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

Факультет прикладної математики

Кафедра системного програмування і
спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ В.П. Тарасенко

«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050102 «Комп'ютерна інженерія»

на тему: «Додаток для діагностування автомобілів на базі OBD-II під мобільну платформу iOS»

Виконав:
студент IV курсу, групи КВ-52
Болгов Іван Михайлович _____

Керівник:
к. т. н. доц. каф. СПСКС
Клятченко Ярослав Михайлович _____

Консультант з нормоконтролю:
к. т. н. доц. каф. СПСКС
Клятченко Ярослав Михайлович _____

Рецензент:

_____
_____ _____

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

Київ – 2019 року

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

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050102 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.П. Тарасенко
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проект студента

Болгова Івана Михайловича

1. Тема проекту «Додаток для діагностування автомобілів на базі OBD-II під мобільну платформу iOS», керівник проекту Клятченко Ярослав Михайлович, к.т.н., доц. каф. СПСКС, затверджені наказом по університету від 22.05.2019 р. №1330-С

2. Термін подання студентом проекту «14» червня 2019 р.

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

4. Зміст пояснювальної записки:
   - аналіз існуючих додатків для діагностики автомобілів для платформи ios та обґрунтування теми дипломного проекту;
   - особливості розробки діагностичного мобільного додатка для операційної системи iOS;
   - реалізація діагностичного додатку.

5. Перелік обов'язкового графічного матеріалу:
   - структура діагностичного мобільного додатка (схема структурна);.
   - структура програмного модуля для збереження даних (схема структурна);
   - схема асинхронної передачі даних (схема алгоритму);
   - схема комунікації із діагностичним адаптером (схема алгоритму);

– презентація за темою роботи.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль			

7. Дата видачі завдання «__» _____ 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	01.04.2019	
2.	Розроблення та узгодження технічного завдання	05.04.2019	
3.	Аналіз існуючих мобільних додатків для діагностики автомобілів	10.04.2019	
4.	Розроблення структури додатки та переліку програмних модулів	14.04.2019	
5.	Підготовка матеріалів першого розділу дипломного проекту	24.04.2019	
6	Розроблення програмного модуля для комунікації через OBDII-адаптер	30.04.2019	
7.	Підготовка матеріалів другого розділу дипломного проекту	05.05.2019	
8	Розробка модуля збереження даних	10.05.2019	
9.	Розробка модуля віддаленої передачі	15.05.2019	
10.	Розробка графічного інтерфейсу	20.05.2019	
11.	Підготовка матеріалів третього розділу дипломного проекту	25.05.2019	
12.	Підготовка графічної частини дипломного проекту	30.05.2019	
13.	Оформлення документації дипломного проекту	05.06.2019	

Студент І. М. Болгов

Керівник проекту Я.М. Клятченко

АНОТАЦІЯ

Об’єкт розробки – створення клієнтського програмного рішення у вигляді мобільного додатка для операційної системи iOS для здійснення віддаленої діагностики транспортних засобів, які підтримують протокол OBD II.

Розроблена система має наступні функції: авторизація в якості унікального користувача для подальшого збереження діагностичної інформації; зчитування помилок транспортного засобу та можливість їх відключення; збереження вибраних параметрів у файли, які можуть бути імпортовані до бази даних; передача параметрів у реальному часі для віддаленого аналізу, який може провести спеціаліст.

В ході розробки:

– проведено аналіз існуючих додатків, які дозволяють взаємодіяти із транспортним засобом через протокол OBD II;

– сформульована архітектура мобільного додатку, яка дозволить легко модифікувати функціонал або створювати новий;

– реалізована комунікація додатка та OBD II-адаптера;

– реалізовано обмін повідомленнями між клієнтом (додатком власника авто) та сервісом (додаток – спеціаліста);

– розроблено веб-сервіс для збереження інформації та переадресації від клієнта до сервісу.

Ключові слова:

OBD II, Web Sockets, віддалена діагностика, Bluetooth LE (Low Energy).

SUMMARY

The main task of this development project is to create a client software solution for remote diagnostics of vehicles, which support OBDII protocol. The realization will base on a mobile application for iOS operating system.

The developed system has the following functions:

– to authorize as an unique user for further saving of the diagnostic information;

– to read the vehicle's errors and to have an opportunity to turn them off;

– to save the selected parameters in csv files which can be imported to the remote database;

– to transfer the diagnostic parameters in real-time for remote analysis by a specialist.

The following was done during development:

– the analysis of the existing applications, which are able to interconnect to the vehicle via the OBDII protocol, is carried out;

– the mobile application architecture, which allow to easy modify the functional or create a new one, is defined;

– the communication of the application with an OBD II-adapter is executed;

– the communication between a client (an application of the car owner) and a service (an application of the automotive specialist);

– the web service for storing information and redirecting from the customer to the service is developed.

Keywords:

OBD II, Web Sockets, діагностика, Bluetooth LE (Low Energy).

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ. 045490.002 ТЗ	Додаток для	4		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Технічне завдання			
	А4	ІАЛЦ. 045490.003 ТП	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Додаток для	61		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Пояснювальна записка			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.001 ОА		
Розробив		Болгов І. М.			Додаток для діагностування автомобілів на базі OBD-II під мобільну платформу iOS ***Опис альбому***	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Клятченко Я. М.					
Консульт.						**КПІ ім. Ігоря Сікорського, ФПМ, КВ-52**	
Н. контроль		Клятченко Я. М.					
Зав. каф.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045490.005 Д1	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Структура діагностичного			
			мобільного додатка.			
			Схема структурна.			
	А4	ІАЛЦ. 045490.006 Д2	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Структура програмного			
			модуля для збереження			
			даних.			
			Схема структурна.			
	А4	ІАЛЦ. 045490.007 Д3	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS			
			Схема асинхронної			
			Передачі даних.			
			Схема алгоритму.			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045490.008 Д4	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Схема комунікації із			
			діагностичним адаптером.			
			Схема алгоритму.			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.002 ОА	Арк. 3

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Додаток для віддаленої діагностики автомобілів на базі протоколу OBDII для мобільної платформи iOS».

Галузь застосування: діагностика транспортних засобів.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи ступеня «бакалавр комп'ютерної інженерії», затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення мобільного додатку для віддаленої діагностики транспортних засобів для операційної системи iOS.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється:

- сумісність з операційною системою iOS;
- можливість авторизації у додатку;
- можливість зчитувати помилки та видаляти їх за бажанням користувача;
- можливість зберігати певні вибрані показники у файл, який можна імпортувати до бази даних;

- можливість передачі показників у реальному часі для детального аналізу спеціалістів;
- можливість отримання результату аналізу, який отримав спецаліст.

5.2. Вимоги до програмного та апаратного забезпечення користувача

- Apple iPhone із операційною системою iOS 11.0 та новіше;
- Наявність транспортного засобу з підтримкою OBDII;
- Наявність доступу до мережі Internet (GPRS, EDGE, 3G, 4G).

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту
1.	Вивчення літератури за тематикою проекту	01.04.2019
2.	Розроблення та узгодження технічного завдання	05.04.2019
3.	Аналіз існуючих мобільних додатків для діагностики автомобілів	10.04.2019
4.	Розроблення структури додатки та переліку програмних модулів	14.04.2019
5.	Підготовка матеріалів першого розділу дипломного проекту	24.04.2019
6	Розроблення програмного модуля для комунікації через OBDII-адаптер	30.04.2019
7.	Підготовка матеріалів другого розділу дипломного проекту	05.05.2019
8	Розробка модуля збереження даних	10.05.2019
9.	Розробка модуля віддаленої передачі	15.05.2019
10.	Розробка графічного інтерфейсу	20.05.2019
11.	Підготовка матеріалів третього розділу дипломного проекту	25.05.2019
12.	Підготовка графічної частини дипломного проекту	30.05.2019
13.	Оформлення документації дипломного проекту	05.06.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новоорозроблена			
	А4	ІАЛЦ. 045490.004 ПЗ	Додаток для	61		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Пояснювальна записка			
	А1	ІАЛЦ. 045490.005 Д1	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Структура діагностичного			
			мобільного додатка.			
			Схема структурна.			
	А4	ІАЛЦ. 045490.006 Д2	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			

			прин		**ІАЛЦ.045490.003 ТП**	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив		Болгов І. М.			Додаток для діагностування автомобілів на базі OBD-II під мобільну платформу iOS Відомість технічного проекту	Літ. / Аркуш 1 / Аркушів 2
Перевірив		Клятченко Я. М.				
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ, КВ-52
Н. контроль		Клятченко Я. М.				
Зав. каф.		Тарасенко В.П.				

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Структура програмного			
			модуля для збереження			
			даних.			
			Схема структурна.			
	А4	ІАЛЦ. 045490.007 Д3	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Схема асинхронної			
			Передачі даних.			
			Схема алгоритму.			
	А4	ІАЛЦ. 045490.008 Д4	Додаток для	1		
			діагностування			
			автомобілів на базі OBD-II			
			під мобільну платформу			
			iOS.			
			Схема комунікації із			
			діагностичним адаптером.			
			Схема алгоритму.			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал			

ЗМІСТ

ДОДАТКИ

Додаток А. Структура діагностичного мобільного додатка.

Додаток Б. Структура програмного модуля для збереження даних.

Додаток В. Схема асинхронної передачі даних.

Додаток Г. Схема комунікації із діагностичним адаптером.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення.

ALDL – протокол діагностичного зв'язку збірки.

API (Application Programming Interface) – опис методів, за допомогою яких одна програма може взаємодіяти з іншою.

ARC (Automatic Reference Counting) – система керування пам’яттю в мові програмування Swift.

BLE (Bluetooth Low Energy) – технологія Bluetooth з наднизьким енергоспоживанням.

BR / EDR (Bluetooth Basic Rate/ Enhanced Data Rate) – класична технологія Bluetooth.

Core Data – фреймворк від компанії Apple, вбудований в операційну систему iOS, MacOS, який дозволяє розробнику взаємодіяти з базою даних.

CSV (Comma-Separated Values) – текстовий формат, який використовується для представлення табличних даних.

DIP (Dependency inversion principle) – принцип інверсії залежностей.

DTC (Diagnostic Trouble Codes) – діагностичні коди помилок.

ECM – протокол для тестування модуля керування двигуном.

EOBD (European On Board Diagnostic) – Європейська бортова діагностична система.

HTTP (HyperText Transfer Protocol) – протокол передачі даних, що використовується в комп'ютерних мережах.

ISP (Interface segregation principle) – принцип розділення інтерфейсу.

LE (Low Energy) – технологія Bluetooth з наднизьким енергоспоживанням.

LSP (Liskov substitution principle) – принцип підстановки Лісков.

MVC (Modal-View-Controller)– архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

MVP (Modal-View-Presenter) – архітектурний шаблон, похідний від MVC, що відділяє візуальне відображення та поведінку обробки подій у різні класи.

MVVM (Model-View-View Model) – архітектурний шаблон, що зв’язує елементи інтерфейсу зі змінними, які їх описують.

OBD (On-Board Diagnostics) – загальна назва передбачених виробником систем діагностики транспортних засобів.

OBDII – стандарт бортової діагностики, розроблений в середини 90-х років.

OCP (Open/closed principle) – принцип відкритості/закритості.

PID (Parameter Identification) – ідентифікатор параметра, що використовується для запиту діагностичної інформації.

Realm – система управління об'єктної базою даних з відкритим вихідним кодом, розрахована для мобільних пристроїв.

REST API (Representational State Transfer) – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

SAE (Society of Automotive Engineers) – товаривство автотраспортних інженерів.

SOLID (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion) – абревіатура складена з перших літер п'яти базових принципів об'єктно-орієнтованого програмування та дизайну запропонована Робертом Мартіном

SRP (Single responsibility principle) – принцип єдиного обов'язку.

UART (Universal Asynchronous Receiver-Transmitter) – тип асинхронного приймача-передавача, компонентів комп'ютерів та периферійних пристроїв, що передає дані між паралельною та послідовною формами.

VIPER (View-Interactor-Presenter-Entity-Router) – архітектурний шаблон, що поділяє задачі на відображення змін, обробку подій та маршрутизацію.

VIP (View-Interactor-Presenter) – архітектурний шаблон, що створює коло викликів від представлення до інтерактора та від інтерактора до презентера, який викликає оновлення у представлені.

ВСТУП

Автомобільна промисловість безумовно є однією з галузей, що швидко розвиваються, інколи випереджаючи свій час. Сучасний автомобіль здатний регулювати свою продуктивність, щоб знизити витрату палива, показати найкращий напрямок з урахуванням трафіку та погодних умов, виявити помилки в двигуні та багато іншого.

Більшість автомобілів обладнані бортовою діагностикою або портом OBDII, який забезпечує доступ до даних з блоку управління двигуном. Щоб отримати інформацію, потрібно підключити сканер-адаптер.

Автоматичний сканер використовується для отримання діагностичних даних та інформації про загальний стан автомобіля. А потім до роботи підключається програмне забезпечення, яке дозволить передавати ці коди на стаціонарний комп'ютер або мобільний пристрій.

Програмне забезпечення аналізує дані та працює як розумний помічник, коли власник автомобіля керує автомобілем. Програмне забезпечення перевіряє багато параметрів і надає користувачеві корисні поради: як керувати більш обережно, як зменшити витрату палива і як продовжити термін служби автомобіля.

Основна мета програмного забезпечення – дати водіям більшу ступінь розуміння їх автомобілів.

Потреба в розробці такого програмного забезпечення для автомобілів стрімко зростає.

В даному дипломному проекті описано власноруч розроблене програмне забезпечення для платформи iOS, що обробляє отримані від автомобіля коди даних, аналізує їх та дає поради автомобілісту або сервісній службі.

1. АНАЛІЗ ІСНУЮЧИХ ДОДАТКІВ ДЛЯ ДАГНОСТИКИ АВТОМОБІЛІВ ДЛЯ ПЛАТФОРМИ IOS ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1 Опис протоколу OBDII.

Проблема, яка розглядається в даному дипломному проекті – необхідність віддаленої діагностики автомобіля (On-board diagnostics або скорочено – OBD).

Історія розробки таких систем починається з 1968 року, коли Volkswagen впровадив свою першу систему для сканування інжекторних двигунів. Потім бортові комп'ютери починають з’являтися на споживчих транспортних засобах.

В 1980 році General Motors реалізує власний інтерфейс і протокол для тестування модуля керування двигуном (ECM) на лінії складання автомобіля. Протокол діагностичного зв'язку збірки (ALDL) транслюється зі швидкістю 160 біт/с.

В 1986 році з’являється оновлена версія протоколу ALDL, яка передається на рівні 8192 біт/с з напівдуплексною UART-сигналізацією.

В 1988 році Товариство автомобільних інженерів (SAE) рекомендує стандартизований діагностичний роз’єм і набір діагностичних тестових сигналів.

Появляється вимога, щоб всі нові автомобілі мали деякі базові можливості OBD. Ці вимоги, як правило, називають «OBD-I», хоча ця назва не застосовується до введення OBD-II.

З 1996 року специфікація OBD-II стає обов’язковою для всіх автомобілів, вироблених у Сполучених Штатах для продажу в США.

З 2001 року і Європейський Союз робить EOBD обов'язковою для всіх бензинових транспортних засобів, що продаються в Європейському Союзі, а з 2003 року ця вимога розповсюджується й на всі дизельні автомобілі.

З 2008 року специфікація OBD починає вимагатися й в Китаї.

На цей час існує багато засобів для діагностики автомобілів. Кожен із засобів має власні переваги та недоліки, в залежності від яких, формується різноманітна аудиторія користувачів. Але с точки зору звичайного користувача, методика використання будь-якого діагностичного приладу однакова:

1. Під’єднати апаратний засіб до відповідного порту в автомобілі.
2. Підключитися до нього за допомогою мережевого сервісу (Bluetooth або Wi-fi).
3. Використати додаток, який отримує дані з автомобіля та формує результати у зручній та зрозумілій для користувача формі (мається на увазі переклад байтів команд у декодований текст та визначення одиниць вимірювання).

В більшості ситуацій людина використовує діагностичне обладнання, щоб власноруч визначити причину несправності у роботі автомобіля. Але кваліфікація власника найчастіше не дозволяє робити висновки щодо стану тих чи інших компонентів авто.

Мета цієї дипломної роботи – знайти баланс між користувацькою, але не ефективною діагностикою та необхідністю витрачати багато часу та коштів на відвідування автомобільного сервісу, та представити програмний засіб, який дозволить передавати діагностичні дані від власника авто до сервісу. Це дозволить зробити перші висновки щодо технічного стану автомобіля, не виходячи з нього.

1.2 Комп’ютерна діагностика автомобіля OBD-II

OBD-II є поліпшенням в порівнянні з OBD-I з точки зору як можливостей, так і стандартизації. Стандарт OBD-II визначає тип діагностичного роз’єму та його роз’єм, наявні протоколи електричної сигналізації та формат обміну повідомленнями. Він також надає список параметрів транспортного засобу для моніторингу разом з тим, як кодувати дані для кожного.

Стандарт OBD-II містить перелік стандартизованих діагностичних кодів. В результаті цієї стандартизації, один пристрій може використовувати

бортовий комп'ютер для цих параметрів у будь-якому транспортному засобі. Стандартизація OBD-II спростила діагностику обладнання для викидів.

Діагностичні коди OBD-II складаються з 4 цифр, перед якими стоїть буква: P – для двигуна і трансмісії (силова передача), B – для корпусу, C – для шасі і U – для мережі. Виробники можуть також додавати власні параметри даних до їх конкретної реалізації OBD-II, включаючи запити даних в реальному часі, а також коди несправностей.

1.3 Ідентифікатори параметрів бортової діагностики

Ідентифікатори параметрів бортової діагностики (PIDs OBD-II) – це коди, що використовуються для запиту даних з транспортного засобу.

Стандарт SAE визначає багато кодів OBD-II. Виробники також визначають додаткові PID, які специфічні для їх транспортних засобів. Багато мотоциклів також підтримують OBD-II PID.

Існує 10 діагностичних режимів, описаних в останньому стандарті OBD-II SAE J1979:

01 – показати поточні дані;

02 – показувати дані стоп-кадру;

03 – показати збережені діагностичні коди проблем;

04 – очистити діагностичні коди проблем і збережені значення;

05 – результати випробувань, моніторинг датчиків кисню;

06 – результати випробувань, моніторинг інших компонентів/систем;

07 – показати діагностичні коди неполадок, які очікують на розгляд;

08 – керування роботою бортового компонента/системи;

09 – запит інформації про транспортний засіб;

0a – постійні діагностичні коди несправностей (dtcs).

Виробники автомобілів не зобов'язані підтримувати всі режими, але кожен виробник може визначити додаткові режими.

1.4 Аналіз існуючих рішень

На ринку представлено багато видів програмного забезпечення для бортової діагностики автомобіля. Вони дозволяють контролювати різномані-

тні вузли, так як двигун, трансмісія, паливна система, датчики температури та інші.

Забезпечення доступу до параметрів автомобіля покладається на OBDII-адаптер, який підтримує декілька протоколів передачі даних. Цей протокол визначається в залежності від виробника транспортного засобу та його року випуску.

Звідси, стек діагностичних даних може відрізнятися в залежності від автомобіля, до якого підключається адаптер. При цьому розробник програмного забезпечення не впливає на кількість даних, які отримує адаптер від автомобіля. Тому функціонал існуючих програмних засобів для аналізу відрізняється тим, які дані розробник бере з адаптеру та як їх використовує у своєму додатку.

Розглянемо найбільш популярні програми для аналізу даних, що надходять з OBDII-адаптеру. Серед них:

- Car Scanner;
- Вася Диагност;
- ScanMaster-ELM V2.1 RU;
- Carista OBD2;
- EOBD Facile - Car Diagnostic.

Всі вони мають як переваги, так й недоліки, які найчастіше з’являється в наслідок не сумісності з різними марками та модифікаціями автомобілів, методами підключення до автомобілів та передачі даних, ціною, мовною підтримкою та іншими чинниками.

Порівняння можливостей декількох програмних засобів для діагностики автомобіля представлено в таблиці 1.1.

Таблиця 1.1 – Порівняння можливостей різного програмного забезпечення

ПЗ	Car Scanner	Вася Диагност	ScanMaster-ELM	Carista OBD2	EOBD Facile - Car Diagnostic
Можливості	-графіки з відображенням показників автомобіля, - робота з декількома блоками -читання та скидання помилок, -показники витрат палива, -автоматичне визначення OBD протоколу.	-автопошук помилок, - кодування і програмування блоків управління, - побудова графіків, - видалення кодів помилок. - відомості про блоки управління, - тести виконавців.	- моніторинг параметрів двигуна, - побудова графіків залежностей, - перегляд стоп-кадрів систем, - перегляд і стирання помилок, - відображення статусу паливної системи.	- базова безкоштовна діагностика, -платна поглиблена діагностика, -пошук та скидання помилок.	- видалення кодів помилок (тільки преміальна версія), - відображення коду невизначеності OBD2 конкретних виробників, - відображення в режимі реального часу датчики автомобіля і можлива запису в файл.

Продовження таблиці 1.1

ПЗ	Car Scanner	Вася Диагност	ScanMaster-ELM	Carista OBD2	EOBD Facile - Car Diagnostic

Плат-форми	iOS; Android; Windows; Windows Phone	Windows 2000; Windows XP; Windows Server 2003; Windows Vista; Windows Server 2008; Windows 7, 8, 10; Windows Server 2008	Windows 2000; Windows XP; Windows 2003; Windows Vista 7, 8, 8.1	iOS; Android	iOS; Android; Windows Vista 7, 8, 8.1, 10; Mac OS X
Суміс-ність адапте-ра	Wi-Fi, Bluetooth OBD2 ELM327	VAG-COM 409.1; (з ELM 327 адапте-ром не працю є)	ELM327 Bluetooth; ELM327 USB	Bluetooth LE	Wi-Fi; Bluetooth OBD2 ELM327, ELM320, ELM322, ELM323
Розмір	124,8 МБ	6,71 МБ	482,97 КБ	29,7 МБ – iOS; 6,1 МБ – Android	51,5 МБ

Продовження таблиці 1.1

ПЗ	Car Scanner	Вася Диаг-ност	ScanMaster-ELM	Carista OBD2	EOBD Facile - Car Diagnostic

Мова інтер-фейсу	російська; англійська; угорська; іспанська; італійська; китайська; корейська; німецька; польська; турецька; французька; чеська.	Російська.	російська; англійська.	англійська; німецька; японська; російська; іспанська.	Більше ніж 24 мови, вклю-чаючи анг-лійську, укра-їнську та ро-сійську.
Вироб-ник	Росія	Росія	ELM Electronics (Канада)	Prizmos Ltd. (Болгарія)	Outils OBD Facile (Фран-ція)
Ціна	5-6 USD		220 USD	20 USD (ада-птер); 40 USD в год	18-36 USD

1.5 Постановка задачі

Головна мета цього дипломного проекту – створити діагностичне програмне забезпечення для звичайного автомобіліста, який не повинен аналізувати діагностичні дані, але може отримати інформацію про стан власного автомобіля без необхідності витрачати час на відвідування автосервісу.

Проаналізувавши всі доступні продукти на ринку, врахувавши всі їхні переваги та недоліки, були виділені основні апаратні та програмні вимоги для реалізації свого додатка.

Аналіз існуючих рішень довів необхідність створення подібного рішення та його конкурентоспроможність.

Вимоги, виконання яких необхідне для створення продукту:

- максимальне усунення затримок при передачі кодів з автомобіля у смартфон користувача;
- інтуїтивно зрозумілий інтерфейс додатку;
- економія ресурсу батареї смартфону;
- зрозуміле обґрунтування висновку щодо стану автомобіля.

Задача складається з проектування архітектури мобільного додатку, визначенні стеку технологій, розробці програмних модулів та комплектуванні їх до складу єдиного проекту.

2. ОСОБЛИВОСТІ РОЗРОБКИ ДІАГНОСТИЧНОГО МОБІЛЬНОГО ДОДАТКА ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ IOS

2.1 Аналіз особливостей даної розробки

Головна особливість даної розробки – необхідність компонувати різні програмні модулі для їх коректної роботи. Два головні модулі – це модуль отримання даних з OBDII-адаптеру (далі OBD Service) та модуль передачі декодованих даних на віддалений сервер (далі Remote Service).

Для мінімізації тестування та забезпечення гнучкості додатку, основні модулі повинні функціонувати незалежно один від іншого.

Кожен модуль виконує лише одну конкретну задачу, але не бере залишкової відповідальності. Наприклад, OBD Service не може відображати діагностичні дані на екрані мобільного пристрою або зберігати їх до локального сховища. Єдине його призначення – передати дані з адаптеру до додатку.

Такий підхід до розробки не новий і базується на виконанні принципів SOLID – це принципи об’єктно-орієнтованого програмування, які описують основні п’ять критеріїв, що забезпечують можливість підтримувати розроблену систему.

Опис критеріїв SOLID представлений у таблиці 2.1:

Таблиця 2.1 – опис критеріїв SOLID

Літера	Абревіатура	Назва	Опис
S	SRP	Принцип єдиного обов'язку (Single responsibility principle)	Визначає, що клас об’єкту має виконувати лише одну задачу, яку він інкапсулює
O	OCP	Принцип відкритості/закритості (Open/closed principle)	Визначає, що класи, функції та змінні повинні бути закритими для зміни, але відкритими для розширення.

Продовження таблиці 2.1

Літера	Абревіатура	Назва	Опис
L	LSP	Принцип підстановки Лісков (Liskov substitution principle)	Клас наслідник повинен доповнювати, а не заміщати поведінку базового класу
I	ISP	Принцип розділення інтерфейсу (Interface segregation principle)	Класи не повинні містити методів, які вони не використовують.
D	DIP	Принцип інверсії залежностей (Dependency inversion principle)	Модулі вищого рівня не повинні залежати від модулів нижчого рівня. Абстракції не повинні залежати від деталей реалізації. Деталі реалізації повинні залежати від абстракцій.

Аналіз переваг та недоліків принципів SOLID приведено в таблиці 3.

Таблиця 3 – аналіз переваг та недоліків принципів SOLID

Принципи SOLID	Переваги	Недоліки
SRP	Зменшення кількості дублюючого коду	Зростання кількості класів призводить до ускладнення системи, але гарна структуризація врівноважить цей недолік
	Зменшення ймовірності змін у вже затвердженому класі	
	Відповідність назв клас на опису їх функціональності	
OCP	Легкість додавання нового функціоналу без необхідності дублювання та редагування вже існуючого коду	Велика кількість класів-наступників призводить до зростання кількості класів та ускладнення системи

Продовження таблиці 2.2

Принципи SOLID	Переваги	Недоліки

LSP	Дозволяє створювати гнучку систему наслідування	Цей принцип не збільшує кількість елементів системи (класів, функцій тощо), тому можна визначити, що він не має недоліків
ISP	При об’явлені нової реалізації інтерфейсу не треба реалізувати не потрібні (пусті методи)	Велика кількість інтерфейсів призводить до ускладнення системи
DIP	Дозволяє створювати інтуїтивно зрозумілі ієрархії наслідування від абстрактних класів до реалізацій	Цей принцип не збільшує кількість елементів системи (класів, функцій тощо), тому можна визначити, що він не має недоліків

Проведений аналіз доводить доцільність слідування принципам SOLID при розробці дипломного проекту.

2.2 Архітектурні аспекти мобільного додатку

Оскільки програмний продукт, що розробляється повинен підтримувати можливість подальшого впровадження нових функцій та модифікування вже існуючих, виникає необхідність розробити архітектурну структуру додатку.

Існує безліч архітектурних рішень для мобільних додатків. Далі представлені найбільш уживанні з них:

- MVC (Modal View Controller);
- MVP (Modal View Presenter);
- MVVM (Modal View ViewModel);
- VIPER (View Interactor Presenter Entity Router);
- VIP (View Interactor Presenter), також відомий як Clean Architecture.

Слід зазначити ознаки гарної архітектури:

- збалансований розподіл обов'язків між сутностями з жорсткими ролями;

- можливість тестування - зазвичай випливає з першої ознаки;
- простота використання і низька вартість обслуговування.

Розподіл зменшує навантаження на мозок, коли ми намагаємося з'ясувати, як працює та чи інша сутність. Таким чином, найпростіший спосіб зменшити складність полягає в поділі обов'язків між декількома сутностями за принципом єдиної відповідальності. (див. перший принцип SOLID)

Можливість тестування визначає, наскільки легко буде писати юніт-тести, а найчастіше - чи можна їх писати взагалі. Тести дозволяють розробникам виявляти помилки, які виникають на етапі виконання додатку та оберігають від необхідності стикатися с критичними помилками вже на пристроях користувачів, коли виправлення було б можливо якнайменш через тиждень.

Варто відзначити, що кращий код - це код, який ніколи не був написаний. І чим менше у вас коду, тим менше помилок. Тому бажання писати менше коду зовсім не говорить про те, що розробник лінується. А вибираючи найрозумніше рішення, потрібно завжди враховувати вартість його підтримки.

Проаналізуємо відомі архітектури окремо.

MVC

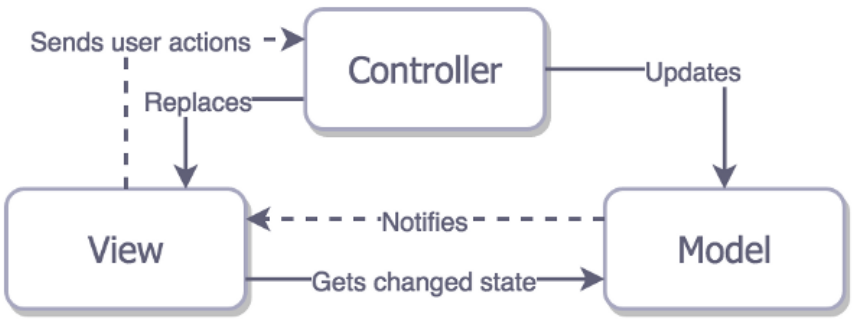


Рисунок 2.1 – Структура архітектури MVC

У традиційному MVC, який ще не був запропонований як офіційна архітектура від Apple, View не зберігає свого стану. Controller «рендерить» View кожен раз при змінах Model. Наприклад, модальне вікно буде повністю перевантажуватись, коли достатньо лише завантажити новий текст повідомлення. Хоча можна реалізувати традиційний MVC у iOS додатку, це не має сенсу через архітектурної проблеми: все три сутності тісно пов'язані, кожна з

них знає про двох інших. Це сильно знижує можливість повторного використання кожного з елементів. З цієї причини традиційний MVC не є доцільним варіантом при виборі архітектури.

Cocoa MVC

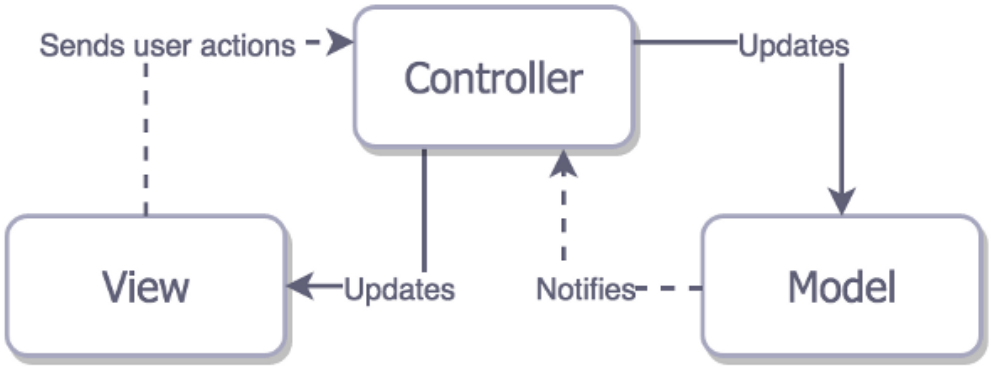


Рисунок 2.2 – Структура архітектури Cocoa MVC

Controller є зв'язуючою ланкою між View і Model, отже, дві останніх не знають про існування один одного. З першого погляду, все зрозуміло. Але існує і інша інтерпретація цієї архітектури як Massive View Controller.

Cocoa MVC заохочує вас писати Massive View Controller, тому що контролер настільки залучений в життєвий цикл View, що важко вважати його окремою сутністю. У більшості випадків вся відповідальність View полягає в тому, щоб відправити дії до контролера. В підсумку все закінчується тим, що View Controller стає делегатом і джерелом даних.

Складується враження, що Cocoa MVC є досить поганим вибором архітектурного партерну. Але його оцінка з точки зору ознак хорошої архітектури, визначених вище:

- розподіл: View і Model насправді розділені, але View і Controller тісно пов'язані;
- тестування: через поганий розподіл можна реалізувати unit-тести лише для Model;
- простота використання: найменша кількість коду у порівнянні з іншими патернами.

Cocoa MVC – це розумний вибір, якщо розробник не планує витрачати багато часу на більш складні патерни, а проект розробки не поділяється на багато компонентів.

MVP

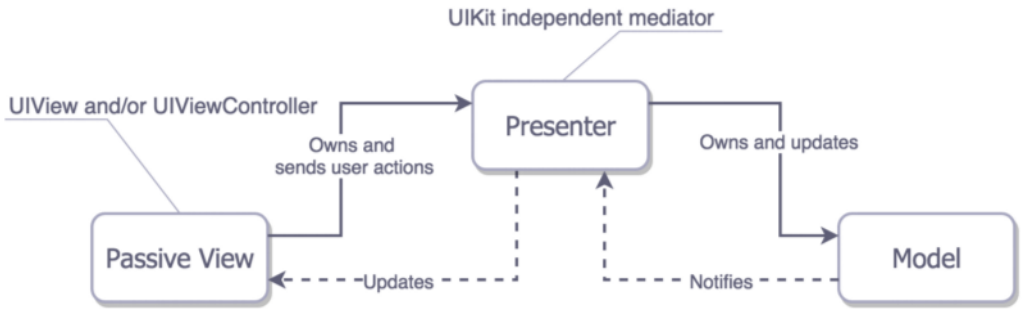


Рисунок 2.3 – Структура архітектури MVP

Посередник в MVP – Presenter, не має відношення до життєвого циклу View Controller. View може бути легко замінена Mock-об'єктами для тестування, тому в Presenter немає layout-коду, але він відповідає за оновлення View.

З точки зору MVP, підкласи UIViewController насправді є View, а не Presenter. Ця різниця забезпечує гарні можливості для тестування, але знижується швидкість розробки, тому що дані і події пов'язуються вручну саме між View і Presenter.

Давайте подивимося на ознаки гарної архітектури для MVP:

- розподіл: велика частина відповідальності розділена між Presenter і Model, а View лише комутує їх;
- тестування: ми можемо перевірити більшу частину бізнес-логіки завдяки бездіяльності View;
- простота використання: кількість коду в два рази більше в порівнянні з MVC, але в той же час ідея MVP дуже проста.

MVP в iOS означає чудову можливість для написання unit-тестів і багато коду, тому цей архітектурний патерн можна використовувати для виконання цього дипломного проекту.

MVVM

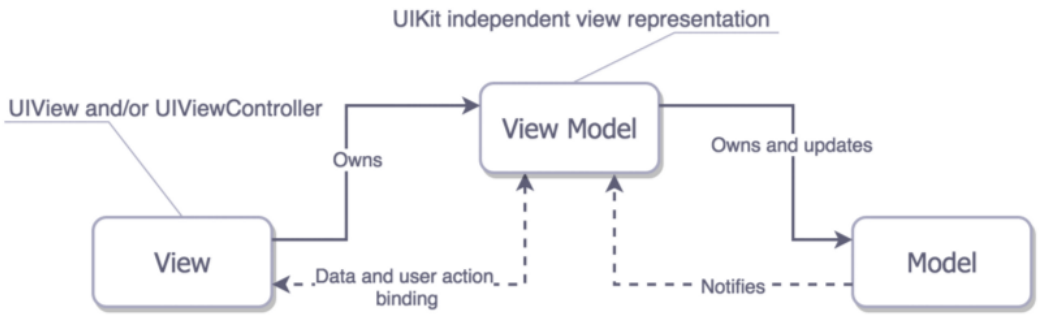


Рисунок 2.4 – Структура архітектури MVVM

MVVM є найновішим із патернів. View Model – це незалежне від UIKit уявлення View. View Model викликає зміни в Model і самостійно оновлюється з уже оновленої Model. І так як Біндінг відбувається між View і View Model, то перша, відповідно, теж оновлюється.

І знову повернемося до оцінки ознак хорошої архітектури:

- розподіл: в MVVM View має більше обов'язків, ніж View з MVP. Тому що перша оновлює свій стан з View Model за рахунок установки прив’язок між View та Viw Model, тоді як друга направляє всі події в Presenter і не оновлює себе власноруч (це робить Presenter);
- тестування: View Model не знає нічого про View, це дозволяє нам з легкістю тестувати її;
- простота використання: той же обсяг коду, як в нашому прикладі MVP, але в реальному додатку, де вам доведеться направити всі події з View в Presenter і оновлювати View вручну, MVVM буде набагато стрункішою.

MVVM є дуже привабливим патерном, так як він поєднує в собі переваги вищезазначених підходів і не вимагає додаткового коду для оновлення View через зв'язки між View та View Model. Проте, тестування все ще знаходиться на хорошому рівні.

VIPER

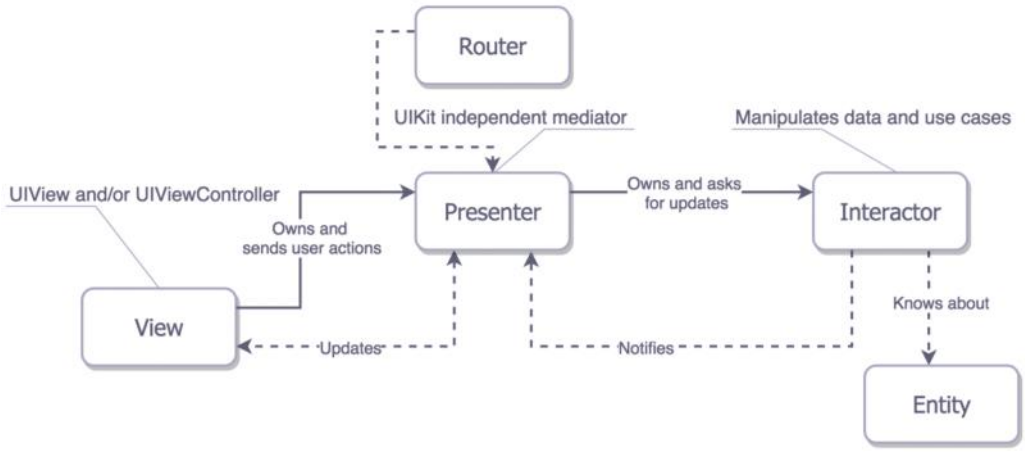


Рисунок 2.5 – Структура архітектури VIPER

VIPER робить ще один крок в бік розподілу обов’язків і замість трьох шарів пропонує п'ять.

Interactor містить бізнес-логіку, пов'язану з даними (Entities): наприклад, створення нових екземплярів сутностей або отримання їх з сервера. Для цих цілей використовуються Сервіси, які розглядаються швидше як зовнішні залежності, а не як частина модуля VIPER.

Presenter містить бізнес-логіку, пов'язану с UI (але UIKit-незалежну), викликає методи в Interactor.

Entities – прості об'єкти даних.

Router несе відповідальність за переходи між VIPER-модулями.

Повернемося до ознак гарної архітектури:

- розподіл: безсумнівно, VIPER є лідером у розподілі обов'язків;
- тестування: чим краще розподіл, тим легше написання тестів;
- простота використання: перші два переваги йдуть за рахунок величезної збільшення кількості коду.

Використання VIPER може бути обумовлено у дуже складних проектах, які об’єднують якнайменш 10 модулів, та потребують постійної модифікації без зайвих витрат часу. Гнучка система дозволяє легко долучати або видаляти функціонал, але розробка основного каркасу потребує написання багатьох інтерфейсів та їх реалізацій.

Аналіз кількох основних архітектурних патернів показав, що не існує одного ідеального рішення для кожного проекту. Вибір архітектури є питанням зважування компромісів у конкретній ситуації.

Найкращим варіантом є поєднання декількох архітектур в одному додатку. Більш прості екрани можуть проектуватися за MVC, а десь доцільніше розділити відповідальність між View та Presenter (або View Model).

2.3 Проектування та організація структури додатку

Для визначення структури додатку треба виділити декілька термінів, що стосуються організації робочого простору для додатків, що розробляються для платформи iOS.

На вершині ієрархії додатку стоїть Робоча Область (Workspace). Вона містить, власне, проект, який розробляється, а також залежності до інших проектів, бібліотек та зовнішніх утиліт.

Для імпорту бібліотек можна використовувати стандартні зв'язки, додаючи код фреймворку до існуючого проекту, але цей спосіб має декілька недоліків. Якщо у розробника використовуваної бібліотеки з'явиться необхідність оновити її або справити помилки, нова версія не буде доступна всім тим, хто використовував цю бібліотеку раніше. Це примушує розробника вручну встановлювати нову версію фреймворку. С цього випливає наступна проблема: як визначити момент часу, коли стала доступна нова версія? Звісно можна перевіряти це вручну, але не виключно, що встановлено багато різних фреймворків. Це може змусити розробника робити перевірки версій так часто, як він перевіряє електрону пошту, або робить комміт до системи контролю версій. Звичайно, такі перевірки витрачають зайвий час.

Необхідність відстеження актуальних версій фреймворків призвела до появи незамінного інструменту як менеджер залежностей рівня додатків – Cocoa Pods.

Cocoa Pods – це дуже потужний інструмент, яким має володіти кожний iOS-розробник. За допомогою нього ми можемо легко, швидко і просто

підключати різні бібліотеки та утиліти, які значно полегшують нам життя при розробці додатків.

Podfile – це специфікація, в якій позначені всі залежності, необхідні для нашого проекту. Podfile завжди створює неявну ціль (target). Цей файл не має розширення і повинен знаходитеся в одній папці з файлом .xcodepoj проекту.

Приклад Podfile:

```
platform :ios, '11.0'
target 'myapp' do
      pod 'Reachability', '~> 3.1.1'
      pod 'sqlite3'
end
```

Спочатку вказується мінімальна версія iOS, яка буде підтримувати розроблений додаток. В полі «target» вказується назва додатку, а в блоці «do-end» список бібліотек, які треба встановити. За бажанням, можна вказувати параметр версії. Це може знадобитися, тільки якщо актуальна версія містить критичні помилки та не може використовуватися у проекті.

Cocoa Pods зчитує Podfile обираючи необхідну версію кожної бібліотеки в залежності від параметрів додатку та пристроїв, на яких додаток повинен запускатися.

Для запуску менеджера залежностей треба виконати команду «pod install» із термінала Mac, знаходячись у директорії проекту.

Класична структура мобільного додатка для платформи iOS представлена на рис. 2.6:

Рисунок 2.6 – Структура додатку для мобільної платформи iOS

2.4 Технологія

2.4.1 Мова програмування

Мобільні додатки для платформи iOS можуть розроблятися використовуючи різноманітний стек технологій. Існує два шляхи, що залежать від вибору мови програмування – Objective-C або Swift.

Objective-C – це перевірена часом мова, яка не мала аналогів до 2014 року, коли з'явилась перша версія Swift. Objective-C має C-подібний синтаксис та дозволяє вручну керувати пам'яттю. Це гарне рішення для додатків, які були створені ще для iOS 7.0 та потребують підтримки та оновлення.

Але майбутнє за Swift. Ця мова була створена компанією Apple виключно для розробки додатків під iOS. В порівнянні з Objective-C був значно спрощений синтаксис. Наприклад, більше не треба писати на кожен клас окремий заголовок-інтерфейс, що значно прискорює розробку. Говорячи о швидкодії, Swift також бере верх. Його новітня система автоматичного контроль за пам'яттю, яка називається ARC (Automatic Reference Counting), рахує активні посилання до кожного об'єкту – якщо кількість посилань дорівнює нулю, об'єкт буде видалений із пам'яті. Також Swift підтримує компіляцію

файлів Objective-C. Це дає змогу використовувати перевірені часом фреймворки у нових проектах.

Можна з повною впевненістю сказати, що Swift - це майбутнє. Все пізнається в порівнянні і, лише відпрацювавши певну кількість проектів, можна стверджувати, чи зручна вибрана мова, і чи може вона ефективно виконувати задачі, які стоять перед розробником.

Незважаючи на песимізм і консерватизм багатьох компаній, заснований на безлічі заяв про те, що Swift сируватий і має безліч недоліків, ця мова активно розвивається. На даний момент остання версія 5.0 є рішенням 99% скарг на попередні версії, а також має нову систему компіляції, яка збирає проект на 30-40% швидше, ніж передній Swift 4.

2.4.2 Протокол-орієнтований підхід до розробки

У 2015 році, коли мова програмування Swift була ще не дуже популярною серед розробників, Apple заявила що їх мова – перша протокол-орієнтована мова програмування.

Використання протоколів не є обов'язковим, але значно покращує розуміння архітектурних взаємодій та робить програмний код чистішим.

Існує декілька шлях використання протоколів:

- протокол як тип - аналогічний поняттю інтерфейс з ООП. Служить для опису функціональності об'єкту. Може використовуватися в якості типу властивості, як тип результату функції, типу елемента гетерогенної колекції. Через обмеження мови, протоколи, які параметризуються зв'язними типами не можуть використовуватися в якості типів об'єктів;
- протокол як шаблон типу служить для опису функціональності об'єкту, але використовується як вимога до типу в узагальнених функціях.

Чіткої межі, в якому випадку використовувати протокол як тип, а в якому – як обмеження на тип, немає. Іноді потрібно використовувати протокол в обох сценаріях.

Випадки використання:

- класи, які надають функціональність для більш високих шарів додатку і передаються класам-споживачам в якості залежності - це сервіси, API-клієнти, призначені для користувача настройки тощо. В цьому випадку зручніше використовувати протокол як тип – його можна буде зареєструвати в IOC контейнері;
- протокол з описом математичних операцій, наприклад порівняння, додавання, конкатенація і подібні речі. У цьому випадку зручно скористатися Self-requirement (коли в функції або властивості протоколу використовується псевдотіпов Self), щоб уникнути небезпечного приведення і використання різних типів, коли операція допускає параметри тільки одного типу (і Int і String в Swift відповідають протоколу Equatable, але якщо спробувати перевірити їх на рівність між собою, компілятор видасть помилку, оскільки оператор порівняння вимагає, щоб параметри були одного типу).

Trait (типаж) – сутність, що надає набір реалізованої функціональності, є набором будівельних блоків для класів та структур.

Ця концепція розроблена для заміни успадкування в класичній теорії ООП. Також клас використовується для створення екземплярів, тому його функціональність повинна бути закінченою. Ці дві ролі класу можуть конфліктувати одна з іншою. Плюс до цього кожен клас має певне місце в ієрархії класів. В якості вирішення пропонується використовувати в ролі одиниць перевикористання коду більш прості сутності – traits, а класам буде відведена роль сполучного елемента для задіяння логіки, успадкованої від traits.

У Swift ця концепція реалізується за допомогою протоколів та розширення протоколів. Щоб «підключити» визначені для протоколу функції потрібно додати створюваному типу відповідність цього протоколу - відпадає необхідність створення базового класу для наслідування функціональності.

Властивості типажів:

- типаж надає набір методів, що реалізують поведінку. Методи додаються за допомогою розширення протоколів;
- типаж – це набір методів, які служать параметрами для забезпечення поведінки. Сигнатури цих методів прописуються в протоколі;
- типаж не можуть мати змінних для зберігання стану або розрахункових властивостей оскільки реалізуються у розширеннях. Методи, що вказуються у типажу, не мають прямого доступу до полів класу. Таким чином методи розширення не мають прямого доступу до даних, а тільки через get властивості;
- класи і типажі можуть бути складені з інших типажів. Конфлікти методів повинні бути вирішені програмістом. Класи можуть ставитись у відповідність протоколам, а протоколи підтримують успадкування інших протоколів.

Як ми бачимо, протоколи повністю відповідають концепції типажів, описаної задовго до появи мови програмування Swift.

Протоколи можуть використовуватись в якості маркерів. Протокол в даному випадку описує не функціональність класів, а то, що якийсь клас підтримує класи, що реалізують протокол, як тип результату запиту.

Розширення – це засіб мови програмування, який дозволяє додати функціональність до існуючого класу або протоколу.

За допомогою розширень ми можемо:

- додати метод в протокол – цей метод буде доступний (в межах області видимості) для використання як всередині класів, які відповідають цим протоколом, так і для їх споживачів;
- написати стандартну реалізацію методу протоколу. Якщо клас містить реалізацію цього методу, то буде використана вона, замість стандартної;
- додати класу або структурі відповідність протоколу.

Не можливо встановити відповідність одного протоколу до іншого. Якби це було можливо то, маючи протокол P1, який відповідає протоколу P2,

всі класи, які реалізують протокол P1 стали б автоматично відповідати і протоколу P2. Для вирішення цієї проблеми можна написати розширення для протоколу P1, в якому містилась реалізація методів протоколу P2, після чого ми можемо додавати відповідність P2 класам, відповідним P1, без реалізацій методів.

Для обмеження використання розширених методів існують спеціальні оператори – обмеження на тип.

Підтримуються наступні: відповідає протоколу, успадковується від класу, має тип. Обмеження використовуються для визначення набору методів, які має узагальнений тип. Якщо аргументи не задовольняють обмеженням типу, компілятор повідомить про помилку.

Обмеження можна застосовувати:

- обмеження на типи параметрів у визначенні узагальненої функції;
- обмеження на зв'язані типи у визначенні протоколу;
- обмеження на доступність методів розширення;
- визначення умовної відповідності протоколу.

Оскільки обмеження на зв'язані типи можна вказувати і в протоколі, і для методів, в які передається протокол, і для розширення протоколів, то виникає питання – куди додавати розширення:

1. Якщо протокол повинен мати одну реалізацію - варто використовувати конкретні типи замість асоційованих.

2. Якщо протокол повинен мати кілька реалізацій (з урахуванням мок-типів, які використовуються для тестів) - зручніше помістити їх в сам протокол, щоб не дублювати в місця використання цього протоколу.

3. Якщо можливе перевикористання протоколу – протокол повинен містити тільки ті обмеження, без яких існування протоколу не має сенсу і на яких побудована основна логіка.

Отже, підсумовуючи огляд теорії протоколів у мові програмування Swift, можна виділити кілька правил для написання чистого коду:

1. Починати не з реалізації, а з опису протоколу (опис функціональності, яку об'єкт буде зобов'язаний надати споживачам). Описувати перевизначену логіку в протоколах, а не в класах. Використовувати протокол як одиницю коду, а клас - як місце для унікальної логіки.

2. Використовувати композицію через розширення. Коли потребується різний набір функціональності, цю функціональність можна або розбити на частини і організувати ієрархію класів, де кожен клас успадковує власну функціональність і додає свою. Або розбити на незв'язні класи, екземпляри яких можна використовувати в сполучному класі. Використовуючи можливість додати відповідність протоколу через розширення ми можемо використовувати композицію не вдаючись до створення допоміжних класів.

3. Замість успадкування використовувати протоколи. Протоколи дозволяють досягти подібності множинного успадкування. Якщо ваш клас містить багато логіки, варто спробувати розбити його на окремі набори функціональності, які можна винести в протоколи.

4. Включати в протоколи методи, які можна перевизначити. Якщо з'явилася необхідність перевизначити загальний метод, визначений у розширені, то перенесіть сигнатуру цього методу в сигнатуру протоколу. Інші класи не доведеться коректувати, тому що вони продовжать використовувати метод із розширення.

2.4.3 Bluetooth

Комунікація із OBDII-адаптером має ключове значення для розробки програмного засобу. Для цього буде використовуватись бездротова технологія Bluetooth. Оскільки зменшення енергоспоживання та збільшення швидкості передачі є пріоритетними задача на етапі отримання даних, було прийнято рішення використовувати технологію Bluetooth 4.0, яка також відома як Bluetooth Low Energy (LE).

Bluetooth LE був представлений у 2009 році, а перший смартфон, що отримав новітній стандарт – iPhone 4S. Зараз технологія мінімального енер-

госпоживання використовується у всіх новітніх гаджетах, що потребують бездротового зв’язку.

В таблиці 2.3 представлена порівняльні характеристики класичного Bluetooth та Low Energy:

Таблиця 2.3 – порівняльна характеристика Bluetooth технологій

Технічна специфікація	**Класичний Bluetooth**	**Bluetooth LE**
Радіочастота	2.4 ГГц	2,4 ГГц
Відстань	100 м	>100 м
Швидкість передачі даних	1-3 Мб/с	1 Мб/с
Пропускна спроможність	0,7-2,1 Mb/s	0,27 Mb/s
Безпека	64/128-bit шифрування	128-bit AES с Counter Mode CBC-MAC шифрування
Надійність	Адаптивна перебудова частоти	Адаптивна перебудова частоти, Lazy Acknowledgement, 24-битовий надлишковий циклічний код, 32-разрядная перевірка цілісності повідомлення
Затримка	100 мс	6 мс

Продовження таблиці 2.3

Технічна специфікація	**Класичний Bluetooth**	**Bluetooth LE**
Мінімальний час передачі даних	100 мс	3 мс
Державне регулювання	В усьому світі	В усьому світі
Орган з сертифікації	Bluetooth SIG	Bluetooth SIG
Передача голосу	Так	Ні
Топологія мережі	Scatternet	Scatternet
Споживана потужність	1 Вт	От 0,01 Вт до 0,5 Вт (в за-

		лежності від варіантів використання)
Максимально споживаний струм	<30 мА	<15 мА
Виявлення служби	Так	Так
Визначення конфігурації	Так	Так

Специфікація Bluetooth 4.0 визначає дві бездротові технології:

- BR / EDR (класичний Bluetooth);
- BLE (Bluetooth Low Energy).

Пристрої, в яких застосований BLE, можуть бути як дворежимні - Bluetooth Smart Ready, сумісні з класичними Bluetooth-пристроями, так і однорежимні – Bluetooth Smart.

Розглянемо основні блоки Bluetooth-пристрою, які показані на рис. 2.7:

- додаток – реалізує корисну для кінцевого користувача логіку роботи;
- провідний пристрій або хост – реалізує верхні рівні стека протоколів Bluetooth;
- контролер – реалізує нижні рівні стека протоколів Bluetooth.

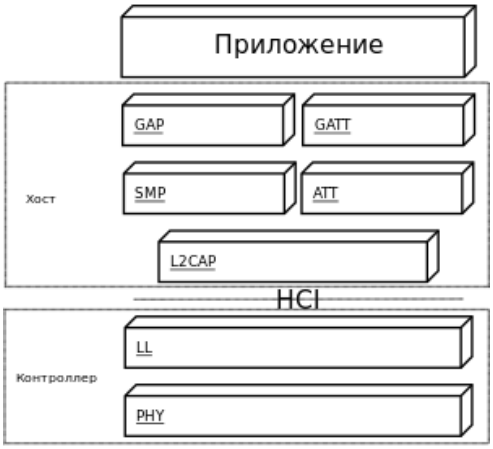


Рисунок 2.7 – Структура Bluetooth-пристрою

2.4.4 Core Bluetooth

Core Bluetooth – це програмна реалізація роботи з Bluetooth 4.0, яка входить до складу базових бібліотек від Apple.

Core Bluetooth був представлений Apple досить давно, в iOS 5. Apple почала роботу по впровадженню BLE в свої пристрої набагато раніше Android, що призвело до зростання популярності технології. Безліч розробників використовують цей фреймворк в своїх додатках, за великим рахунком це – всього лише обгортка, так як самі по собі протоколи BLE досить складні.

Для обміну інформацією з адаптером необхідно визначити основні принципи роботи з Core Bluetooth.

Бібліотека виконує пошук пристроїв, що знаходяться поблизу. Кожен пристрій називається периферал. У кожного периферала є сервіси, їх може бути як дуже багато, і кожен з них має власні характеристики. Можна розглядати периферал як сервер. З усіма наслідками, що випливають: іноді він відключається, іноді потрібен час на передачу даних, а іноді ці дані і зовсім не приходять.

Кожна характеристика містить якесь значення, тип і так далі. Щоб працювати з CoreBluetooth - не потрібно знати що робить кожна з них, найголовніше – знайти характеристики для зчитувати та запису даних.

Окремо слід зазначити, що постійний обмін даними між адаптером та смартфоном не повинен виконуватися на головному потоці додатку. Це може призвести до блокування екрану, тоді користувач не зможе виконувати ніякі дії паралельно. Зі сторони це буде схоже на відмову роботи, що зменшить репутацію додатку.

Для коректної роботи додатку слід виконувати дії по пошуку, підключені, передачі даних та відключені периферaлів на асинхронному потоці, асинхронно переключаючись на основний тільки для оновлення інтерфейсу користувача, коли нові дані вже отримані.

2.4.5 Вибір OBDII-адаптеру

OBDII стандарт – це набір правил і вимог, яким повинен відповідати автомобіль для того, щоб можна було діагностувати несправності, пов'язані з можливими шкодою екології у кожного автомобіля, який відповідає цьому

стандарту. Стандарт включає в себе вимоги до апаратної і програмної частини автомобіля.

Важливо, що у вимогах до апаратної частини є обов'язкова наявність стандартного діагностичного роз'єму в формі трапеції з 16 контактами. Саме до цього гнізда підключається адаптер ELM327 – найпоширеніша реалізація OBDII.

Вимоги до програмної частини включають в себе обов'язкову підтримку автомобілем одного з протоколів:

- SAE J1850 VPM;
- SAE J1850 PWM;
- ISO 9141-2 / ISO 14230-4 KWP;
- ISO 15765-4 CAN;
- SAE J1939 CAN.

Крім того, стандарт регламентує перелік можливих запитів і порядок декодування їх відповідей. Наприклад, для того, щоб дізнатися про поточні оберти двигуна, треба виконати запит 010C. Кожен біт у відповіді позначатиме 0,25 об./хв. Запит і відповідь будуть однаковими у будь-якого автомобіля, що відповідає стандарту.

Діагностика за стандартом OBDII є універсальною. Чіпу ELM327 абсолютно все одно, до якого автомобілю він підключений. Протоколи обміну даними – стандартні, адреси параметрів і датчиків – стандартні. Перелік підтримуваних параметрів повідомляє сам автомобіль.

Середовище платформи iOS починаючи с версії 8.0 вимагає від пристроїв, що підключаються до iPhone, підтримки версії Bluetooth 4.0. Це призводить до того, що не кожен ELM327-адаптер можна підключити до iPhone. Підтримка адаптером Bluetooth 4.0 є необхідною умовою для обміну даними з додатком, встановленим на iPhone.

2.4.6 Sockets

Веб-сокети – це технологія, яка дозволяє створювати інтерактивне з'єднання між клієнтом, в якості якого виступає веб-браузер або мобільний

додаток, та сервером для обміну повідомленнями в режимі реального часу. Сокети, на відміну від HTTP, дозволяють працювати із двонаправленим потоком даних.

WebSocket розроблений для втілення в веб-браузерах і веб-серверах, але він може бути використаний для будь-якого клієнтського або серверного додатка. Протокол WebSocket - це незалежний протокол, заснований на протоколі TCP. Він робить можливим більш тісний контакт між браузером і веб-сайтом, сприяючи поширенню інтерактивного вмісту та створення додатків реального часу.

Порівняльний аналіз технологій HTTP та Web-Sockets:

HTTP

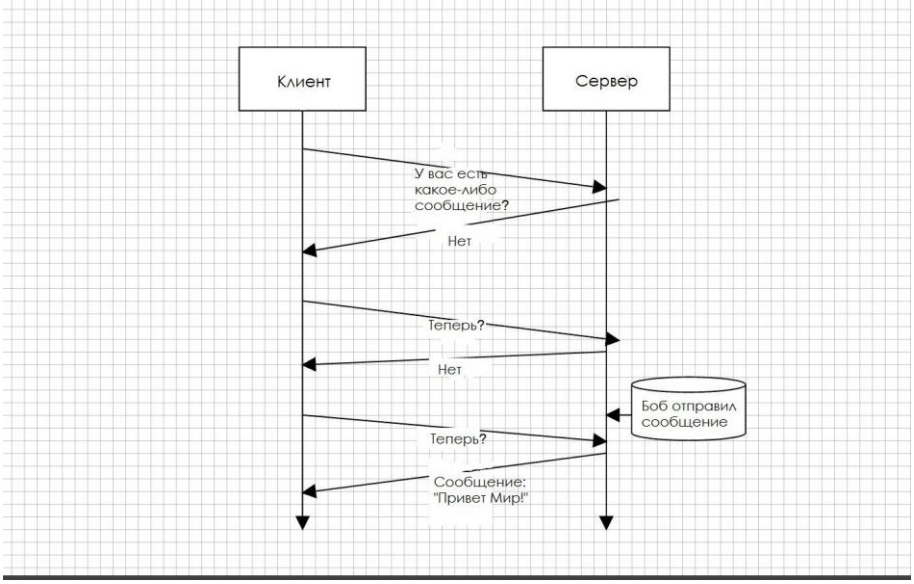


Рисунок 2.8 - Схема обміну повідомленнями по HTTP

Браузер постійно запитує у сервера, чи є для нього нові повідомлення, і отримує їх, якщо такі є.

Веб-сокети

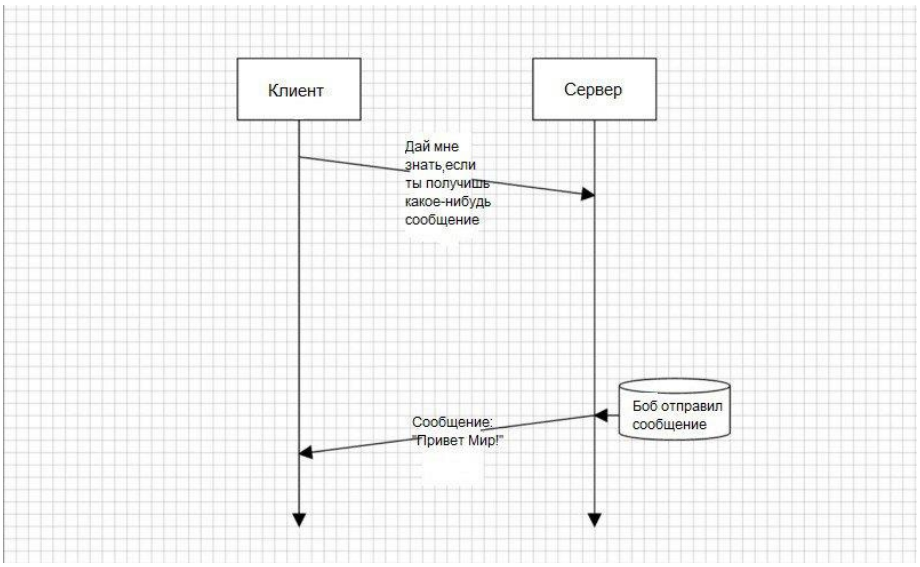


Рисунок 2.9 - Схема обміну повідомленнями при використанні веб-сокетів

Веб-сокети не вимагають циклічної відправки запитів. Досить виконати один запит і чекати відгуку. Ви можете просто слухати сервер, який буде відправляти вам повідомлення, коли вони з'являться на сервері.

Веб-сокети можна використовувати, якщо ви розробляєте:

- додатки реального часу;
- чат-додатки;
- багатокористувацькі ігри.

Але треба бути обережним в рішенні щодо використання веб-сокетів. Інколи розробники використовують їх там, де це зовсім не потрібно і достатньо застосовувати звичайний REST API з методами GET, POST, PUT та DELETE.

У даному дипломному проекті використовуються обидві технології спілкування з віддаленим сервером.

Веб-сокети стануть у нагоді при передачі діагностичних даних у реальному часі, спілкуванні клієнта з віддаленим спеціалістом за допомогою чату.

REST API буде використовуватись для отримання інформації про авторизацію, реєстрацію, передачу статистичних файлів, збереження налаштувань додатка для індивідуального користувача тощо.

2.4.7 Логування діагностичних даних

Іноді обставини не дозволяють передати діагностичні дані у реальному часі. Наприклад, водій не має можливості встановити з'єднання з мережею Інтернет. Тоді користувач має змогу зберігати результати тестування у пам'яті смартфону, а вже потім передавати їх на сервер.

Існує багато способів зберігання локальної інформації. Щоб визначити який саме буде оптимальним для задач дипломного проекту, був проведений аналіз найпопулярніших рішень.

CSV

CSV (Comma Separated Values) – це текстовий файл, в якому міститься структурована інформація.

Кожен рядок файлу, розділений символом нового рядку, - це окремий рядок таблиці, а стовпчики відокремлені один від іншого символами-роздільниками. В якості роздільника найчастіше використовують кому, але існує можливість ставити й інші роздільні символи, наприклад, пробіли, крапки з комою, табуляції тощо.

Переваги:

- простота використання;
- легко змінити структуру документу, якщо треба добавити чи вилучи якесь поле із логу;
- елементарне кодування у програмну модель та навпаки, можливість фільтрації даних;
- сумісність з іншими серверними та клієнтськими платформами.

Недоліки:

- зберігання інформації в файлах потребує постійного контролю за станом локальних файлів;
- іменування файлів дуже важливо. Збереження файлу з ім'ям, яке дублює ім'я вже створеного файлу призведе до втрати попередніх даних.

Core Data

Core Data – це рішення “із коробки”, використання якого пропагує компанія Apple - потужний і гнучкий фреймворк для зберігання і управління графом вашої моделі, який займає своє місце в арсеналі iOS-розробника.

Переваги:

- структура реалізована на використані баз даних SQLite, тому легко засвоюється розробниками, які працювали із системами керування базами даних;
- графічний інтерфейс відносин між сутностями дає уявлення про організовану структуру;
- можливість використовувати багато сторонніх бібліотек, які спрощують взаємодію із сутностями.

Недоліки:

- важко слідувати вибраній архітектурі, тому що доступ до даних виконується через асинхронний контекст. Розробник повинен слідкувати, в якому контексті він працює. Наприклад, при додаванні сутності в одному контексті, вона може ще не з’явитися в іншому. Це змушує розробника синхронізувати контексти, які можна прирівняти до потоків;
- швидкодія не викликає нарікань, але порівняно з іншими методами, вона в десятки разів менше;
- потребує складної структурної реалізації, що викликає необхідність в написанні нового коду та, відповідно, збільшує ймовірність появи помилок.

Realm

Realm - це система управління об'єктної базою даних з відкритим кодом, яка була створена спеціально для мобільних пристроїв.

Переваги:

- за рахунок оптимізації процесів, швидкість опрацювання запитів в десятки разів перевищує швидкість Core Data;

- абстрагування від слідкування за контекстом. Виклик Realm виконується в будь-якому місці програми, головне створювати новий об’єкт Realm для кожного потоку – далі система зробить все сама.

Недоліки:

- значно збільшує розмір додатка;
- потребує написання великих моделей даних.

Отже, можна зробити такі висновки.

1. Core Data є функціонально повним інструментом зберігання даних, але потребує багато часу для його підтримки та тестування. Для дипломного проекту використання цієї бібліотеки є надмірним функціоналом, який значно ускладнить проект.

2. Теж саме стосується і системи Realm, яка занадто велика для додатку, що розроблюється.

3. Ідеальним рішенням буде використання файлів у форматі CSV. Структурований імпорт дозволить з легкістю передавати інформацію, а зробити попередній аналіз, відкривши файл у будь-якому текстовому редакторі.

3. РЕАЛИЗАЦІЯ ДІАГНОСТИЧНОГО ДОДАТКУ

3.1 Графічний інтерфейс

Графічний інтерфейс діагностичного додатку розроблений при дотриманні стандартів Apple Human Interface Guidelines – це офіційні рекомендації щодо проектування користувацького інтерфейсу у мобільних додатках, що виконуються на iOS. Слідкування цим правилам зменшує інтуїтивний бар'єр між розробником та його міркуваннями щодо використання додатку та кінцевого користувача, який виконує фактичні дії. Досягти цього допомагає збіжність між керуванням звичною операційною системою та додатком.

Наприклад, перехід між екранами виконується за допомогою нижнього меню, яке виділяє вкладку активного екрану. Таке меню використовується в багатьох стандартних сервісах від Apple, тому користувач заздалегідь знає як керувати додатком, тому що ті ж самі елементи він використовує кожен день.

В якості інструменту побудови графічного інтерфейсу використовується стандартна бібліотека AutoLayout. Вона дозволяє керувати графічним інтерфейсом за допомогою спеціальних файлів, які мають розширення *.storyboard. Ці файли – звичайні XML-таблиці, але середа розробки XCode при відкритті таких файлів автоматично генерує інтерфейс на основі XML подання.

Такий підхід до розробки графічного інтерфейсу не завжди доцільний. Наприклад, коли екран містить лише стандартні графічні компоненти, швидше написати його звичайним програмним кодом. Але в діагностичному додатку, що розробляється використовуються кастомізовані графічні елементи, тому більшість екранів будуть мати відповідне представлення у storyboard.

Розглянемо головні екрани додатка:

1. Екран реєстрації або авторизації. Після реєстрації (авторизації) екран кабінету користувача.

При реєстрації новий користувач вказує свій статус у додатку. В даній роботі розглядається тільки функціонал клієнту – недосвідченого водія.

Екран приватного кабінету використовується для зберігання інформації про користувача, швидкого доступу до переліку автомобілів, які діагностувалися за допомогою додатку, збережених файлів зі статистикою та результатами діагностичних тестів.

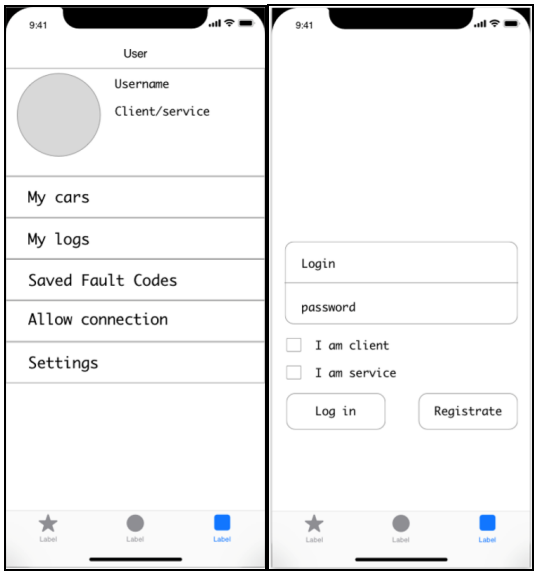


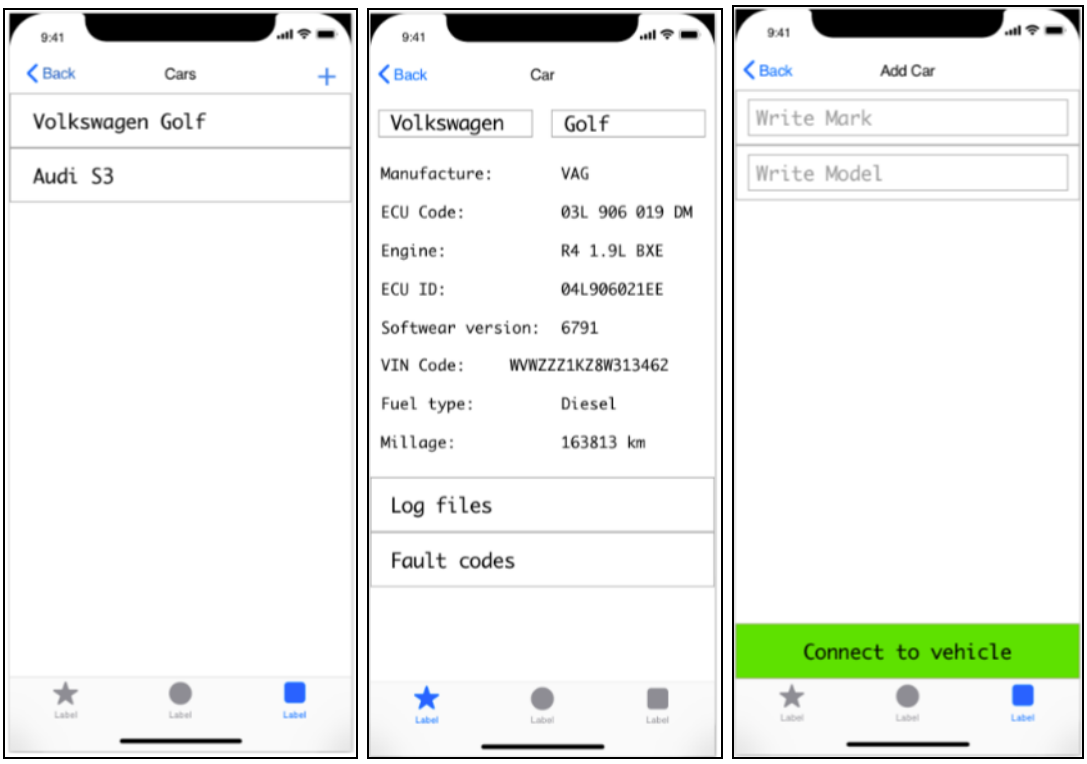
Рисунок 2.9 –Приватний кабінет Рисунок 2.10 – Вікно реєстрації

Опис елементів кабінету.

При натисканні на перший елемент таблиці відбувається перехід до детального огляду автомобіля. Користувач може бачити головні параметри свого авто та має змогу перейти до списку збережених діагностичних даних або помилок виключно для свого авто. Це забезпечує швидкий пошук інформації, коли головний перелік переповняється інформацією і користувачу важко виконувати навігацію. Списки логів на цьому екрані формуються виключно для вибраного автомобіля.

При натисканні на символ “+” у верхньому меню користувач переходить до нового екрану, де він може додати новий автомобіль. Слід зауважити, що неможливо додати новий автомобіль, не приєднавши його за допомогою OBDII-адаптеру. Такий тип підключення дозволяє фільтрувати тільки авто, що підтримують стандарт OBDII. Це робиться для того, щоб не викликати у користувача відчуття пустоти, коли він бачить автомобіль у переліку,

але не може застосовувати функціонал додатку до нього. Після вдалого встановлення з'єднання, відбувається перехід на попередній екран. Натиснувши на тільки що доданий автомобіль, користувач переходить до детального відображення, де вже доступні параметри нового авто.



(зліва направо)

Рисунки 2.11 – Перелік автомобілів

Рисунок 2.12 – Детальна інформація для обраного автомобіля

Рисунок 2.13 – Додавання нового автомобіля у додаток

На цьому екрані відображаються списки всіх збережених логів. Представлення логу включає назву файлу, дату та час його створення.

Натискаючи на вибраний лог, користувач переходить до веб-контролеру. Не слід плутати вікно мобільного браузера та веб-представлення. В даному випадку відкривається стандартний контролер, що може правильно відображати CSV файл.

Рішення про використання веб-контролеру було прийнято, щоб дати водію кращій досвід користування. Вибраний CSV файл буде однаково відображатися на будь-якому девайсі, що враховуючі його складну структуру, дає змогу краще розуміти представлені дані.

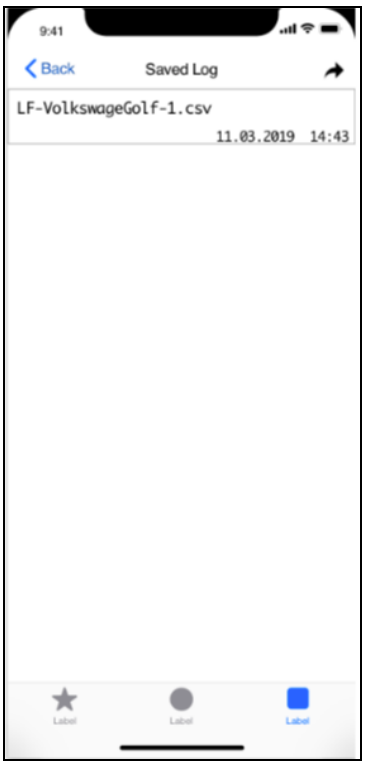


Рисунок 2.14 – Перелік збережених csv файлів

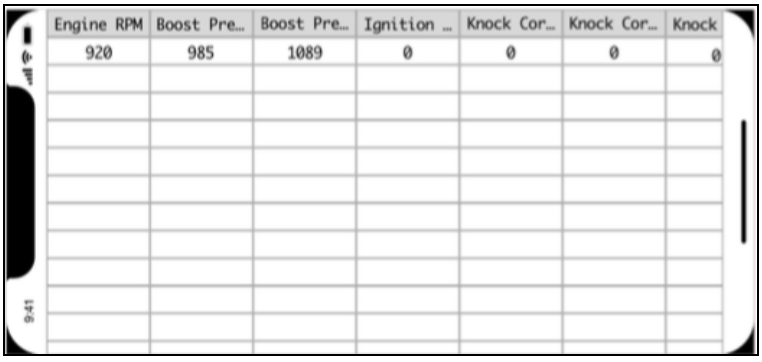


Рисунок 2.15 – Відображення csv файлу

Перелік збережених кодів помилок (DTC – Data Trouble Code)

На цьому екрані відображаються списки всіх збережених кодів помилок. Представлення логу включає назву файлу, дату та час його створення.

Аналогічно до діагностичних звітів, при натисканні на файл відбувається перехід до веб-контролера.

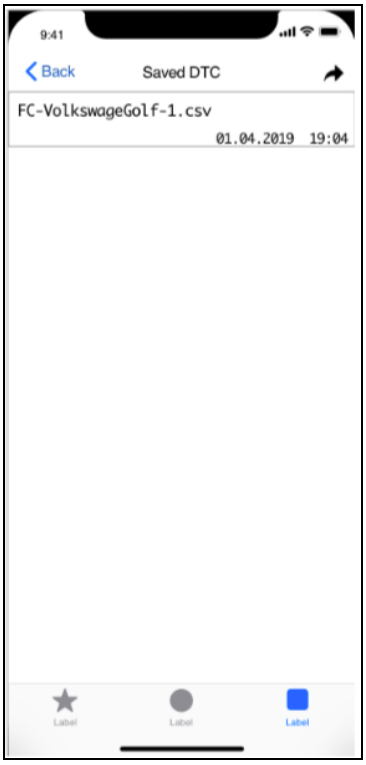


Рисунок 2.20 – Перелік збережених звітів про помилки

2. Екран показників у реальному часі

Рисунок 2.21 – Екран показників у реальному часі

Цей екран концентрує увагу користувача на принципі взаємодії діагностичного адаптера та смартфона. Динамічні показники оновлюються при підключені до автомобіля через Bluetooth.

Для побудови спідометра були використані стандартні графічні об’єкти. Побудова кіл здійснюється за допомогою малювання арок певного радіуса від початкового умовного кута x1 до кінцевого x2. Шлях, по якому будуються лінії є нероздільним. Тобто, не можливо задати одній частині білий колір, а іншій – червоний. Рішенням цієї проблеми є маскування певних ліній, що дозволяє накладати інші кольори на фігуру.

Розглянемо інші графічні елементи.

У верхньому лівому куті розташований показник температури двигуна – один з найважливіших факторів стабільної роботи усього автомобіля.

У верхньому правому куті знаходиться індикатор найвідомішої помилки «Перевірити двигун» («Check engine»). Якщо помилки не має, індикатор не горить, тобто колір індикатора задається із непрозорістю 60-80%. При виникненні помилки колір стає насиченим, інформуючи користувача про неполадки в роботі двигуна.

Трохи нижче розташовані шкали температур масла та охолоджуючої рідини. При отримані даних, маленька біла стрілка-індикатор почне рухатися до актуального значення. Припускається, що показники не перебільшують 120 градусів Цельсія та не менше 60 градусів.

Спідометр представляє собою опосередковану версію спідометра реального автомобіля. Крім того, в залежності від типу двигуна – бензиновий чи дизельний – максимальна кількість оборотів буде обмежена до восьми та семи тисяч обертів на хвилину відповідно. Ділення та відмітки на колі допомагають орієнтуватись у актуальній кількості обертів на хвилину. В центрі знаходиться індикатор поточної швидкості у кілометрах за годину.

Четверта чверть кола відокремлена від трьох інших. Вона має відношення тільки до турбованих автомобілів, доля яких на ринці значна збільшилась. Ця шкала відслідковує тиск у системі турбіни. Цей фактор значно впливає на динаміку розгону автомобіля.

Знизу розташований паливний блок. В залежності від максимального об’єму паливного бака та поточної кількості палива, колір блоку буде міня-

тися від зеленого, коли автомобіль не витратив навіть третини палева, до червоного, коли залишилось менше третини. Поточна кількість палива відображається справа від блоку. Об'єм паливного баку на кількості палива вимірюється у літрах.

3. Екран параметрів.

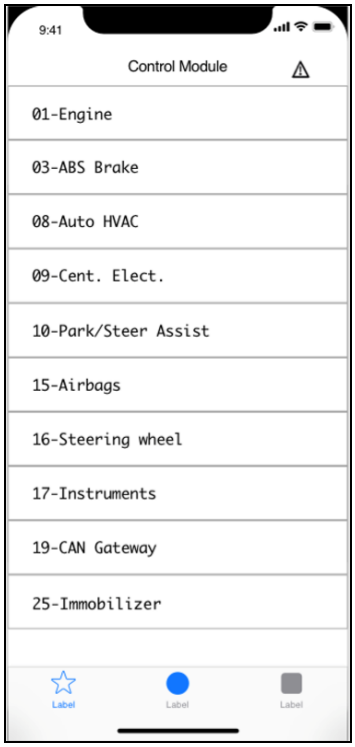


Рисунок 2.22 – Перелік категорій параметрів

Цей екран структурує діагностичні параметри та поділяє на категорії для зручної навігації.

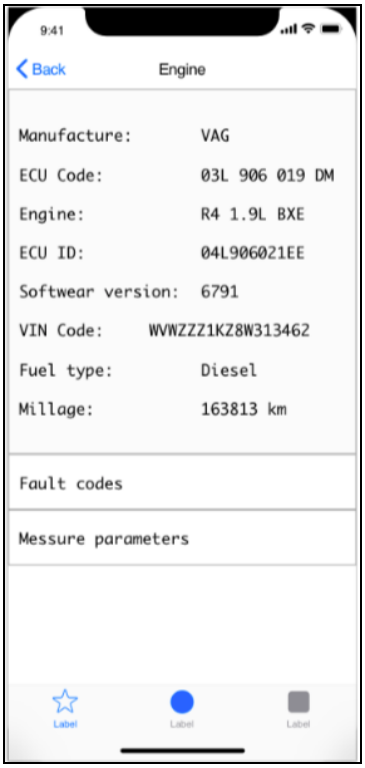


Рисунок 2.23 – Базові параметри для обраної категорії

При виборі певної категорії параметрів відкривається нове вікно з детальним представленням обов'язкових для зчитування параметрів цієї категорії.

Кнопка «Fault codes» робить перехід до переліку діагностичних помилок виключно цієї категорії. Якщо помилок не має, з'являється відповідне повідомлення.

Кнопка "Measure parameters" відкриває екран вибору параметрів для зчитування з OBDII-адаптеру.

Рисунок 2.24 – Вибір параметрів для відстеження

На цьому екрані ми маємо перелік параметрів, доступ до яких надає діагностичний адаптер. Зліва відображається назва параметра, а справа булева комірка для включення або виключення параметру зі списку для зчитування.

На верхньому блоці знаходиться кнопка «Log». Вона виконує перехід до схожого екрану, але замість булевих комірок, показуються актуальні – записані після останнього зчитування, параметри. Це вікно дозволяє мати доступ до загальної статистики всіх відомих параметрів, якщо вони колись зчитувались.

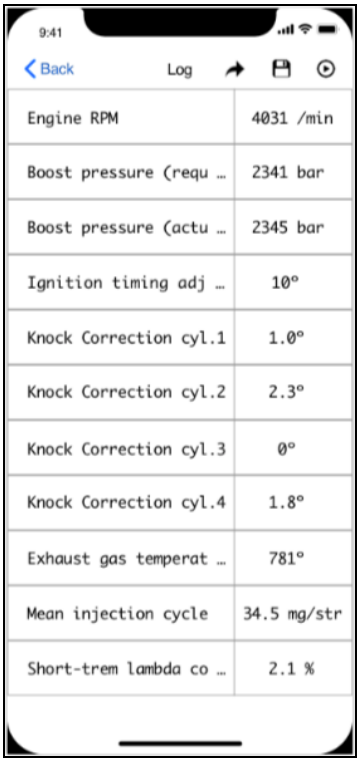


Рисунок 2.25 – Результат діагностики

Зверху від таблиці параметрів знаходяться кнопки керування логом.

Ліва кнопка зі стрілкою видаляє попередні логи, отже після її натискання усі поля даних будуть пусті доки відповідні параметри не будуть зчитані заново.

В центрі знаходиться кнопка збереження лога. Після її натискання усі параметри відповідної категорії будуть структуровані для csv-подання та збережені у CSV-файл, який можна передати як звичайний файл використовуючи імпорт із додатка. Apple надає безліч стандартних функцій для імпорту. Додаток може передати файл через електрону пошту, сервіс Air Drop, занести його до сховища iCloud. Окрім стандартних методів, де місце призначення – це завжди сервіс від Apple, ми можемо імпортувати файли до будь-якого власноруч встановленого додатка, що має змогу отримувати файли з обраним розширенням.

Права кнопка відправляє діагностичні дані вибраної категорії на сервер, якщо раніше було створено з’єднання.

3.2 Модуль «OBDII»

Модуль взаємодії із діагностичним адаптером – найважливіша частина додатку, тому саме вона розробляється першою.

Модуль, розроблений за архітектурою MVP має чотири основні складові: OBDII View Controller (View), OBDII Presenter (Presenter), OBD Configurator, OBDII Router, OBDII Service, Command (Model).

OBDII View Controller – головна частина модулю, що ініціалізує усі інші, оскільки контролер завантажується першим після переходу на новий екран. MVP-структура модуля зображена на малюнку 1:

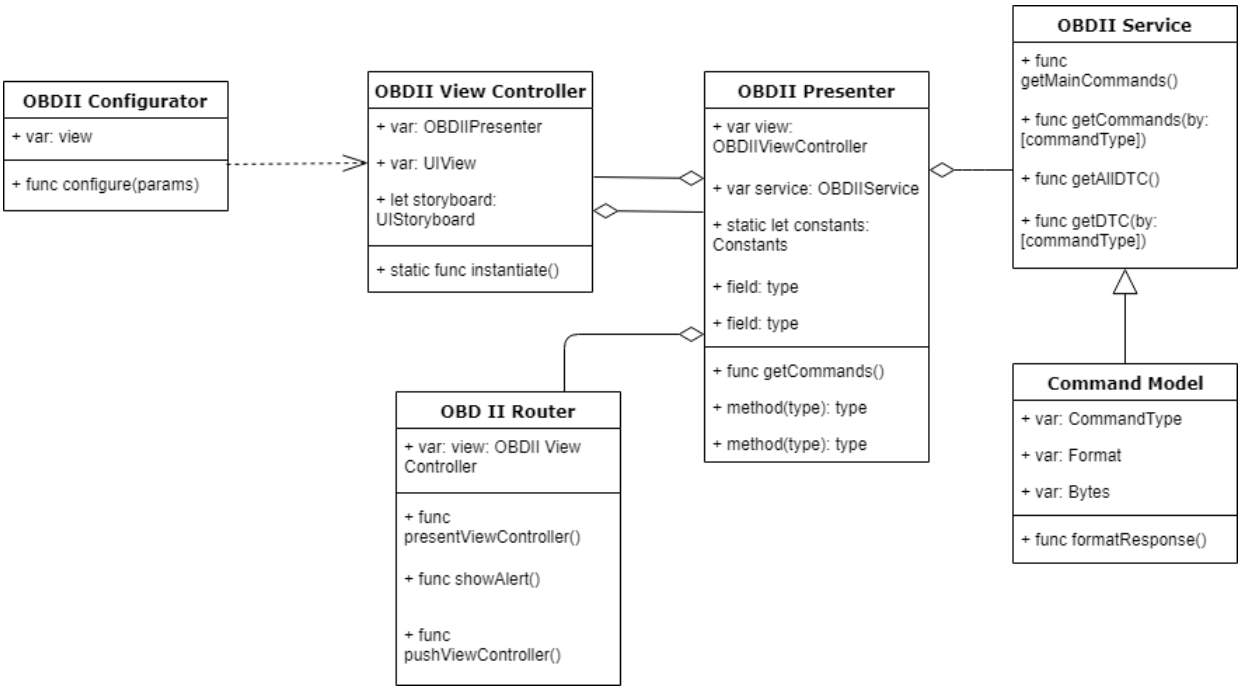


Рисунок 3.1 – Структура OBDII-модуля

Порядок завантаження модуля OBDII:

1. Звернення до відповідного storyboard-файлу повертає згенерований OBDII View Controller.

2. Ініціалізується OBDII Configurator до конструктора якого передається попередньо створений OBDII View Controller.

3. Викликається функція конфігуратора configure(:), яка приймає опціональні параметри. На момент розробки модуля нема необхідності у параметризації, але, як правило, така потреба з’являється при подальшому ускладнені логіки додатку. Тому доцільно зробити це зараз, виходячи з того, що

передача параметрів – звичайна річ, а їх відсутність – виключення для даного модуля.

4. Модуль готовий для використання.

Приклад роботи модуля:

Для отримання даних з OBDII-адаптера необхідно виконати запит до презентера, метода getCommands. Презентер, в свою чергу, звертається до OBDII Service, який виконую запит до адаптера за допомогою функції getMainCommands(). Слід зауважити, що в даному випадку не можливо повертати діагностичні дані за допомогою оператора return, тому що операції зі зчитування даних та передача через Bluetooth виконуються асинхронно. Найпростіше рішення цієї проблеми – виконання функціональних замикань, які передаються в якості параметрів та викликаються після підтвердження отримання результату запиту. Цей механізм дає змогу повертати не тільки результати запитів, але й повідомлення про помилки, не руйнуючи архітектури додатка.

Отже, сигнатура функції getMainCommands() перетворюється на getMainCommands(completion: (Result)->Void). Виклик completion блоку виконується після отримання даних з адаптеру.

Після отримання результату необхідно оновити графічний інтерфейс та в залежності від активності з’єднання із сервером передати дані до пристрою спеціаліста.

В блоці completion викликається функція контролеру view.updateInterface(commands), де параметри – це нові значення, які необхідно відобразити. Але не слід передавати до контролера специфічні дані такі як набори байтів, достатньо передавати рядки форматованих значень.

Створений модуль є функціонально повним, але не відповідає принципам SOLID.

OBDII Presenter має повний доступ до OBDII View Controller, але йому не потрібно керувати багатьма параметрами контролера, а лише тими, що потребують змін в залежності від стану моделі презентера. Необхідно позбу-

тися відкритості полів та методів контролера, окрім метода updateInterface(). Для цього застосуємо протоколи, які потім будуть реалізовані контролером та презентером. Це дозволить звертатися із презентера лише до тих методів та полів, які об'явленні у протоколі, та навпаки – контролер не буде знати про незалежні від нього частими презентера. На малюнку 2 зображена структура модуля із урахуванням взаємодії за допомогою протоколів:

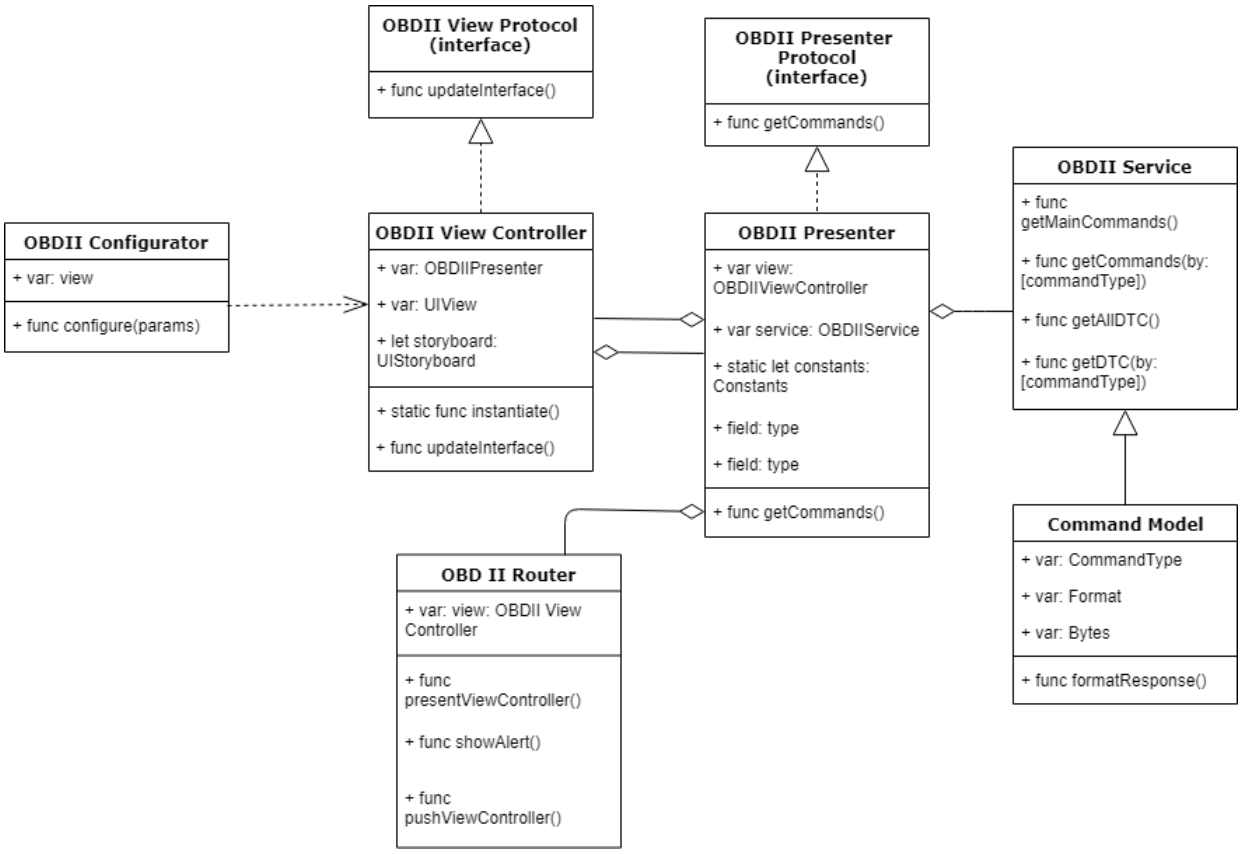


Рисунок 3.2 – Структура OBDII-модуля з використанням протоколів

Така структура дозволяє розмежувати внутрішні поля та методи і ті, що можуть викликатись з інших модулів.

Окремою перевагою використання протоколів є легкість тестування. Наприклад, якщо необхідно покрити тестами презентер цього модуля, ми можемо замінити в конфігураторі OBDII Service на його Mock-реалізацію, наприклад, MockOBDII Service.

Mock-класи – це клони звичайних класів, від яких вони наслідуються, але замість виконання зовнішніх запитів, вони замінюють їх статичними даними. Це дозволяє емулювати дані з серверу або адаптеру та тестувати робо-

ту презентера по обробці даних. Для генерації Mock-класів використовується наслідування від протоколу, аналогічно до контролеру і презентеру.

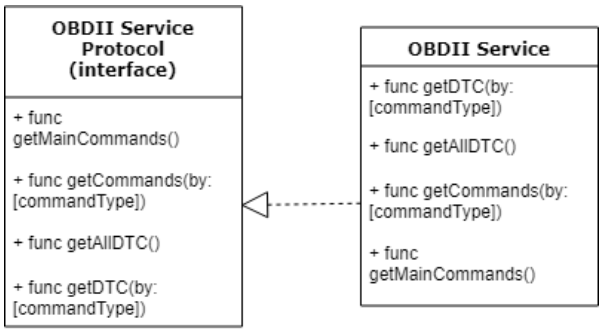


Рисунок 3.3 – Структура OBDII сервісу

Для переходів між екранами використовується OBDII Router, який має доступ до всіх функцій контролера, а отже, може перенаправляти екрани для різного відображення. Наприклад, він може презентувати новий екран як модальне вікно, або повідомлення, або добавити його до існуючого навігаційного стеку.

3.3 Модуль «Remote»

Після отримання даних необхідно мати змогу передати їх на віддалений сервер. Для цього використовується глобальний модуль, який можна використовувати з будь-якого екрану.

Оскільки для передачі даних через сокети необхідне попереднє підключення, то воно автоматичне відбуваються після завантаження додатку (якщо користувач вже був зареєстрований), або після процедури реєстрації. Не завжди є можливість встановити з’єднання. Причин для цього багато, наприклад, мобільний пристрій може буде не підключений до Інтернету, або сервер може бути відключений якийсь проміжок часу.

Так як більша частина функціоналу додатку не потребує встановлення віддаленого з’єднання, то не слід навантажувати користувача постійними перепідключеннями в разі попередніх невдалих спроб. В разі першого невдалого підключення з’являється повідомлення про помилку, після чого користувач може виконати повторне підключення до серверу власноруч.

Принцип роботи модуля по встановленню з’єднання показаний намалюнку 4:

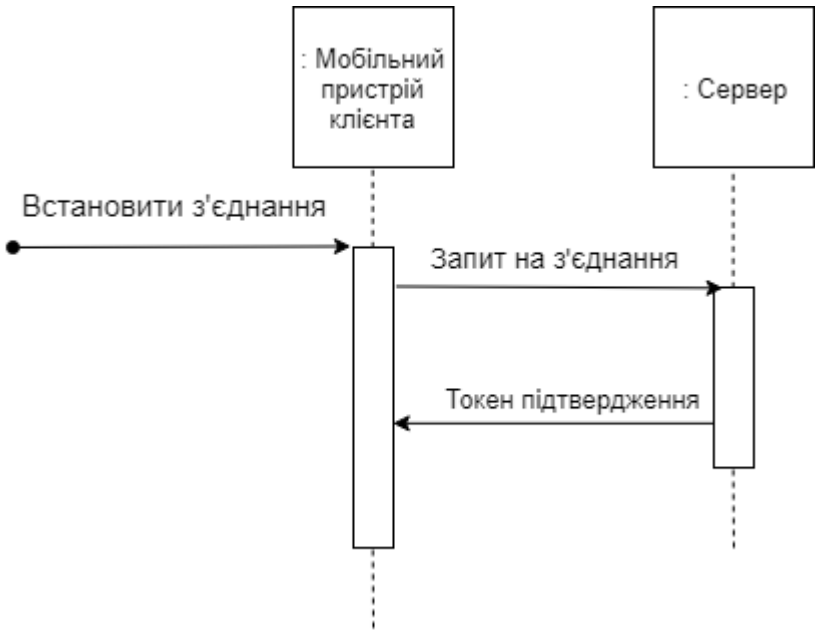


Рисунок 3.4 – Встановлення з’єднання з сервером

Передача у режимі реального часу виконується окремо від передачі файлів діагностичних звітів.

При отримані чергового пакету, що надходить з діагностичного адаптера, викликається функція передачі цього пакету на сервер. Передача відбувається за допомогою бібліотеки Socket.IO, яка візуально спрощує взаємодію із сокетами.

Socket.IO головним чином використовує протокол WebSocket, але підтримує й інші методи, наприклад Adobe Flash сокети, JSONP запити. Крім того, що Socket.IO може бути використана як оболонка для WebSocket, вона містить багато інших функцій, включаючи трансляцію на кілька сокетів, зберігання даних, пов'язаних з кожним клієнтом, і асинхронний ввід / вивід даних.

Повідомлення надсилаються за ключем «realtime». Сервер слухає сокет по цьому ключу і, коли надходять дані, визначає наступних адресатів, яким треба передати отримані дані.

Також необхідно слухати сокет з ключем «response», який надсилає повідомлення в разі, якщо діагностичні дані передаються у невірному форматі, або не передаються взагалі.

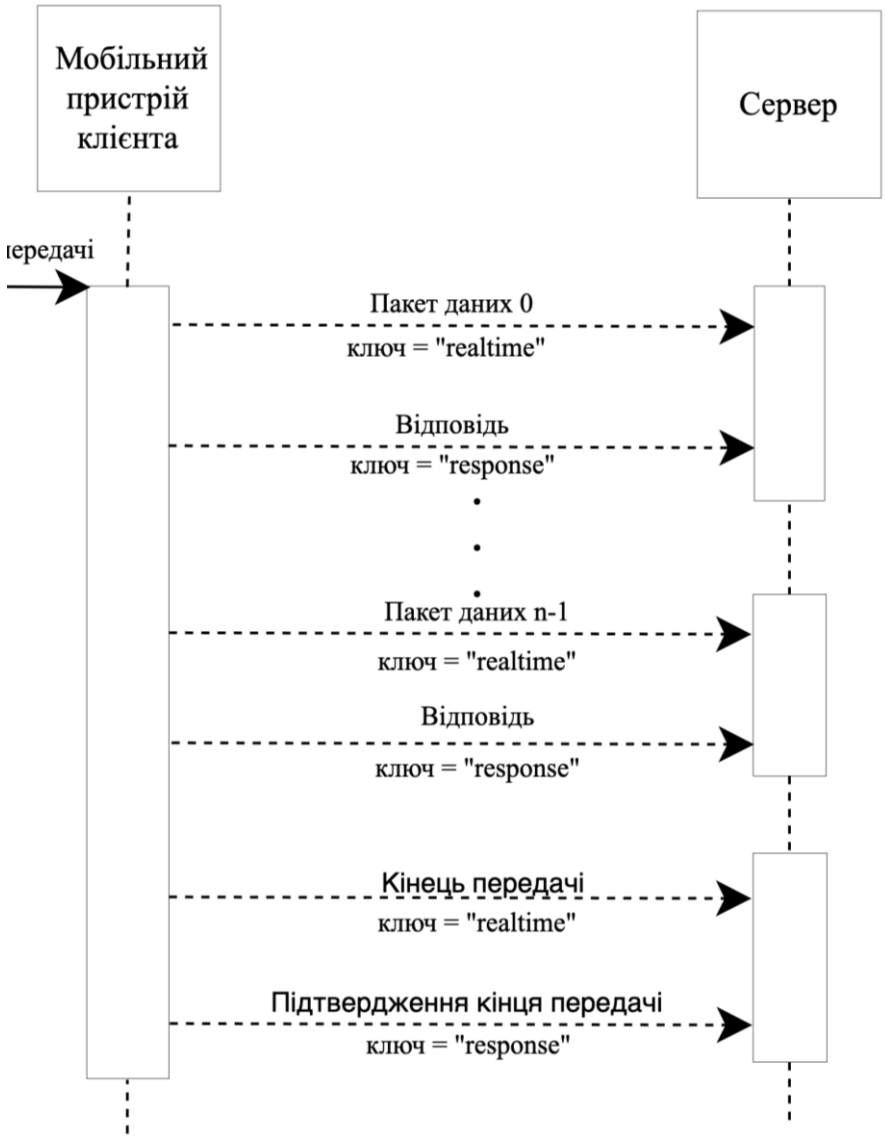


Рисунок 3.5 – Передача даних від клієнта до сервера

3.4 Модуль «CSV»

Модуль CSV використовується для відображення отриманих від серверу логів у CSV-файли, які зберігаються локально на мобільному пристрої і можуть бути передані на віддалений сервер.

Цей модуль керує локальним сховищем:

- При додаванні діагностичних даних нового автомобіля створюється нова директорія, в якої ім’я – це VIN код автомобіля, який є унікальним ідентифікатором транспортного засобі та обов’язковим параметром для кожного зчитування.

- При додаванні діагностичних даних створюється CSV файл, в якому символом відокремлення є «,». Цей файл записується до локальної папки, яка іменується VIN кодом того авто, з якого прийшли дані. В разі відсутності такої директорії, вона створюється разом із файлом.
- При відправлені CSV файлу на сервер, користувач вибирає, чи потрібно залишити локальним файл або видалити його.
- Після видалення CSV файлу, відновити його вже неможливо.
- Якщо файл, що був видалений – останній діагностичний лог для вибраного транспортного засобу, його папка теж видаляється.

На рис. 3.6 наведена структура модуля CSV. Така організація роботи з локальним сховищем дозволяє легко структурувати збережені файл, але не накопичувати дані, необхідності в яких вже нема.

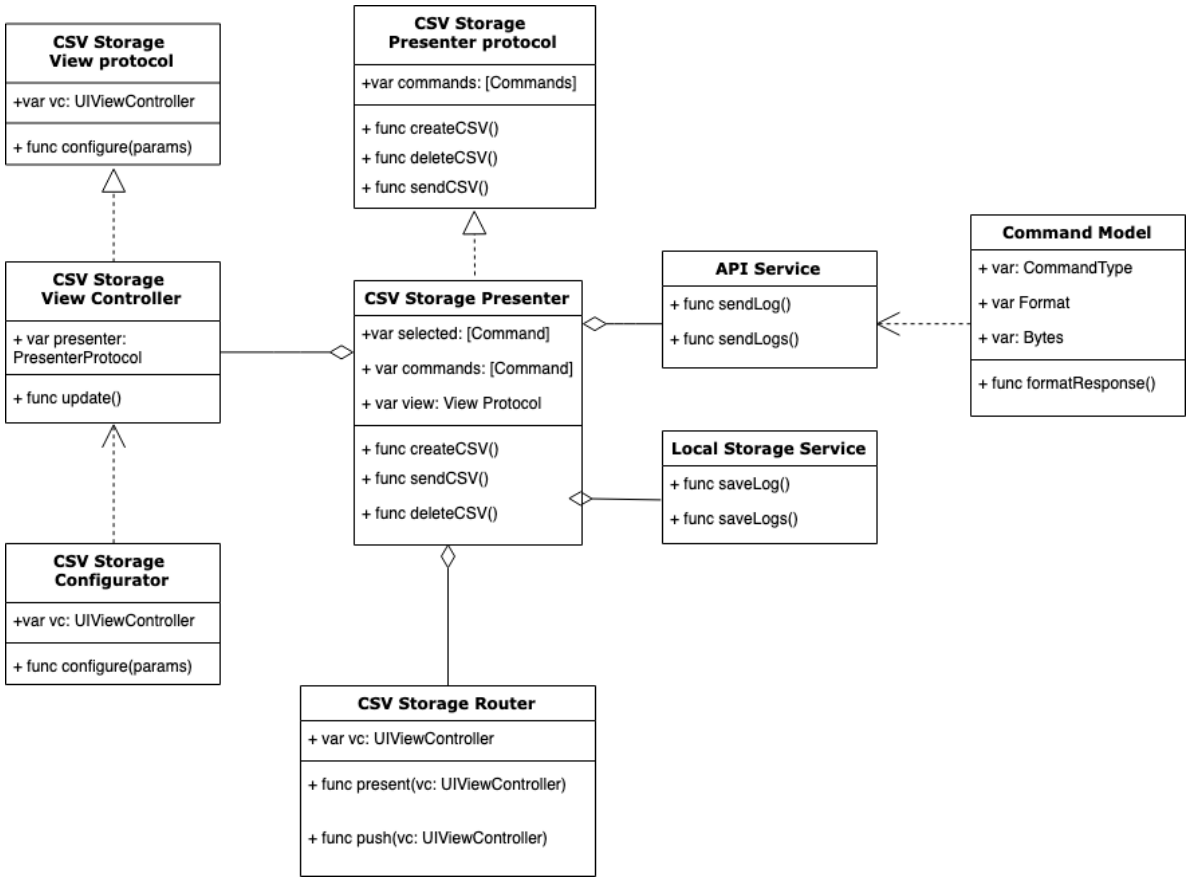


Рисунок 3.6 – Структура модуля CSV

3.5 Модуль “Спідометр”

Цей модуль створений, щоб привернути увагу користувача.

Контрастний графічний інтерфейс наглядно демонструє роботу додатка. Важливим фактор для використання цього модулю є те, що він першим відображається на екрані, створюючи перше враження користувача від додатку.

Кастомізоване представлення SpeedometerView має декілька ключових параметрів, які відображають оновлення показників:

- поточна швидкість;
- поточна температура двигуна;
- поточна температура масла;
- поточний тиск турбіни, якщо двигун оснащується їй. Не усі автомобілі мають встановлену турбіну, але, з іншого боку, майже всі сучасні автомобілі оснащуються турбінами і демонстрація цього показника може впливати на перше враження користувача.

ВИСНОВКИ

В процесі виконання дипломного проекту досліджено існуючі програмні рішення для діагностики автомобілів. Серед обраних програм для аналізу діагностичних даних проаналізовані ті, що дозволяють зчитувати інформація за допомогою мобільного пристрою під управлінням операційної системи iOS.

Аналіз показав, що деякі додатки не дозволяють правильно отримувати інформацію із бортових систем автомобіля, деякі з них застарілі та не мають необхідного функціонала, але більшість популярних додатків орієнтовна на користувачів, які здатні аналізувати отримані дані.

Ці умови довели доцільність створення нового діагностичного додатка та дозволили спроектувати модульну структуру для майбутньої розробки, яка орієнтована на звичайного власника авто, який має намір зекономити час на діагностиці. Але, в той же час, отримані дані зберігаються у стандартному неформатованому вигляді для швидкого аналізу з боку спеціаліста.

Розроблено структуру кожного окремого модуля виходячи із його складності, кількості компонентів та ймовірності розширення. Проведено аналіз архітектурних рішень щодо проектування модулів для мобільних додатків для платформи iOS. В результаті була обрана архітектура MVP, яка дозволяє розділяти модульні компоненти, відділяючи логіку від графічного інтерфейсу, але залишаючи структуру легкою для сприйняття та модифікування.

Розроблено графічний інтерфейс, який забезпечує користувачу інтуїтивний доступ до функціоналу та не повторює дизайн вже існуючих програм на ринку.

Також використано особливості мобільної платформи iOS, яка дозволяє використовувати технологію Bluetooth Low Energy для комунікації із

додатком. Ця технологія значно спрощує програмну структуру периферійного пристрою, який дозволяє створити структуровану взаємодію між додатком та OBDII-адаптером.

Розробка діагностичного додатка дозволила покращити розуміння потреб власників авто за рахунок впровадження мобільних технологій у життя людини.

Створений додаток дозволяє автомобілісту легко контролювати стан авто та діагностичні показники в незалежності від технічних знань щодо роботи автомобіля.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Protocol-Oriented Programming [Електронний ресурс] // Habr. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/358804/.
2. Advanced iOS App Architecture (First Edition): Real-world app architecture in Swift / Rene Cacheaux & Josh Berlin, 2018. – Raywenderlich.
3. Data Structures and Algorithms in Swift, 2017. – Raywenderlich.
4. Design Patterns by Tutorials, 2018. – Raywenderlich.
5. Concurrency by Tutorials, 2017. – Raywenderlich.
6. iOS 10 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / Matt Neuburg, 2017.
7. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch, 2018.
8. Основы разработки приложений под iOS и macOS / Василий Усов, 2018.
9. Разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK / Молли Маскри, 2017.
10. Swift Programming Language / Apple, 2019
11. Human Interface Guidelines / Apple, 2019.
12. 100 Things Every Designer Needs to Know About People / Susan Weinschenk, 2011.