self.pushButton.setAcceptDrops(False)
        self.pushButton.setStyleSheet("background-color: rgb(0, 0,
96);\n"
"font: 63 16pt \"Segoe UI Semibold\";\n"
"color: rgb(255, 255, 255);")

self.pushButton.setLocale(QtCore.QLocale(QtCore.QLocale.Ukrainian,
QtCore.QLocale.Ukraine))
        self.pushButton.setIconSize(QtCore.QSize(30, 30))
        self.pushButton.setDefault(False)
        self.pushButton.setObjectName("pushButton")
        self.groupBox_2 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_2.setGeometry(QtCore.QRect(0, -11, 381,
111))
```

```
        self.groupBox_2.setStyleSheet("background-color: rgb(0, 0,
96);")
        self.groupBox_2.setObjectName("groupBox_2")
        self.label_2 = QtWidgets.QLabel(self.groupBox_2)



def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
    self.groupBox.setTitle(_translate("MainWindow", "GroupBox"))
    self.label.setText(_translate("MainWindow",
"<html><head/><body><p><span style=\" font-size:14pt;\">For the
program to work properly,</span></p><p><span style=\" font-
size:14pt;\">you need to turn on
Bluetooth</span></p></body></html>"))
    self.pushButton.setText(_translate("MainWindow", "Turn on
Bluetooth"))
    self.groupBox_2.setTitle(_translate("MainWindow", "GroupBox"))
    self.label_2.setText(_translate("MainWindow",
"<html><head/><body><p><span style=\" font-size:10pt;
color:#ffffff;\">Car diagnostic</span></p></body></html>"))



import re

class DataHendler (object):
    def handle(self):
        WHITESPACE_PATTERN = re.search("\\s", rawValue)
        BUS_INIT_PATTERN = re.search("(BUS INIT)|(BUSINIT)|(\\.)",
rawValue)
        SEARCHING_PATTERN = re.search("SEARCHING", rawValue)
        CARRIAGE_PATTERN = re.search("[\r\n]", rawValue)
        CARRIAGE_COLON_PATTERN = re.search("[\r\n].:", rawValue)
        COLON_PATTERN = re.search(":", rawValue)
        STARTS_WITH_ALPHANUM_PATTERN = re.search("[^a-z0-9 ]",
rawValue)



from bs4 import BeautifulSoup
import requests
import re

url = "https://www.obd-codes.com/p"
class ParseOBDCodes(object):

    def parse(self, analizedValue, url):
        url = url + analizedValue
        requestForAllData = requests.get(url)
```

```
soup = BeautifulSoup(requestForAllData.text, 'lxml')
raw_header = str(soup.find('h1'))
errorDescription = re.sub(r'<h1>', ' ', raw_header)
```