Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації
(повна назва)
Кафедра Радіотехнологій інформаційно-комунікаційних систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти **перший (бакалаврський)**

РОЗРОБКА ДОДАТКУ ДЛЯ IOS, НАДАННЯ ДАНИХ ПРО ЯКІСТЬ ПОВІТРЯ
(тема)

Виконав:
студент **IV** курсу, групи **ІТІР-20-1**

Косенко М. А.
(прізвище, ініціали)

Спеціальність **126 Інформаційні системи та технології**
(код і повна назва спеціальності)

Тип програми **освітньо-професійна**

Освітня програма **Інформаційні технології інтернету речей**
(повна назва освітньої програми)

Керівник **Ст. викл. Ганшин Д. Г.**
(посада, прізвище, ініціали)

Допускається до захисту
В.о.зав. кафедри РТІКС _____ **Зарудний О.А.**
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації

Кафедра Радіотехнологій інформаційно-комунікаційних систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 Інформаційні системи та технології
(код і повна назва)

Тип програми Освітньо-професійна

Освітня програма Інформаційні технології інтернету речей
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові **КОСЕНКУ МИКОЛІ АЛЬБЕРТОВИЧУ**
(прізвище, ім'я, по батькові)

1.Тема роботи **РОЗРОБКА ДОДАТКУ ДЛЯ IOS, НАДАННЯ ДАНИХ ПРО ЯКІСТЬ ПОВІТРЯ**

затверджена наказом по університету від **27 травня 2024 р**. **№ 500 Ст**

2. Термін подання студентом роботи до екзаменаційної комісії **10 червня 2024 р.**

3. Вихідні дані до роботи _____

3.1 Провести огляд та аналіз аналогічних додатків

3.2 Розробити UML схему додатку

3.3 Обрати чіткий алгоритм для реалізації додатку

3.4 Розробити програмне забезпечення роботи додатку

4. Перелік питань, що потрібно опрацювати в роботі _____

Вступ. 1 Аналіз існуючих рішень та теоретичні основи. 2 Проєктування додатка. 3 Програмно – апаратна частина. 4 Розробка ios додатка. Висновки. Перелік джерел посилання. Додатки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) (п.5 включається до завдання за рішенням випускової кафедри)_____
Слайди комп'ютерної презентації, датчик вимірювання якості повітря BME680, плата ESP32, Google API, UML діаграма класів, ios додаток, рисунки схеми, скріншоти виконаного завдання._____

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	Ст. викл. Ганшин Дмитро Геннадійович		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Вступ	06.05 - 07.05.2024	виконано
2	Огляд та аналіз аналогічних систем	08.05 - 10.05.2024	виконано
3	Розробка структурної схеми пристрою	11.05 - 14.05.2024	виконано
4	Вибір та обґрунтування елементної бази	15.05 - 20.05.2024	виконано
5	Розробка програмного забезпечення	21.05 - 28.05.2024	виконано
6	Висновки	01.06.2024	виконано
7	Оформлення пояснювальної записки	05.06.2024	виконано
8	Оформлення ілюстрацій	07.06.2024	виконано
9	Представлення роботи на кафедру	10.06.2024	виконано

Дата видачі завдання **06 травня 2024 р.**

Студент _____ М.А. Косенко
(підпис)

Керівник роботи _____ ст. викл. Д.Г. Ганшин
(підпис)

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з пояснювальної записки, що містить 92 сторінки тексту, 80 рисунків, 9 джерел посилання і 4 додатка.

ПОВІТРЯ. ЯКІСТЬ. КООРДИНАТИ.
РОЗРОБКА. ДАТЧИК. ПРОГРАМА. АРДУІНО.

Об'єктом розробки є ios додаток якості повітря.

Метод дослідження – описово-аналітичний

Метою даної кваліфікаційної роботи бакалавра є розробка додатка для ios, надання даних про якість повітря.

В даній роботі проведено огляд і аналіз вимог до моніторингу якості повітря, огляд і аналіз існуючих систем моніторингу, розроблено UML схему системи моніторингу якості повітря та запропоновано програмне забезпечення для управління системою.

ABSTRACT

The qualification work of the bachelor consists of an explanatory note containing 92 pages of text, 80 figures, 9 literary sources and 4 appendices.

AIR. QUALITY. COORDINATE.
SOFTWARE. SENSOR. PROGRAM. ARDUINO.

The object of development is the air quality ios application.

Research method - descriptive and analytical

The purpose of this bachelor's qualification work is to develop an application for ios, providing data on air quality.

In this paper, a review and analysis of requirements for air quality monitoring, a review and analysis of existing monitoring systems, a UML diagram of an air quality monitoring system was developed, and software for system management was proposed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування;

AQI (Air Quality Index) – рівень забруднення повітря;

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі гіпертексту;

IDE (Integrated Development Environment) – інтегроване середовище розробки;

IOS (iPhone operating system) – операційна система айфону;

IOT (Internet of Things) – інтернет речей;

JSON (JavaScript Object Notation) – текстовий формат обміну даними на основі JavaScript;

LLDB (Low Level Debugger) – відкладчик низького рівня програмування;

MacOS (Macintosh operating systems) – операційна система макбуку;

MVVM (Model ViewView Model) – шаблон проектування архітектури програми;

RGB (Red Green Blue) – базовий цвітоколір;

TvOS (Apple TV Software) – операційна система телевізорів;

UI (User Interface) – інтерфейс користувача;

UML (Unified Modeling Language) – уніфікована мова моделювання;

UX (User experience) – досвід користувача;

WatchOS (Apple Watch operating system) – операційна система годинників;

ВСТУП

У сучасному світі, де проблема забруднення повітря стає все більш актуальною, розробка інноваційних рішень для моніторингу та аналізу якості повітря має першорядне значення. Забруднене повітря негативно позначається на здоров'ї людини і викликає різні захворювання дихальної та серцево-судинної системи. Крім того, низька якість повітря впливає на загальний добробут населення, продуктивність праці і навіть тривалість життя. Зростаюча індустріалізація та урбанізація, а також збільшення кількості автомобільних та промислових підприємств значно погіршують стан атмосфери та вимагають негайних дій як з боку уряду, так і громадськості.

З огляду на ці проблеми, створення мобільних додатків, які надають дані про якість повітря в режимі реального часу, стало важливим інструментом у боротьбі за чисте довкілля, такі додатки дозволяють користувачам отримувати актуальну інформацію про рівень забруднення повітря в своєму регіоні, швидко реагувати на небезпечні зміни і захищати здоров'я. Наприклад, користувачі можуть обмежити свій час на вулиці або використовувати захисне спорядження у разі високого рівня забруднення.

Крім того, додаток допомагає підвищити обізнаність громадськості про екологічні проблеми, заохочуючи позитивні дії населення щодо поліпшення екологічної ситуації. Знання стану погоди сприяє формуванню екологічної обізнаності та відповідального ставлення до навколишнього середовища. Легкий доступ до відкритих даних та інформації може сприяти громадським ініціативам щодо скорочення забруднення, таким як кампанії з озеленення міст, проведення екологічних фестивалів та підтримання чистоти транспорту.

Проблеми для розробників включають забезпечення точності та надійності даних, оптимізацію роботи додатків для мобільних пристроїв та вирішення проблем безпеки та конфіденційності даних користувачів. Очікування користувачів пов'язані не тільки з точністю даних, але і з простотою викорис-

тання програми, швидкістю і можливістю отримання персоналізованих рекомендацій.

Я сподіваюсь, що цей проект принесе значний внесок у створення більш чистого і безпечного середовища для всіх. Впровадження таких мобільних додатків сприяє поліпшенню здоров'я і якості життя людей у всьому світі і є важливим кроком у вирішенні глобальної проблеми забруднення повітря.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНІ ОСНОВИ

1.1 Огляд літератури на тему якості повітря та його моніторингу

Якість повітря є важливим показником екологічного стану навколишнього середовища, що має прямий вплив на здоров'я населення та екосистеми. Забруднювачі повітря, такі як тверді частинки (PM2.5 і PM10), озон (O_3), двоокис азоту (NO_2), двоокис сірки (SO_2) та оксид вуглецю (CO), є основними компонентами, що негативно впливають на якість повітря. За даними Всесвітньої організації охорони здоров'я (ВООЗ), понад 90% населення світу дихає повітрям, яке перевищує допустимі рівні забруднення.

Моніторинг якості повітря здійснюється кількома способами, включаючи використання стаціонарних станцій моніторингу, мобільних лабораторій, супутникових спостережень і сенсорних мереж. Сучасні технології дозволяють інтегрувати дані з різних джерел для забезпечення поточного та оперативного моніторингу погодних умов.

Дослідження, проведене Лі Кан Хін (2021), показує, що комбінування даних зі стаціонарних станцій та супутникових знімків дозволяє значно покращити точність прогнозування рівня забруднення повітря. У роботі використовується метод машинного навчання для аналізу великих масивів даних.

В іншому дослідженні, Мартінез та Спаркс (2019) розглядають використання низьковартісних сенсорних мереж для моніторингу якості повітря у міських районах. Вони зазначають, що такі мережі можуть бути ефективними для виявлення локальних джерел забруднення.

1.2 Існуючі рішення для моніторингу якості повітря

Мобільні додатки для моніторингу якості повітря стають все більш популярними, оскільки вони надають користувачам можливість отримувати оперативну інформацію про стан повітря у реальному часі.

AirVisual – важливий інструмент моніторингу якості повітря, який допомагає користувачам розпізнавати стан навколишнього середовища та вживати необхідних заходів для захисту свого здоров'я. Додаток надає точні і актуальні дані про якість повітря, які допомагають підвищити обізнаність громадськості про екологічні проблеми.

Додаток AirVisual є одним з найпопулярніших додатків для моніторингу якості повітря. Він надає дані з тисяч станцій по всьому світу та використовує штучний інтелект для прогнозування рівнів забруднення у різних країнах та охоплює понад 10 000 міст (рисунок 1.1).

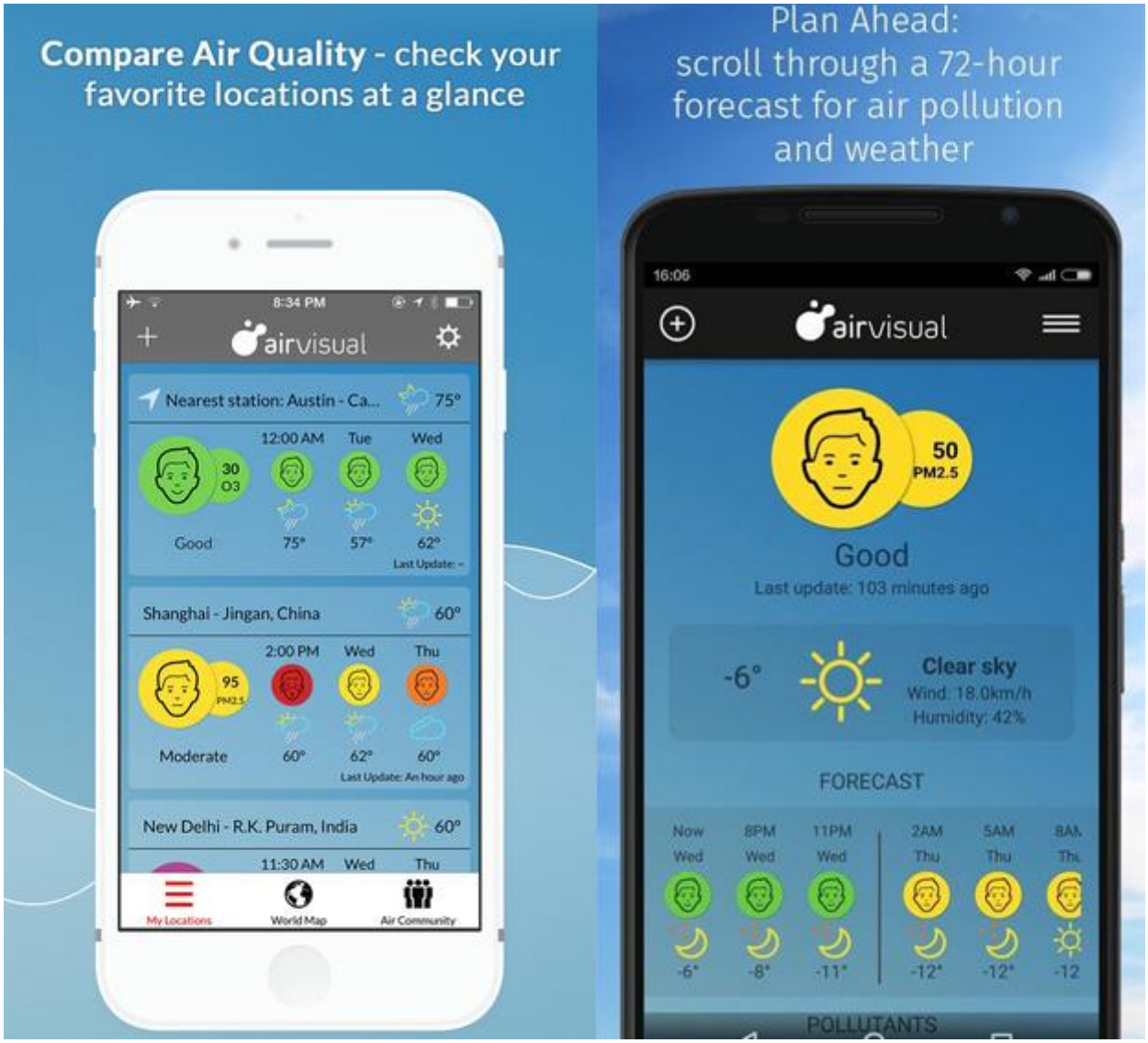


Рисунок 1.1 – Інтерфейс мобільного додатку AirVisual

Додаток Plume Labs надає детальну інформацію про рівень забруднення повітря у різних містах світу. Він також містить функцію прогнозування, що дозволяє користувачам планувати свою активність на основі очікуваних змін якості повітря. Використовуючи передові технології обробки даних і глобальне охоплення.

Plume Labs допомагає користувачам дізнатися про стан навколишнього середовища і вжити необхідних заходів для захисту свого здоров'я, Plume Labs допомагає підвищити точну і актуальну обізнаність громадськості про екологічні проблеми.

Також додаток підтримує актуальні дані для широкого кола регіонів, що потребує значних ресурсів для збору і обробки інформації, а алгоритми машинного навчання допомагають прогнозувати зміни якості повітря на основі історичних даних та поточних тенденцій, також зазвичай надає прогноз якості повітря на кілька днів вперед (рисунок 1.2).

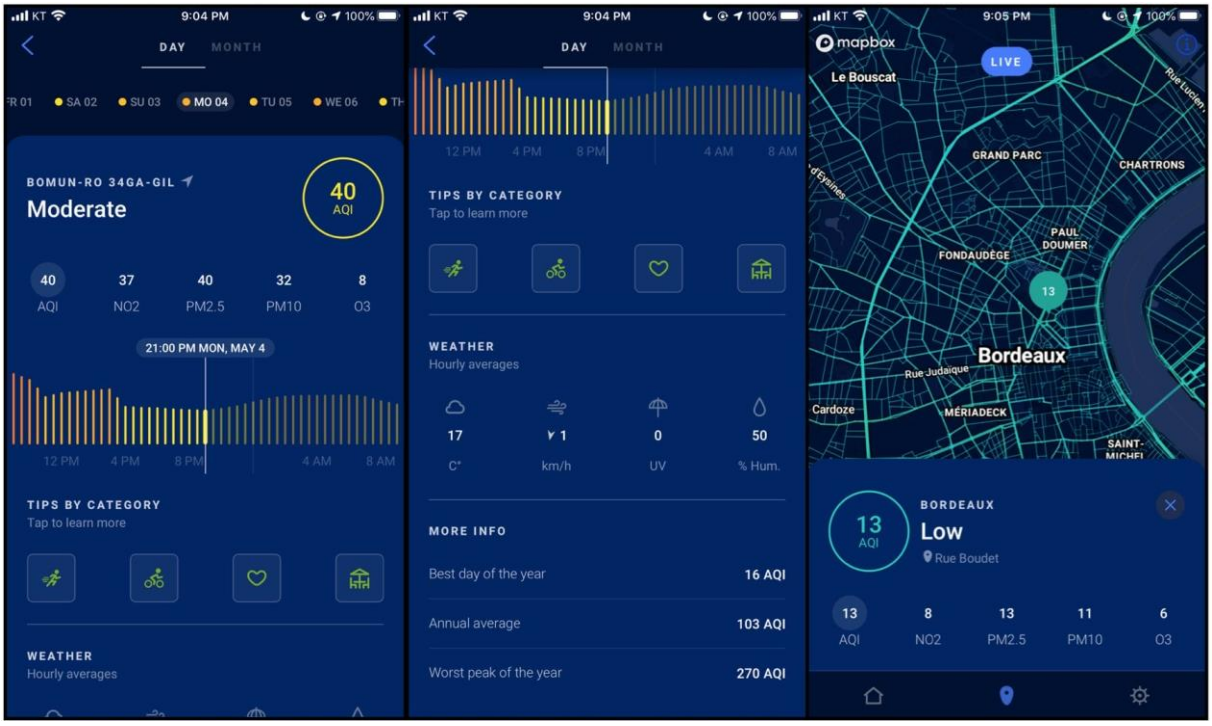


Рисунок 1.2 – Інтерфейс мобільного додатку Plume Labs

Веб-сервіси, такі як AirNow та World Air Quality Index, також надають доступ до даних про якість повітря в режимі реального часу. AirNow є офіційним веб-сервісом Агентства з охорони навколишнього середовища США

(EPA), який надає дані про якість повітря у різних регіонах країни. Сервіс використовує дані зі стаціонарних станцій моніторингу та мобільних пристроїв (рисунок 1.3).

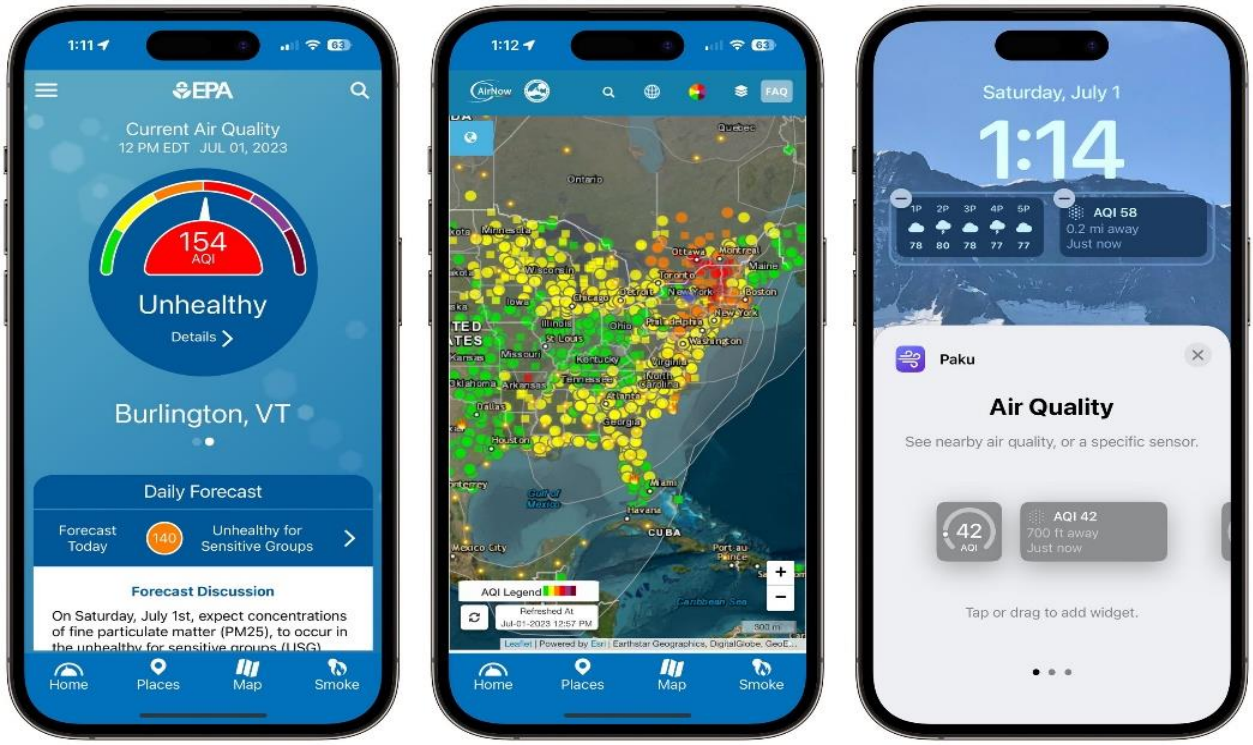


Рисунок 1.3 – Інтерфейс мобільного додатку AirNow

World Air Quality Index надає глобальні дані про якість повітря з тисяч станцій по всьому світу. Сервіс також містить інтерактивну карту, що дозволяє користувачам легко знаходити інформацію про рівень забруднення у конкретних місцях (рисунок 1.4).

Аналіз iOS додатків для моніторингу якості повітря виявив кілька ключових функцій, що надаються користувачам:

– Інформація про поточний рівень забруднення повітря;

– Прогноз якості повітря на кілька днів вперед;

– Сповіщення про критичні рівні забруднення;

– Інтерактивні карти забруднення;

– Історичні дані та тренди;

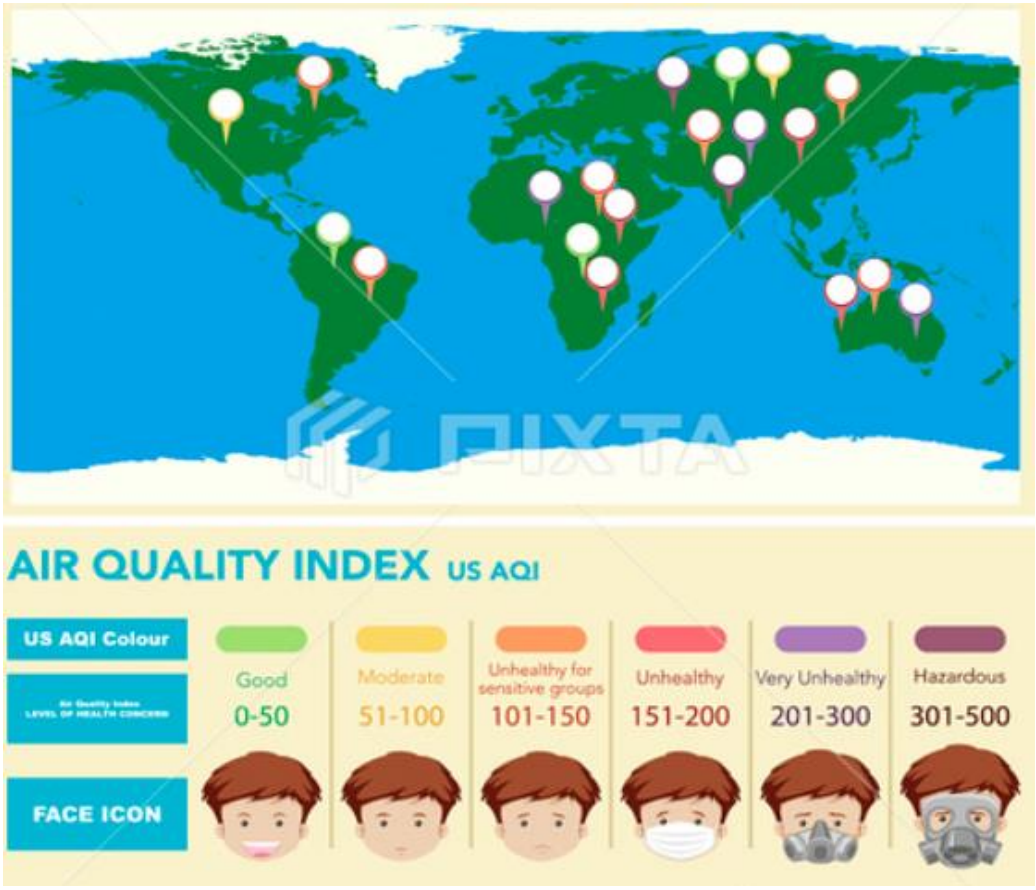


Рисунок 1.4 – Інтерфейс мобільного додатку World Air Quality Index

Додаток AirMatters надає інформацію про якість повітря з понад 180 країн. Він також інтегрується з розумними пристроями, такими як очищувачі повітря, для автоматичного регулювання параметрів на основі якості повітря, має високий рейтинг серед користувачів (рисунок 1.5).

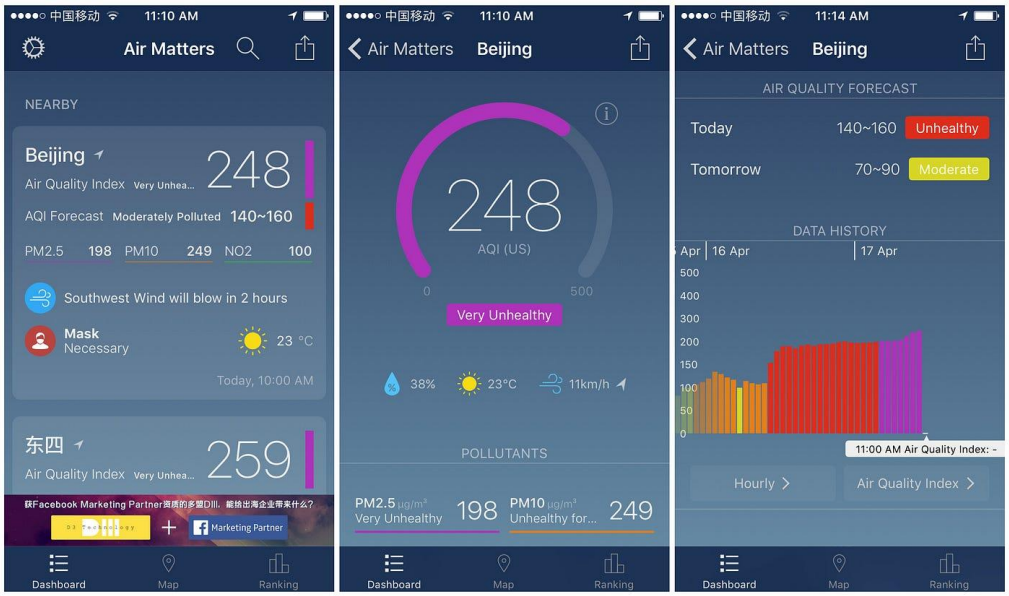


Рисунок 1.5 – Інтерфейс мобільного додатку AirMatters

Додаток BreezoMeter використовує дані з декількох джерел, включаючи державні установи та сенсорні мережі, для надання високоточної інформації про якість повітря в режимі реального часу. Він також пропонує індивідуальні рекомендації для користувачів з проблемами зі здоров'ям. Використовує стандартний індекс якості повітря, що дозволяє легко інтерпретувати дані про рівень забруднення повітря (рисунок 1.6).

Рисунок 1.6 – Інтерфейс мобільного додатку BreezoMeter

Якщо ми порівняємо функціональність різних додатків для iOS і точність даних, найбільш поширені додатки мають схожі базові функції, але відрізняються додатковими функціями і простотою використання.

На основі проведеного огляду літератури та аналізу існуючих рішень можна зробити наступні висновки:

– Якість повітря є важливим екологічним показником, що впливає на здоров'я населення;

– Існує багато методів і технологій для моніторингу якості повітря, включаючи стаціонарні станції, мобільні лабораторії, супутникові спостереження та сенсорні мережі;

– Мобільні додатки та веб-сервіси для моніторингу якості повітря є популярними інструментами, які надають користувачам актуальну інформацію про погоду;

– Аналіз існуючих додатків для iOS показує, що вони мають схожі базові функції, але мають додаткові функції та різну точність даних;

– Подальший розвиток технології моніторингу якості повітря повинен враховувати потреби користувачів у точних, надійних та корисних інструментах для отримання інформації про стан повітря;

Ці результати можуть бути використані для розробки нових рішень або поліпшення існуючого моніторингу якості повітря technologies.In зокрема, він дозволяє створювати більш ефективний і зручний додаток для iOS.

1.3 Принципи розробки мобільних додатків для iOS

Розробка мобільних додатків для iOS включає використання архітектури, що забезпечує масштабованість, зручність підтримки та ефективність роботи. Основні компоненти архітектури:

– View (Вигляд) - відповідає за відображення даних користувачу;

– Model (Модель) - управляє даними та логікою додатку;

– ViewModel (Вигляд моделі) - посередник між View і Model, що забезпечує їх взаємодію;

Для розробки iOS додатків використовуються мови програмування Swift або Objective-C.

У нашому проєкті ми віддамо перевагу Swift, оскількі ця мова програмування є більш новою, швидкою і потужною.

У той час як Objective-C – є вже більш старою, та підходить для підтримки додатків на старих версіях ios.

Використання Swift допоможе зробити актуальний проєкт (рисунок 1.7).

Рисунок 1.7 – Мова програмування Swift

Також для реалізації проєкту необхідне інтегроване середовище розробки Xcode, в ньому ми і будемо розробляти наш додаток, а також перевіряти недоліки (рисунок 1.8).

Рисунок 1.8 – Інтегроване середовище розробки Xcode

Розробка програми, що відображає дані про якість повітря, зібрані датчиком BME680. Додаток отримує та обробляє дані з сервера через API у форматі JSON та деархівує ці значення.

Далі ми можемо відображати ці дані на екрані у вигляді кругових графіків та індикаторів (рисунок 1.9).

Програми можуть взаємодіяти з обладнанням через різні інтерфейси, такі як Bluetooth та Wi-Fi, для збору даних з датчиків та інших пристроїв. Розробка iOS додатку, що використовує Bluetooth для отримання даних про якість повітря з пристрою на базі ESP32 і сенсора BME680. Додаток періодично запитує дані з пристрою і відображає їх у реальному часі.

Виходячи з теоретичних аспектів та аналізу поточного рішення, можна

зробити висновок, що використання платформи ESP32 з датчиком BME680 є ефективним способом збору та обробки даних про якість повітря. Розробка мобільного додатка для iOS, який приймає та відображає ці дані, може надати користувачам актуальну інформацію про якість повітря та допомогти у прийнятті рішень щодо охорони здоров'я та навколишнього середовища, надаючи актуальну інформацію у зручний для користувача час.

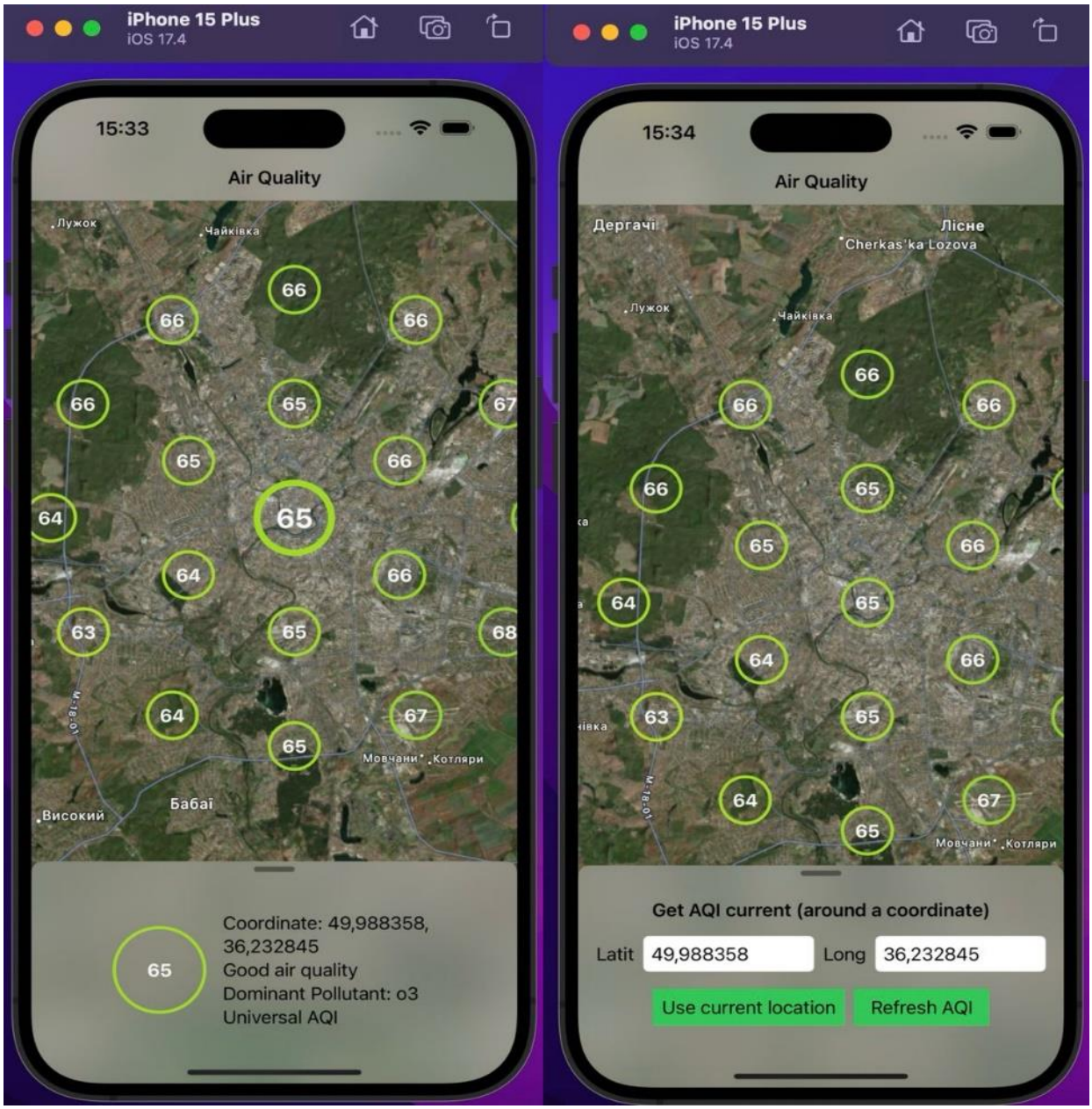


Рисунок 1.9 – Приклад відображення зібраних даних у додатку

2 ПРОЄКТУВАННЯ ДОДАТКА

2.1 Опис бази даних та моделі даних

Для демонстрації роботи додатка у цій дипломній роботі використовується Google AQI API замість власного сервера. Це рішення обрано з метою спрощення розробки та зосередження на функціональності додатка. Написання сервера для цього додатка стане темою моєї магістерської дипломної роботи.

Google AQI API надає доступ до даних про якість повітря в режимі реального часу. За допомогою цього API можна отримувати інформацію про основні показники якості повітря, такі як тверді частинки (PM2.5 і PM10), озон (O_3), двоокис азоту (NO_2), двоокис сірки (SO_2) та оксид вуглецю (CO).

Модель даних додатка буде базуватися на структурі, яку надає Google AQI API. Дані з API будуть оброблятися та відображатися у додатку.

Дані з API будуть отримуватися через генеруючий ключ (API key), які можуть бути реалізовані за допомогою методу AirQualityClient або сторонніх бібліотек, таких як XCAAQI.

Дані будуть парситися у відповідні моделі та відображатися у додатку. Цей підхід дозволяє ефективно демонструвати роботу додатку, отримуючи актуальні дані про якість повітря з надійного джерела, такого як Google AQI API.

Функціональні вимоги до роботи:

– Збір даних - додаток повинен отримувати дані про якість повітря з Google AQI API, пристрою на базі ESP32 та сенсора BME680 через Bluetooth або Wi-Fi;

– Відображення даних - додаток має відображати поточні дані про якість повітря у режимі реального часу. Дані будуть представлені у вигляді кружечків, які змінюють свій колір залежно від значення AQI: чим гірше якість повітря (ближче до 0), тим червоніший кружечок, чим краще (ближче до 100), тим зеленіший;

– Детальна інформація - при натисканні на кружечок відображається детальна інформація, включаючи координати (довгота та широта), якість повітря та домінуючий забруднювач;

– Налаштування - користувач має мати можливість вводити координати для отримання даних про якість повітря у потрібному місці;

2.2 Архітектура додатка - (MVVM)

Модель - представлення, представлення, модель (MVVM) – це архітектурний шаблон, що розділяє логіку додатка на три основні компоненти – Model, View, та ViewModel. Це забезпечує модульність, полегшує тестування та підтримку додатку.

Model містить бізнес-логіку та дані додатку. В даному випадку це може бути структура, яка зберігає дані про якість повітря, отримані з API (рисунок 2.1).

```
8  import Foundation
9  import MapKit
10 import Observation
11 import SwiftUI
12 import XCAAQI
13
14 enum LocationStatus: Equatable {
15     case requestLocation
16     case locationAutorized(String)
17     case error(String)
18     case requestAQIConditions
19     case standby
20 }
21
22 @Observable
23 class ViewModel: NSObject {
24     let locationManager = CLLocationManager()
25     let aqiClient = AirQualityClient(apiKey: "AIzaSyA610Rl_yWfGJctGLEiZSZdCl43zDaWh6c")
26     let coordinatesFinder = CoordinatesFinder()
27
28     var currentLocation: CLLocationCoordinate2D?
29     var locationStatus = LocationStatus.requestLocation
30     var position: MapCameraPosition = .automatic
```

Рисунок 2.1 – Приклад коду у розділі Model

View відповідає за відображення даних та взаємодію з користувачем. Саме цей розділ надає варіативність вибори дизайну.

У нашому випадку це інтерфейс користувача, реалізований за допомогою SwiftUI, що в свою чергу надає широкий спектр можливостей:

– Дизайн карти;

– Можливість масштабування карти;

– Колір тексту;

– Розмір тексту;

– Колір кнопок;

– Розмір кнопок ;

– Різні плавні анімації;

– Додаткові вікна;

– Можливість переключення між денним та нічним режимами;

Тому цей розділ є обгорткою нашого додатку (рисунок 2.2).

```
7  import MapKit
8  import SwiftUI
9  import XCAAQI
10
11 struct ContentView: View {
12     @State var vm = ViewModel()
13     var body: some View {
14
15         Map(position: $vm.position, selection: $vm.selection) {
16             ForEach(vm.annotations) { aqi in
17                 Annotation(aqi.aqiDisplay, coordinate: aqi.coordinate) {
18                     CircleViewAqi(aqi: aqi, isSelected: aqi ==
                           vm.selection)
19                 }
20                 .tag(aqi)
21                 .annotationTitles(.hidden)
22             }
23         }
24         .mapStyle(.hybrid(elevation: .flat, pointsOfInterest: .all,
               showsTraffic: false))
25         //.mapControls {
26            // MapUserLocationButton()
27           //  MapCompass()
28        // }
29         .sheet(isPresented: .constant(true)) {
30             ScrollView {
31                 VStack{
```

Рисунок 2.2 – Приклад коду у розділі View

ViewModel: Посередник між Model та View, що забезпечує зв'язок та синхронізацію даних між ними (рисунок 2.3).

```swift
import Foundation
import CoreLocation

struct CoordinatesFinder {

    let R = 6371000.0 // radius of the Earth in meters
    let pi = 3.141592653589793 // pi constant

    // Define a function to convert degrees to radians
    func deg2rad(_ deg: Double) -> Double {
        return deg * pi / 180
    }

    // Define a function to convert radians to degrees
    func rad2deg(_ rad: Double) -> Double {
        return rad * 180 / pi
    }

    // Define a function to find coordinates on a circle around a given point
    func findCoordinates(_ coordinate: CLLocationCoordinate2D, r: Double, n: Int) ->
        [CLLocationCoordinate2D] {
```

Рисунок 2.3 – Приклад коду у розділі ViewModel

2.3 Інтерфейс користувача (UX/UI дизайн)

SwiftUI пропонує гнучкий функціонал, який дозволяє створювати сучасний, естетичний та інтуїтивно зрозумілий інтерфейс користувача. Завдяки цьому фреймворку, ми можемо легко реалізувати гнучкий та красивий UX/UI дизайн для відображення поточних даних про якість повітря.

Для нових користувачів, які не мають досвіду роботи з подібними додатками, дизайн головного екрану повинен бути мінімальним і простим. Поточні дані про якість повітря відображаються на головному екрані у вигляді кольорового кола. Колір кола змінюється в залежності від якості повітря: від зеленого (хорошої якості) до червоного (поганої якості). Клацніть коло, щоб переглянути детальну інформацію, включаючи координати (довготу та широту), якість повітря та переважаючі забруднювачі (рисунок 2.4).

Користувальницький інтерфейс Swift дозволяє створювати компоненти, які можна легко використовувати повторно, що значно спрощує розробку додатків та їх підтримку.

Використання різних методів для реалізації гарного інтерфейсу допомагає розробити дійсно щось цікаве.

Рисунок 2.4 – Інформація про натискані на коло

Завдяки широкому функціоналу ми можемо створювати динамічне відображення даних, що зробить наш додаток більш зручним та привабливим.

Наприклад, для відображення якості повітря коло реалізується як окремий компонент за допомогою RGB технології в залежності від значення AQI, також це впливає і на його розмір та масштабованість (рисунок 2.5).

```
Circle()
    .stroke(Color(red: aqi.color.red, green: aqi.color.green,
        blue: aqi.color.blue), lineWidth: isSelected ? 4 : 3)
    .frame(width: size.width, height: size.height)
    .overlay {
        Text(aqi.aqiDisplay)
            .foregroundStyle(.white)
            .fontWeight(.bold)
    }
    .scaleEffect(isSelected ? CGSize(width: 1.5, height: 1.5)
        : CGSize(width: 1, height: 1))
```

Рисунок 2.5 – Реалізація кружечків за допомогою AQI даних та RGB технології

Завдяки гнучким можливостям SwiftUI, ми можемо створити інтерфейс користувача, який є одночасно функціональним та естетичним. Мінімалістичний дизайн головного екрану забезпечить зручність для нових користувачів. Інтерактивні елементи та адаптивність SwiftUI дозволяють легко налаштовувати та розширювати функціональність додатку.

2.4 Вибір і обґрунтування технологій та інструментів

Swift – це основна мова програмування для розробки iOS додатків, створена компанією Apple. Вперше анонсована у 2014 році на конференції WWDC, Swift швидко здобула популярність завдяки своїй простоті, ефективності та безпеці. Розробка Swift почалася у 2010 році Крісом Латтнером, який був також одним із основних розробників компілятора LLVM, а згодом до нього приєдналися інші інженери з Apple. Однією з основних переваг Swift є його сучасний синтаксис, який робить код більш читабельним та легким для розуміння. Мова підтримує інтерактивне середовище Playground, що дозволяє розробникам експериментувати з кодом та бачити результати змін у реальному часі без необхідності компіляції всього проекту. Це значно покращує процес навчання та прототипування. Swift дозволяє розробникам писати безпечний та надійний код, знижуючи ймовірність помилок завдяки таким функціям, як опціональні типи, автоматичне управління пам’яттю та строгий контроль типів (рисунок 2.6).

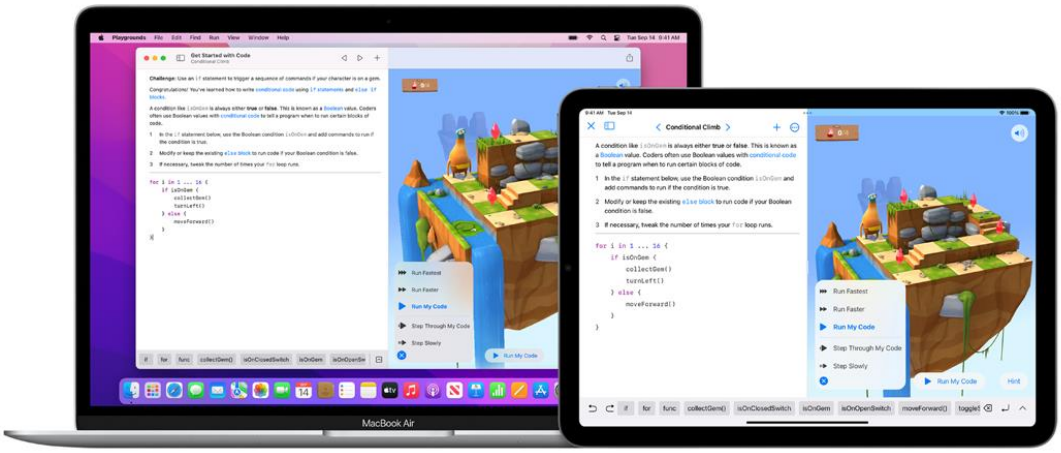


Рисунок 2.6 – інтерактивне середовище Playground

Опціональні типи допомагають уникнути поширених помилок, пов'язаних з роботою з нульовими значеннями, забезпечуючи явну обробку таких ситуацій. Автоматичне управління пам’яттю, яке використовує механізм ARC (Automatic Reference Counting), знімає з розробників необхідність вручну керувати пам’яттю, що значно знижує ризик витоків пам’яті та інших пов’язаних

проблем (рисунок 2.7).

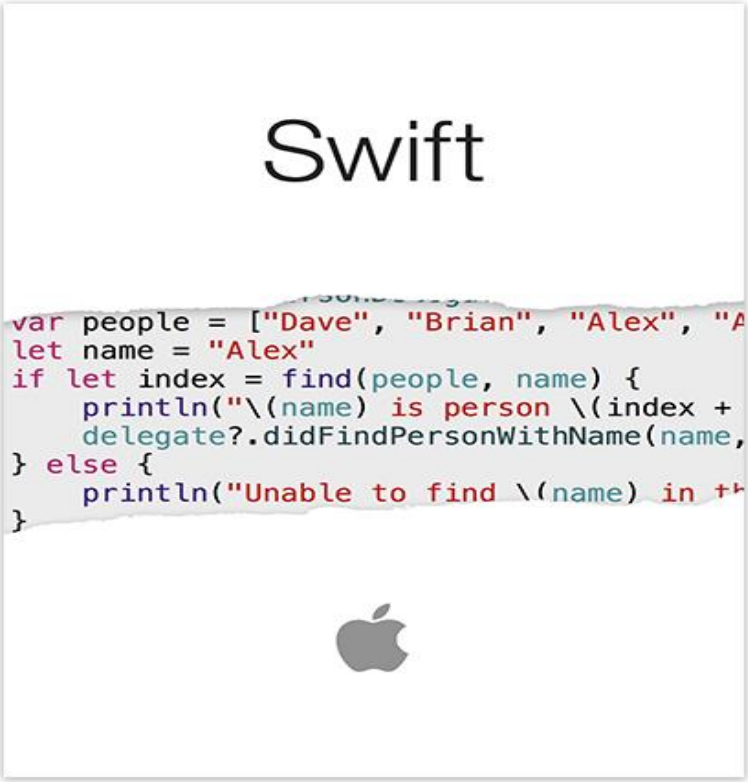


Рисунок 2.7 – мова програмування Swift

Swift також підтримує функціональне програмування, що дозволяє використовувати замикання, високого порядку функції та інші функціональні підходи для створення більш модульного та гнучкого коду. Це робить Swift універсальною мовою, яка підходить для широкого спектру завдань, від системного програмування до розробки високорівневих додатків.

Мова Swift постійно розвивається та оновлюється. Спільнота розробників активно сприяє її розвитку через відкритий вихідний код проекту, що був відкритий у грудні 2015 року. Це дозволяє розробникам усього світу брати участь у вдосконаленні мови та адаптації її до своїх потреб.

Використовуючи Swift, розробники можуть створювати продуктивні та ефективні додатки для різних платформ Apple, включаючи iOS, macOS, watchOS та tvOS. Завдяки своїм численним перевагам, Swift став вибором номер один для багатьох розробників, які прагнуть створювати сучасні, швидкі та надійні додатки для екосистеми Apple.

SwiftUI – це фреймворк, розроблений компанією Apple для створення користувацького інтерфейсу. Вперше представлений на WWDC 2019, SwiftUI дозволяє розробникам створювати сучасні, інтерактивні та адаптивні інтерфейси з меншою кількістю коду. Використовуючи SwiftUI, розробники можуть працювати у більш інтуїтивному середовищі, що значно прискорює процес розробки та полегшує підтримку коду (рисунок 2.8).

Рисунок 2.8 – Фреймворк SwiftUI

Однією з головних особливостей SwiftUI є його декларативний підхід до побудови інтерфейсу. Замість того, щоб описувати покроково, як створювати та оновлювати інтерфейс, розробники описують, що інтерфейс має робити. Це робить код більш чистим та зрозумілим, дозволяючи легко вносити зміни та додавати новий функціонал. Наприклад, для створення текстового поля з певними стилями достатньо написати такий невеликий код (рисунок 2.9).

```
Text("Hello, World!")
.font(.largeTitle)
.foregroundColor(.blue)
```

Рисунок 2.9 – Приклад створення текстового поля

Результатом відтворення цього коду стане відображення на канвасі, яке має змогу динамічно змінюватися залежно від значень (рисунок 2.10).

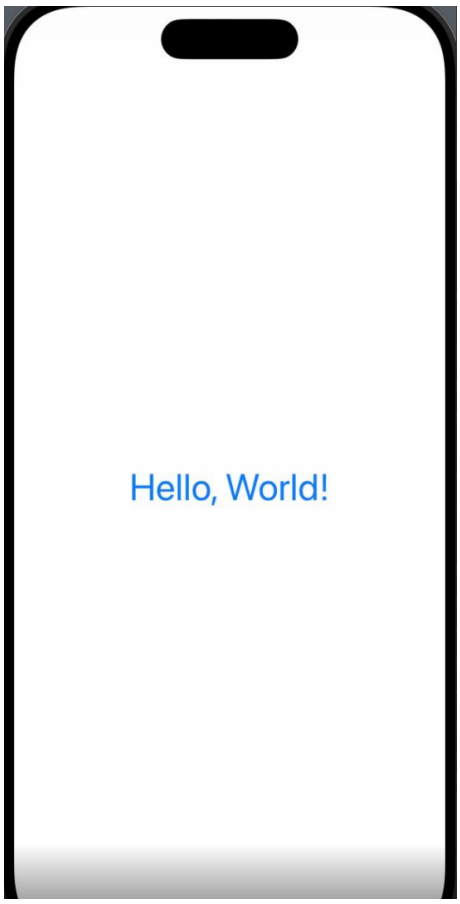


Рисунок 2.10 – Відображення тестового поля на екрані

SwiftUI забезпечує автоматичну сумісність з різними пристроями Apple. Це означає, що розробники можуть створювати інтерфейси, які будуть виглядати та працювати коректно на iPhone, iPad, Apple Watch та Apple TV, без необхідності писати окремий код для кожного типу пристрою. SwiftUI автоматично підлаштовує компоненти інтерфейсу під різні розміри екранів та орієнтації, що дозволяє створювати адаптивні додатки з мінімальними зусиллями.

Фреймворк також підтримує створення анімацій та переходів між станами інтерфейсу. Завдяки цьому розробники можуть легко додавати візуальні ефекти та анімації, які роблять додатки більш привабливими та зручними для користувачів. Наприклад, анімацію зміни кольору тексту можна реалізувати наступним чином (рисунок 2.11).

Здавалося б створення невеликого коду дає змогу побачити результат анімації на наглядному прикладі, це і є широкий функціонал та продуктивність роботи цього фреймворку.

```
4       @State private var isHighlighted = false
5       var body: some View {
6           Text("Tap me")
7               .padding()
8               .background(isHighlighted ? Color.red :
                    Color.blue)
9               .animation(.easeInOut(duration: 0.5))
10              .onTapGesture {
11                  isHighlighted.toggle()
12              }
13      }
14  }
```

Рисунок 2.11 – Приклад створення анімації з кнопкою

При натисканні на кнопку з тестом через пів секунди з анімацією кнопка змінить свій колір на інший.

Реалізація анімації:

– Створення кнопки з текстом;

– Надання кольору кнопці ;

– Анімація при змінні кольору;

– Змога натискання на кнопку;

Цю анімацію ми також маємо змогу побачити завдяки оновленню значень у канвасі (рисунок 2.12).

Рисунок 2.12 – Відображення анімації зміни кольору кнопки

SwiftUI також інтегрується з інструментом Xcode, надаючи можливість використовувати Canvas для попереднього перегляду інтерфейсу в режимі реального часу. Це дозволяє розробникам бачити зміни у коді негайно, без необхідності запускати додаток на симуляторі чи реальному пристрої. Така інтерактивність значно прискорює процес розробки та дозволяє швидше знаходити та виправляти помилки

Canvas – це потужний інструмент для створення графічних інтерфейсів. Це полегшує інтеграцію складних графічних елементів у ваш додаток, забезпечуючи при цьому високу продуктивність та простоту використання. Завдяки декларативному підходу інтерфейсу Swift, Canvas створює процес створення графічних інтерфейсів інтуїтивно зрозумілим та ефективним способом. (рисунок 2.13).

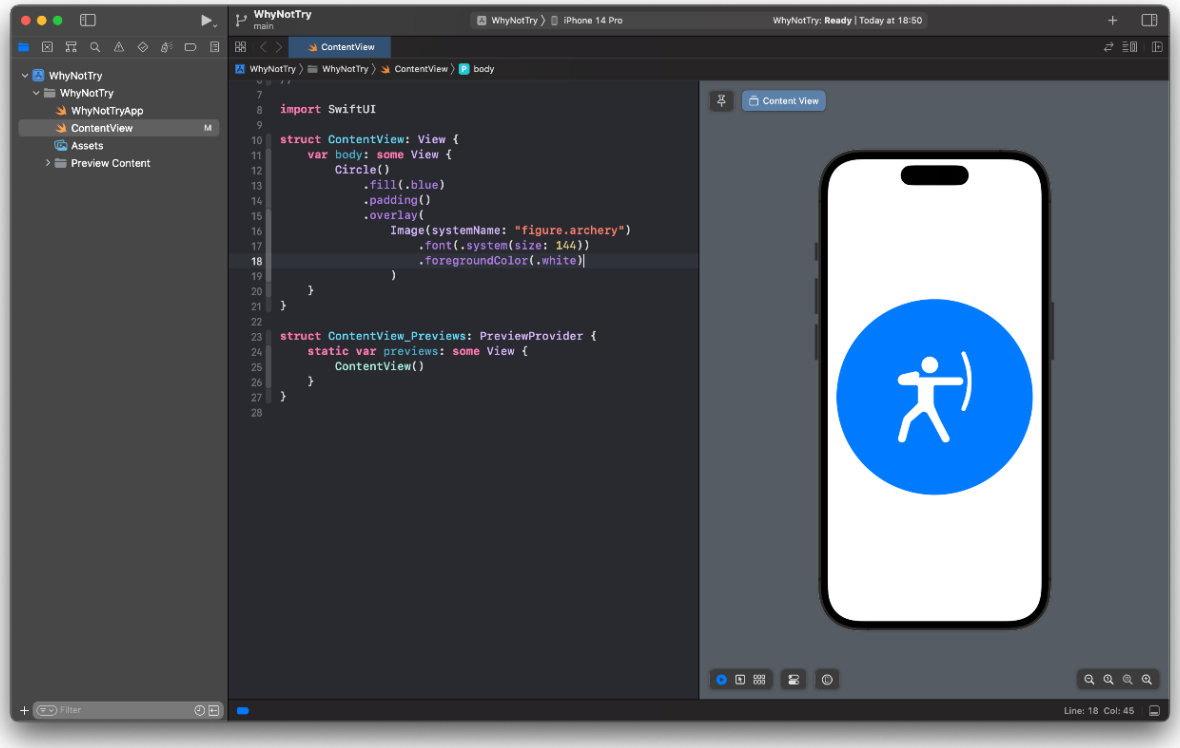


Рисунок 2.13 – Інтегрування SwiftUI та Xcode на Canvas

Фреймворк підтримує модульність, що дозволяє розробникам розбивати інтерфейс на невеликі багаторазові компоненти. Це покращує структуру коду, роблячи його більш гнучким і простим в обслуговуванні. Кожен компонент

можна перевірити індивідуально, а якість кінцевого продукту можна легко покращити.

Завдяки своїм численним перевагам SwiftUI став важливим інструментом для багатьох розробників, які прагнуть створити сучасні, ефективні та зручні програми для екосистеми Apple.

Xcode – це інтегроване середовище розробки (IDE), створене Apple для розробки додатків на платформах iOS, macOS, watchOS та tvOS. Спочатку випущений в 2003 році, Xcode постійно оновлюється і вдосконалюється, щоб задовольнити потреби новітніх розробників і підтримувати новітні технології Apple.

Xcode надає візуальний редактор SwiftUI, який дозволяє розробникам перетягувати компоненти та негайно бачити результати. Це значно спрощує процес створення складних користувальницьких інтерфейсів.

Окрім статичного попереднього перегляду, Xcode підтримує попередній перегляд у режимі реального часу, який дозволяє взаємодіяти з інтерфейсом, виконувати анімації та переглядати динамічні зміни. (рисунок 2.14).

Рисунок 2.14 – Xcode IDE

Рефакторинг коду є важливою складовою процесу розробки, яка дозволяє підтримувати чистоту, організованість і зрозумілість коду. Xcode надає низку інструментів і функцій для полегшення цього процесу, що дозволяє розробникам Swift та SwiftUI швидко і ефективно змінювати структуру коду без зміни його зовнішньої поведінки.

Xcode надає інструменти для автоматичного перетворення коду з однієї структури в іншу. Наприклад, конвертація старого синтаксису Swift в новий, або перетворення імперативного коду в декларативний.

Рефакторинг в Xcode для SwiftUI допомагає підтримувати чистоту, організованість і зрозумілість коду, що є важливим для довготривалої підтримки та розвитку додатків.

Однією з ключових переваг Xcode є його комплексність. Це середовище об'єднує в собі всі необхідні інструменти для створення, тестування та налагодження додатків. Текстовий редактор Xcode підтримує підсвічування синтаксису, автодоповнення коду та рефакторинг, що полегшує написання та підтримку коду. Комп'ютер в Xcode оптимізований для роботи з мовами Swift і Objective-C, забезпечуючи швидку компіляцію та мінімальний час очікування (рисунок 2.15).

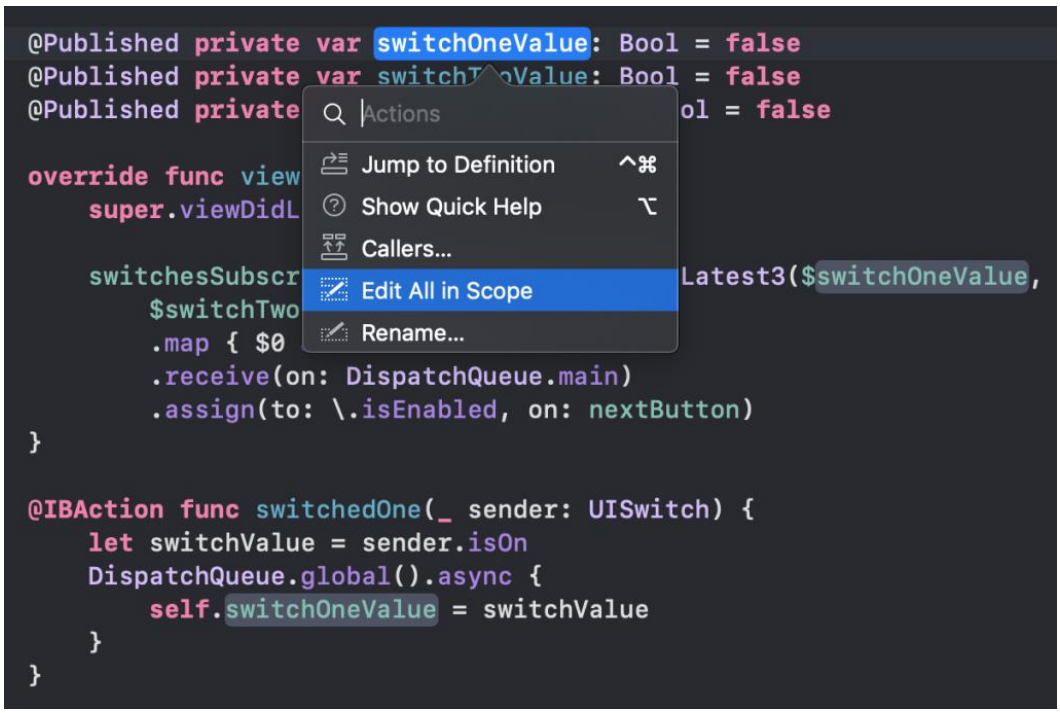


Рисунок 2.15 – підсвічування синтаксису в Xcode

Xcode відкадчик – це незмінний інструмент для розробників для виявлення та виправлення помилок, оптимізації коду та покращення якості додатків. Потужні функції, такі як точки зупинки, навігація кодом, посилання на змінні та консоль LLDB, дозволяють розробникам ефективно та швидко налагоджувати свої програми, забезпечуючи точну та стабільну роботу.

Основні функції відкладчика:

– Step Over - виконати поточний рядок коду і перейти до наступного;

– Step Into - увійти всередину функції або методу;

– Step Out - вийти з поточної функції або методу;

Особливості відкладчика:

– Перегляд значень локальних змінних;

– Перегляд властивостей об'єктів;

– Можливість зміни значень змінних на льоту;

Xcode надає спеціальні інструменти для налагодження інтерфейсу користувача, які допомагають виявляти та виправляти проблеми з розміткою та відображенням UI елементів.

Відладчик Xcode надає розробникам можливість детально аналізувати код, знаходити та виправляти помилки. Завдяки інтеграції з LLDB, розробники можуть використовувати точки зупинки, переглядати значення змінних у реальному часі та виконувати покрокове виконання коду. Це значно спрощує процес виявлення та усунення проблем (рисунок 2.16).

Xcode також підтримує інтеграцію з іншими інструментами Apple, такими як Interface Builder, що дозволяє розробникам створювати та налаштовувати користувальницькі інтерфейси за допомогою візуального редактора. Interface Builder надає можливість створювати інтерфейси за допомогою перетягування, що значно прискорює процес розробки і спрощує внесення змін.

Вбудовані інструменти тестування Xcode дозволяють розробникам автоматизувати процес тестування своїх програм. За допомогою XCTest розробники можуть створювати модульні тести, інтеграційні тести та тести інтерфейсу користувача для перевірки функціональності програми. Це забезпечує

високу якість кінцевого продукту та допомагає швидко виявляти та виправляти помилки.

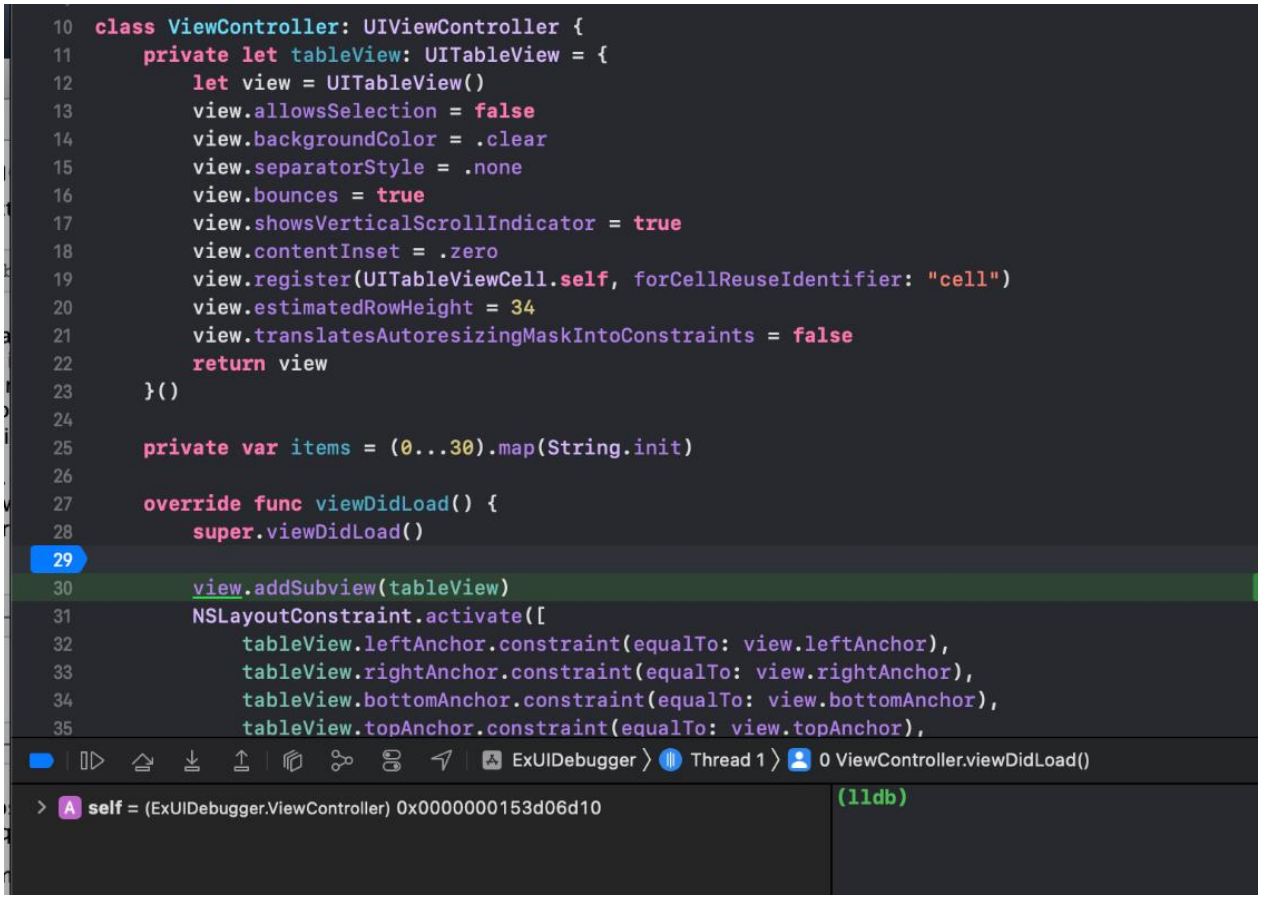


Рисунок 2.16 – Відладчик Xcode

Xcode підтримує системи керування версіями, такі як Git, що дозволяє розробникам відстежувати зміни коду, працювати в команді та керувати версіями проектів. Інтеграція з Git дозволяє використовувати репозиторій безпосередньо з Xcode для виконання всіх необхідних операцій.Це значно спрощує роботу з кодом і покращує організацію проекту.

Остання версія Xcode включає інструменти для роботи з найновішими технологіями Apple, такими як SwiftUI, Combine та Catalyst. Це дозволяє розробникам створювати сучасні програми, які використовують розширені функції та можливості платформи Apple.

Маючи всі ці функції, Xcode є потужним інструментом розробки додатків, який забезпечує повний цикл розробки, від створення коду до тестування

та налагодження. Це робить Xcode важливим інструментом для всіх розробників, які працюють в екосистемі Apple.

Xcode Simulator – це інструмент, вбудований у Xcode, який дозволяє розробникам тестувати програми на різних моделях iPhone, iPad, Apple Watch та Apple TV без необхідності фізичного пристрою. Симулятор Xcode, вперше представлений з Xcode 3.0 у 2007 році, постійно вдосконалюється, щоб забезпечити точне відтворення прикладного середовища, і його можна використовувати для виявлення та виправлення помилок навіть на етапі розробки (рисунок 2.17).

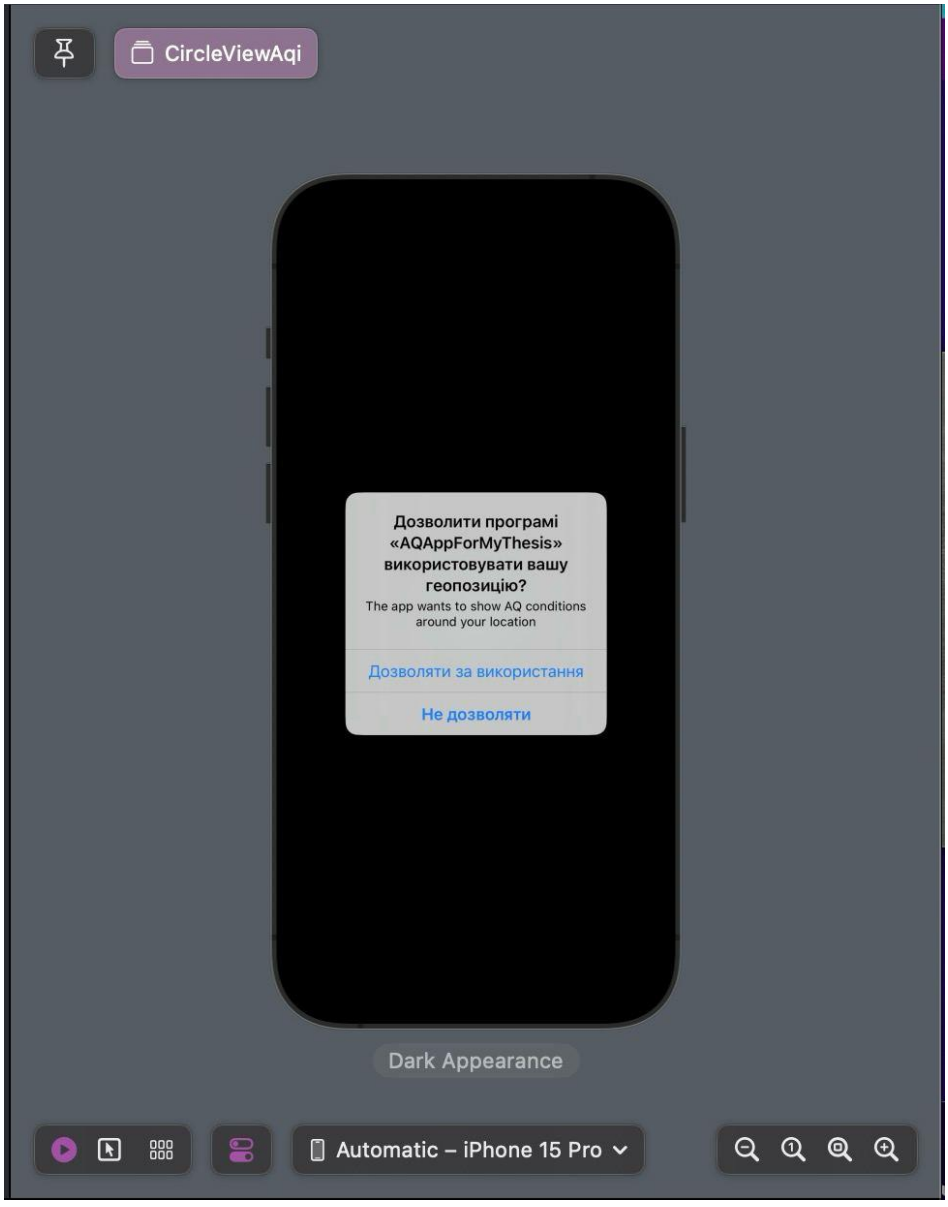


Рисунок 2.17 – Xcode Simulator

Xcode Simulator точно відтворює не тільки апаратні характеристики пристрою, але і роботу операційної системи і користувальницького інтерфейсу. Це дозволяє розробникам тестувати сумісність і функціональність додатків на різних пристроях і версіях iOS, macOS, watchOS і tvos. Зокрема, симулятор підтримує різні розміри екрана, орієнтацію та налаштування пристрою, щоб ви могли перевірити адаптивні конструкції та різні випадки використання рисунок (2.18).

Рисунок 2.18 – Налаштування Xcode Simulator

Окрім тестування інтерфейсу, Xcode Simulator також підтримує функції тестування, такі як GPS, камери, гіроскопи, акселерометри та інші апаратні компоненти. Це дозволяє розробникам моделювати Умови використання реального додатка без необхідності мати фізичний пристрій з відповідними функціями. Наприклад, розробники можуть імітувати зміни місцезнаходження для тестування функцій, що залежать від місцезнаходження, або використовувати віртуальні камери для тестування функцій, що залежать від доступу до камери.

Xcode Simulator також підтримує інтеграцію з автоматизованими інстру-

ментами тестування, такими як XCTest, які дозволяють розробникам запускати модульні тести та тести інтерфейсу безпосередньо в симуляторі. Це значно спрощує процес автоматичного тестування і забезпечує високу якість кінцевого продукту. Розробники можуть налаштувати симулятор таким чином, щоб він автоматично запускав тести під час створення проекту.Це допоможе вам швидко виявити та виправити помилки.

Ще одна корисна функція Xcode Simulator-це можливість записувати відео та скріншоти. Це дозволяє розробникам створювати демонстрації, налагоджувальні документи та звіти про помилки, які можна легко надіслати колегам або використовувати для проектної документації.

Xcode Simulator також підтримує багатозадачність, дозволяючи запускати кілька симуляторів одночасно. Це особливо корисно, коли ви хочете перевірити взаємодію між програмами або перевірити сумісність програми з іншою версією операційної системи або деки пристрою. Наприклад, розробник може протестувати додаток на iPhone одночасно з іншою версією iOS, щоб переконатися, що він працює послідовно на всіх підтримуваних пристроях.

Завдяки цим функціям Xcode Simulator є незамінним інструментом для розробників, які прагнуть забезпечити високу якість своїх додатків на кожному етапі розробки. Ви можете швидко та ефективно тестувати свої програми, скорочуючи час і зусилля, необхідні для їх тестування на реальних пристроях, і точно відтворюючи середовище своїх додатків, щоб виявляти та виправляти помилки на ранніх етапах розробки.

Xcode Simulator є важливим інструментом для розробників додатків для ios, tvOS та watchOS для ефективного тестування та налагодження програм без використання фізичних пристроїв. Використовуючи можливості симулятора, розробники можуть забезпечити безперебійну роботу своїх додатків на всіх підтримуваних пристроях і в різних середовищах.

Основні Можливості Xcode Simulator:

– Тестування на різних пристроях і версіях iOS;

– Емуляція різних станів мережі;

– Використання фізичних можливостей пристрою;

– Налагодження та аналіз додатків;

– Тестування інтерфейсу користувача;

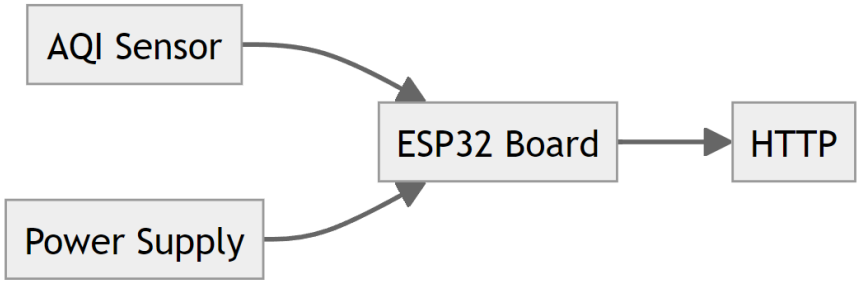
Simulator дозволяє вибирати різні моделі iPhone, iPad, Apple Watch та Apple TV, а також версії операційних систем. Це допомагає переконатися, що додаток працює правильно на різних пристроях та версіях ОС.

Вибір цих технологій для розробки додатків базується на їх перевагах та простоті використання. Swift забезпечує швидкість і безпеку розробки, SwiftUI дозволяє створювати сучасний адаптивний інтерфейс, Xcode надає повний набір інструментів для всіх етапів розробки, а Xcode Simulator дозволяє ефективно тестувати додатки на різних пристроях. Поєднання цих технологій забезпечує повний цикл розробки, від генерації коду до тестування та налагодження, що дозволяє розробляти високоякісні конкурентоспроможні програми.

3 ПРОГРАМНО – АПАРАТНА ЧАСТИНА

3.1 Апаратна частина

В збиранні схеми підключаємо необхідні компоненти (датчик якості повітря BME680, який вимірює температуру, вологість, тиск та газовий опір) до плати ESP32 через відповідні пінові з'єднання. Підключаємо джерело живлення до плати ESP32. Дані з датчика надсилаються на сервер за допомогою HTTP POST запитів рисунок (3.1).



AQI (Air Quality Index) – рівень забруднення повітря, HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі гіпертексту

Рисунок 3.1 – Структурна схема проєкту

Опис компонентів структурної схеми за модулями:

Модуль 1 – плата ESP-WROOM-32, використовується для зчитування даних. Приклад ESP-32 наведено на рисунку 3.2

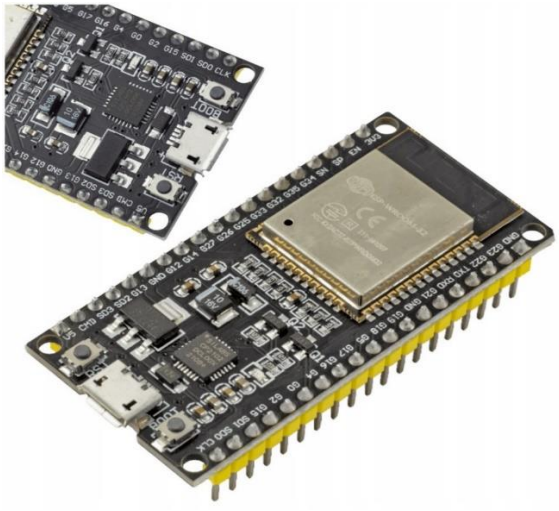


Рисунок 3.2 – Плата ESP-WROOM-32 ESP32 (Wi-Fi + Bluetooth)

Характеристики:

– Напруга живлення: 5 В;

– USB-UART чип: CP2102 / CH9102X;

– Максимальний струм стабілізатора напруги: 800 мА;

– Стандарти Wi-Fi: FCC / CE / IC / TELEC / KCC / SRRC / NCC;

– Протоколи: 802.11 б/с Так/ UK/ D/ Четверта голосна складової деванагарі/ Найближчі/ Закінчення відмінка, що показує володіння або спорідненість/ R (802.11n до 150 Мбіт / с);

–Підтримка та підтримка A-MPDU та A-MSDU з інтервалом захисту 0,4 сек;

– Частотний діапазон: 2,4~2,5 ГГц;

– Bluetooth протоколи: Bluetooth v4.2 BR / EDR и специфікація BLE;

– NZIF – радіостанція з чутливістю: -98 дБм;

– Передатчики: клас – 1, клас – 2 и клас – 3 AFH;

– Аудіо: CVSD и SBC;

– Пристрій та інтерфейс: SD, UART, SPI, SDIO, IxoC, LED PWM, Motor PWM, IxoS, IXC, IR GPIO, датчик датчика, АЦП, ЦАП, LNA-прекурсор;

– Датчики «на борту»: датчик Холла, датчик температури;

– Генератори: кристалічні 26 МГц і 32 кГц

– Живлення мікромодуля: В 2,2~3,6;

– Робочий струм, середній: 80 мА;

– Робочий струм піковий 500 мА;

– Діапазон робочих температур: -40 °C ~ 85 °C;

– Програмне забезпечення: режими Wi-Fi Station / softAP / SoftAP + станція / P2P;

– Захист: WPA / WPA2 / WPA2-Enterprise / WPS;

– Сертифікація: AES/RSA/ECC/SHA;

– Оновлення програмного забезпечення: UART Download / OTA (мережа);

– Разработка: Cloud Server Development / SDK для розробки кастомних

прошивок;

– Мережеві протоколи: IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT;

– Налаштування користувача: набір інструкцій AT, хмарний сервер, додаток для Android / iOS.

Модуль 2 – BME680 (датчик вимірювання якості повітря), використовується для зчитування значень AQI. Приклад BME680 наведено на рисунку 3.3

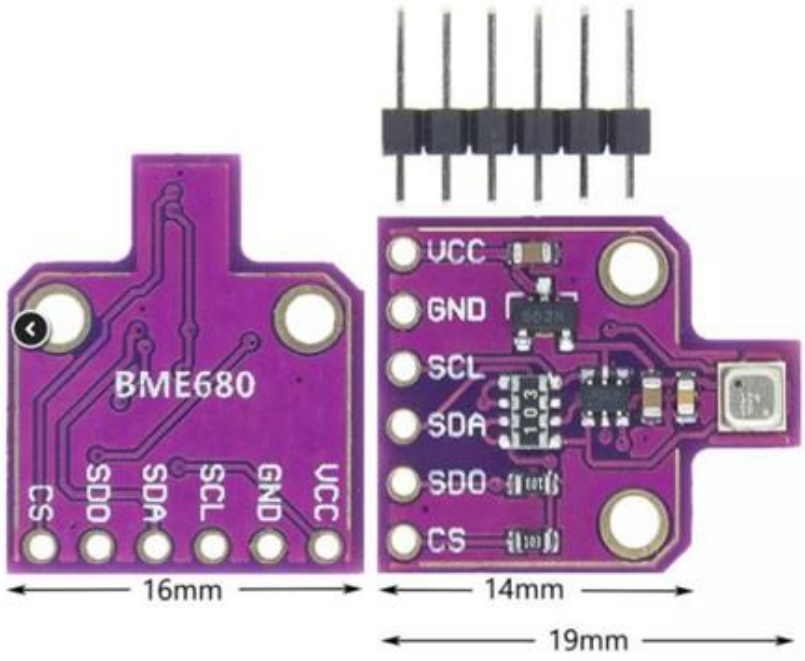


Рисунок 3.3 – датчик вимірювання якості повітря (BME680)

Характеристики:

– Діапазон вимірювання температури: від -40°C до +85°C;

– Діапазон вимірювання вологості: від 0% до 100%;

– Діапазон вимірювання IAQ: від 0 до 500;

– Діапазон вимірювання тиску повітря: від 300 до 1100 гПа;

– Частота вимірювання: за замовчуванням 3 с;

– Робоча напруга: 5 В;

– Середній струм споживання: 12 мА;

– Робоча температура: від -40°C до +85°C;

– Температура зберігання: від -40°C до +125°C;

– Розмір: 12 х 30 мм (рисунок 3.3);

Для збору даних про якість повітря використовуються різні сенсори та платформи, такі як BME680 і плати Arduino або ESP32. Дані збираються через встановлені інтервали часу та зберігаються у вигляді логів або передаються на сервер для подальшої обробки (рисунок 3.4).

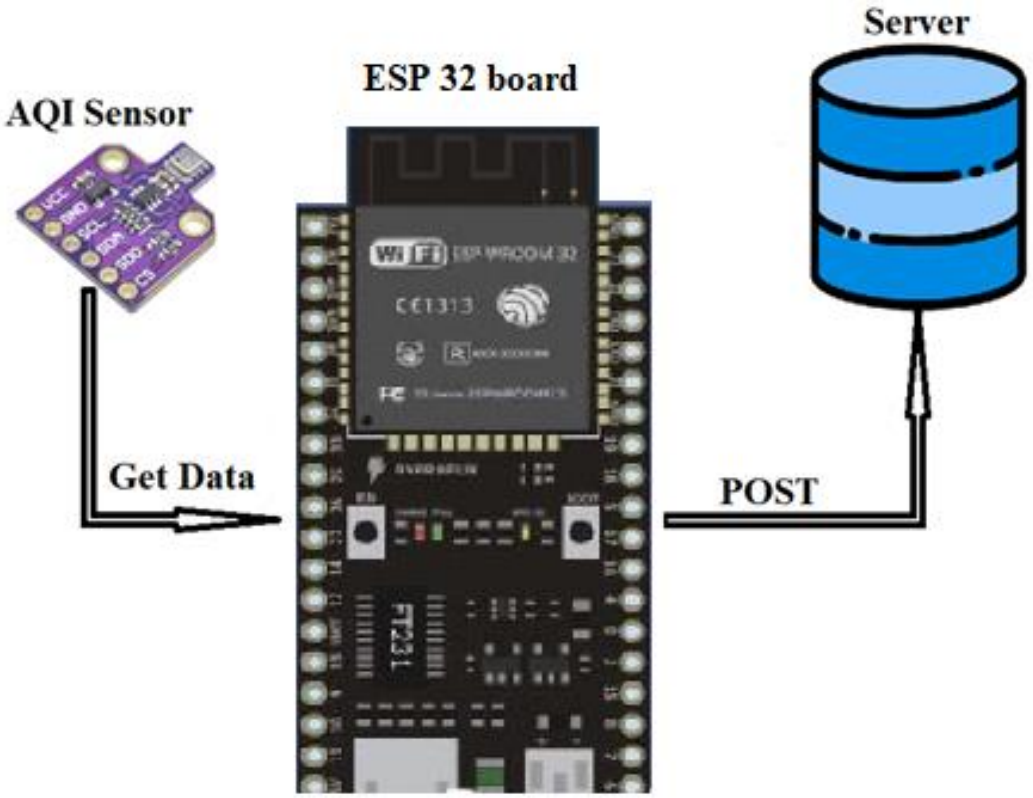


Рисунок 3.4 – Приклад збору даних та їх відправки на сервер

Використання плати ESP32 з підключеним датчиком BME680 для збору даних про температуру, вологість, тиск і концентрацію забруднюючих речовин. Дані можуть зберігатися на SD-карті або надсилатися на сервер через Wi-Fi.

Обробка зібраних даних включає фільтрацію, калібрування, аналіз та візуалізацію. Його можна реалізувати за допомогою програмного забезпечення на комп'ютері або безпосередньо на мікроконтролері. Він обробляє дані з датчика BME680 на esp32, використовуючи бібліотеку для роботи з датчиком. Дані відкалібровані та відфільтровані для усунення шуму, а потім відправлені на сервер для подальшого аналізу та візуалізації за допомогою спеціалізованих інструментів.

3.2 Програмна частина

Для початку розробимо UML діаграму для чіткого плану роботи (рисунок 3.5).

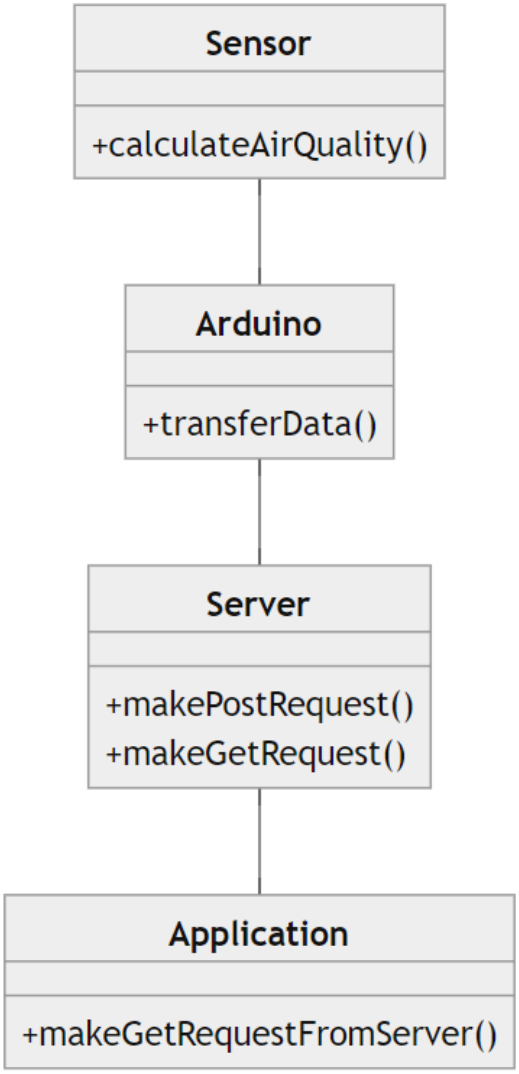


Рисунок 3.5 – UML діаграма проєкту

Є датчик якості повітря (air quality sensor), який зчитує дані про якість повітря (calculateAirQuality). Ці дані передаються на плату ESP32, підключену до датчика через пінові з'єднання. Потім ESP32 підключається до Wi-Fi мережі та робить POST запит на сервер (transferData). Відповідно, з нашого додатка ми робимо GET запит до сервера для отримання даних (makeGetRequestFromServer).

У цьому розділі детально описується процес розробки додатку для моніторингу якості повітря за допомогою плати ESP32 та датчиків BME680. Цей скетч реалізує З'єднання Wi-Fi, збирає дані з датчика та надсилає ці дані на сервер через HTTP-запит.

Перелік алгоритму роботи скетчу:

– Підключення бібліотек;

– З’єднання з мережею Wi-Fi;

– Підключення до сервера;

– Підключення сенсора;

– Підключення функції setup();

– Підключення функції loop();

На початку скетчу підключаються необхідні бібліотеки, які надають нам змогу працювати з Wi – Fi та іншими можливостями:

– WiFi.h – для підключення до Wi-Fi мережі;

– HTTPClient.h – для виконання HTTP-запитів;

– ArduinoJson.h – для створення та обробки JSON-об'єктів;

– Wire.h і Adafruit_BME680.h – для роботи з сенсором BME680;

Бібліотеки, які необхідні для подальшої роботи (рисунок 3.6)

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_BME680.h>
```

Рисунок 3.6 – Підключення бібліотек

Далі вказуються дані для налаштування Wi-Fi мережі (рисунок 3.7).

```
const char* ssid = "MERCUSYS_5G_27";
const char* password = "dk32645dk";
```

Рисунок 3.7 – Необхідні дані для Wi-Fi мережі

Вказуються параметри сервера, до якого будуть відправлятися дані (рисунок 3.8).

```
const char* serverAddress = "http://cleanairapi.com";
const int serverPort = 80;
const String endpoint = "/api/air_quality_data";
```

Рисунок 3.8 – Підключення до серверної частини

Ініціалізація сенсора BME680, створити об'єкт для роботи з сенсором (рисунок 3.9).

```
Adafruit_BME680 bme;
```

Рисунок 3.9 – Підключення сенсора BME680

Функція setup() робить:

– Ініціалізацію серійного зв'язку для відлагодження;

– Підключення до Wi-Fi мережі;

– Ініціалізацію сенсора BME680;

– Налаштування параметрів вимірювання сенсора;

Функція setup() виконується один раз при запуску пристрою (рисунок 3.10).

```
void setup() {
  Serial.begin(115200);

  // Підключення до Wi-Fi мережі
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Ініціалізація сенсора BME680
  if (!bme.begin()) {
    Serial.println("Could not find BME680 sensor!");
    while (1);
  }
  Serial.println("BME680 sensor found");

  // Установка інтервалу вимірювань
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
}
```

Рисунок 3.10 – Приклад роботи функції setup()

Функція loop() робить:

– Зчитування даних з сенсора;

– Створення JSON об'єкта для передачі даних;

– Перетворення JSON об'єкта у строку;

– Відправка HTTP POST запиту на сервер з JSON даними;

– Відображення результату запиту у серійному моніторі;

– Затримка на 1 хвилину перед наступним вимірюванням;

Функція loop() виконується безперервно (рисунок 3.11).

```
void loop() {
  // Вимірювання показників якості повітря
  float temperature = bme.temperature;
  float humidity = bme.humidity;
  float pressure = bme.pressure;
  float gasResistance = bme.gas_resistance / 1000.0; // Конвертування в кОм
  float airQualityIndex = bme.readGas();

  // Створення JSON об'єкта
  DynamicJsonDocument jsonDocument(200);
  jsonDocument["temperature"] = temperature;
  jsonDocument["humidity"] = humidity;
  jsonDocument["pressure"] = pressure;
  jsonDocument["gasResistance"] = gasResistance;
  jsonDocument["airQualityIndex"] = airQualityIndex;

  // Перетворення JSON об'єкта в рядок
  String jsonString;
  serializeJson(jsonDocument, jsonString);

  // Відправка POST запиту на сервер
  HTTPClient http;
  http.begin(serverAddress, serverPort, endpoint);
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST(jsonString);

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
  } else {
    Serial.print("Error sending POST request! HTTP Error code: ");
    Serial.println(httpResponseCode);
  }

  http.end();

  delay(60000); // Затримка в 1 хвилину перед наступним вимірюванням
}
```

Рисунок 3.11 – Приклад роботи функції loop()

Код надає повний цикл роботи з сенсором BME680, від підключення до Wi-Fi мережі до відправлення зібраних даних на сервер. Під час виконання програми, дані про якість повітря постійно збираються та передаються на сервер кожну хвилину, що дозволяє мати актуальну інформацію про стан повітря в режимі реального часу. Дані передаються на сервер у форматі JSON (рисунок 3.12).

JSON – це формат обміну даними, що використовується для зберігання та передачі структурованої інформації між клієнтом та сервером у веб-додатках. JSON - файл містить дані у вигляді пари "ключ-значення", що робить його зручним для читання і написання як людьми, так і машинами. Він базується на підмножині мови програмування JavaScript, але є незалежним від мови і може використовуватися в різних мовах програмування

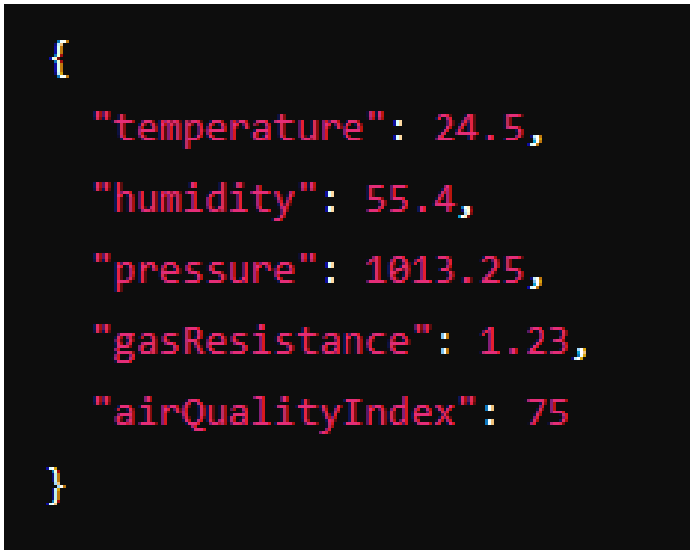


Рисунок 3.12 – Передані дані у форматі JSON

Файл містить ключі ("temperature" та "humidity") визначають тип даних, а значення (24.5 та 55.4) – самі дані. JSON дозволяє легко структурувати дані у вигляді об'єктів та масивів, що робить його ідеальним для передачі складної інформації в простому і зрозумілому форматі.

У цьому підрозділі ми розібрали роботу скетчу по частинам:

– Підключення необхідних бібліотек;

– Підключення до Wi-Fi мережі;

– Підключення до серверної частини;

– Ініціалізація датчика ;

– Функція setup();

– Функція loop();

Якщо ви бажаєте переглянути цей скетч повністю, ви зможете знайти його у додатку А.

4 РОЗРОБКА IOS ДОДАТКА

Ми будемо розбирати код iOS додатка поступово, за архітектурою MVVM (Model View ViewModel). Спочатку надамо код розділу Model, а потім детально опишемо, що в ньому відбувається.

4.1 Налаштування додатка (Model)

У цьому розділі коду ми визначаємо модельні класи та структури, необхідні для роботи додатку, а також логіку управління даними, а саме:

– Імпортування бібліотек;

– Перерахування станів геолокації;

– Огляд класу ViewModel;

– Метод для отримання даних з власного сервера;

– Розширення класу ViewModel;

Імпортовані бібліотеки надають функціональні можливості, необхідні для зміни географічного розташування, відстеження змін, створення користувацьких інтерфейсів та отримання даних про якість повітря (рисунок 4.1).

Рисунок 4.1 – Імпортовані бібліотеки для розділу Model

Перерахування LocationStatus визначає різні стани, які може мати додаток під час роботи з геолокацією та даними про якість повітря (рисунок 4.2).

```
enum LocationStatus: Equatable {
    case requestLocation
    case locationAutorized(String)
    case error(String)
    case requestAQIConditions
    case standby
}
```

Рисунок 4.2 – Перерахування LocationStatus

Клас ViewModel відповідає за управління даними та логікою додатку. Він включає такі елементи:

– locationManager: для управління геолокацією;

– aqiClient: для отримання даних про якість повітря з API;

– coordinatesFinder: для пошуку координат;

– currentLocation, locationStatus, position, annotations, selection, presentationD, lat, long: для зберігання поточного стану додатку та даних про місцезнаходження;

Ініціалізація:

– У конструкторі init ми ініціалізуємо масив radiusArray та налаштовуємо locationManager для отримання дозволу на використання геолокації;

Методи:

– handleCoordinateChange();

– Викликається при зміні координат, оновлює стан додатку та отримує дані про якість повітря для нових координат;

– getCoordinates();

– Генерує список координат для обраного радіусу навколо даної точки;

Цей клас є одним з ключових на даному етапі, так як завдяки ньому ми з’єднуємо додаток з нетворкінгом.

У подальшій роботі ми розробимо логіку цього класу для з’єднання з власним сервером

Наглядний приклад коду цього розділу (рисунок 4.3).

```
@Observable
class ViewModel: NSObject {
    let locationManager = CLLocationManager()
    let aqiClient = AirQualityClient(apiKey: "AIzaSyA610R1_yWfGJctGLEiZSZdCl43zDaWh6c")
    let coordinatesFinder = CoordinatesFinder()

    var currentLocation: CLLocationCoordinate2D?
    var locationStatus = LocationStatus.requestLocation
    var position: MapCameraPosition = .automatic
    var annotations: [AQIResponse] = []
    var selection: AQIResponse?
    var presentationD = PresentationDetent.height(176)
    var lat: Double = 0
    var long: Double = 0

    var radiusArray: [(Double, Int)]

    init(radiusArray: [(Double, Int)] = [(4000, 6), (8000, 12)]) {
        self.radiusArray = radiusArray
        super.init()
        locationManager.delegate = self
        locationManager.requestWhenInUseAuthorization()
    }
    @MainActor
    func handleCoordinateChange(_ coordinate: CLLocationCoordinate2D) async {
        do {
            self.locationStatus = .requestAQIConditions
            self.position = .region(.init(center: coordinate, latitudinalMeters: 0,
                longitudinalMeters: 16000))
            let coordinates = getCoordinates(coordinate)
            self.annotations = try await aqiClient.getCurrentConditions(coordinates:
                coordinates.map {($0.latitude, $0.longitude)})
            self.locationStatus = .standby
        } catch {
            self.locationStatus = .error(error.localizedDescription)
        }
    }
    func getCoordinates(_ coordinate: CLLocationCoordinate2D) -> [CLLocationCoordinate2D] {
        var results: [CLLocationCoordinate2D] = [coordinate]
        radiusArray.forEach {
            results += coordinatesFinder.findCoordinates(coordinate, r: $0.0, n: $0.1)
        }
        return results
    }
}
```

Рисунок 4.3 – Вміст класу ViewModel

Цей метод закоментований, оскільки, як ми розібрали раніше, ми будемо реалізовувати його у наступній роботі. Замість використання даних з власного сервера, у цьому проекті ми використовуємо дані Google API.

Використання цього методу дозволить отримувати дані про якість пові-

тря з власного сервера. У майбутньому, після налаштування сервера та інтеграції з нашим додатком, ми зможемо розкоментувати цей метод та використовувати його для отримання даних. Код для GET запиту з власного сервера (рисунок 4.4).

```
// Закоментований метод для отримання даних з власного сервера
    /*
    @MainActor
    func fetchAQIDataFromServer() async {
        do {
            let serverURL = URL(string: "http://cleanairapi.com/api/air_quality_data")!
            let (data, _) = try await URLSession.shared.data(from: serverURL)
            let decoder = JSONDecoder()
            let aqiResponses = try decoder.decode([AQIResponse].self, from: data)
            self.annotations = aqiResponses
            self.locationStatus = .standby
        } catch {
            self.locationStatus = .error(error.localizedDescription)
        }
    }
    */
```

Рисунок 4.4 – Вміст закоментованого методу для серверних даних

Відповідає за обробку подій, пов'язаних із визначенням місцезнаходження пристрою. Функція locationManagerDidChangeAuthorization() викликається, коли змінюється статус авторизації доступу до геолокації. Вона перевіряє поточний статус авторизації:

– Якщо статус authorizedWhenInUse (дозвіл на доступ до місцезнаходження під час використання програми), то manager.requestLocation() запитує поточне місцезнаходження пристрою;

– У випадку іншого статусу, змінюється значення locationStatus на .locationAutorized з повідомленням про те, що доступ до місцезнаходження не авторизований;

Функція locationManager() викликається, коли CLLocationManager не вдається отримати місцезнаходження через помилку. Вона змінює значення locationStatus на .error та передає текст помилки, що дозволяє додатку відповідно реагувати на помилку.

Функція locationManager() викликається, коли CLLocationManager успішно оновлює місцезнаходження пристрою. Вона:

– Отримує перше доступне місцезнаходження з масиву locations;

– Якщо currentLocation ще не встановлено (перевірка на nil), оновлює lat та long новими координатами та викликає асинхронний метод handleCoordinateChange для обробки зміни координат;

– Оновлює значення currentLocation новими координатами;

Розширення ViewModel забезпечує обробку різних сценаріїв, пов'язаних з геолокацією:

– Відстеження змін у статусі авторизації доступу до місцезнаходження;

– Обробка помилок, що виникають під час спроби отримати місцезнаходження;

– Оновлення координат пристрою та виклик відповідних методів для обробки цих змін;

Цей функціонал дозволяє додатку динамічно реагувати на зміни місцезнаходження користувача та забезпечує користувача актуальною інформацією про якість повітря у поточному місцезнаходженні (рисунок 4.5).

```
extension ViewModel: CLLocationManagerDelegate {
    func locationManagerDidChangeAuthorization(_ manager: CLLocationManager) {
        switch manager.authorizationStatus {
        case .authorizedWhenInUse:
            manager.requestLocation()
        default:
            self.locationStatus = .locationAutorized("Unauthorized location access")
        }
    }
    func locationManager(_ manager: CLLocationManager, didFailWithError error: any Error) {
        self.locationStatus = .error(error.localizedDescription)
    }
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
        [CLLocation]) {
        guard let coordinate = locations.first?.coordinate else { return }
        if currentLocation == nil {
            lat = coordinate.latitude
            long = coordinate.longitude
            Task { await self.handleCoordinateChange(coordinate)}
        }
        currentLocation = coordinate
    }
}
```

Рисунок 4.5 – Розширення для класу ViewModel

4.2 Функціонування додатка (ViewModel)

У цьому розділі коду визначається структура CoordinatesFinder, яка відповідає за обчислення координат точок на колі навколо заданої точки. Це потрібно для відображення на карті декількох точок, які представляють місця вимірювання якості повітря.

Тут використовується:

– Імпортування необхідних бібліотек;

– Огляд структури CoordinatesFinder;

– Функції конвертації;

– Функція findCoordinates;

Імпортовані бібліотеки надають необхідний функціонал для роботи з базовими типами даних та геолокацією, надаючи змогу робити математичні операції з радіанами (рисунок 4.6).

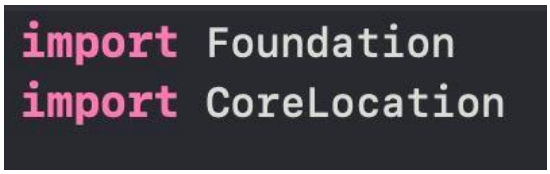


Рисунок 4.6 – Імпортування бібліотек для CoordinatesFinder

Константа радіусу Землі (R:), використовується для обчислення кутових відстаней. Константа пі (pi:), використовується для конвертації між градусами та радіанами (рисунок 4.7).

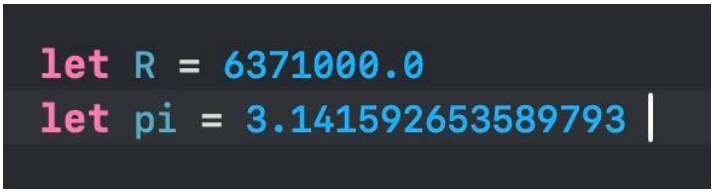


Рисунок 4.7 – Використання констант

Детальне роз’яснення роботи функцій. Метод deg2rad() – використовується для конвертації градуси у радіани. Метод rad2deg() – використовується для конвертації радіани у градуси (рисунок 4.8).

```swift
func deg2rad(_ deg: Double) -> Double {
    return deg * pi / 180
}

func rad2deg(_ rad: Double) -> Double {
    return rad * 180 / pi
}
```

Рисунок 4.8 – Конвертація

Функція findCoordinates використовується для наступних цілей:

– findCoordinates() – знаходить координати на колі навколо заданої точки;

– Конвертує початкові координати у радіани;

– Обчислює кутову відстань;

– Генерує координати для заданої кількості точок (n) на колі з радіусом (r);

– Конвертує результуючі координати у градуси та додає їх до масиву;

– Повертає масив координат;

Приклад розробка цієї функції буде виглядати наступним чином (рисунок 4.9).

```swift
// Define a function to find coordinates on a circle around a given point
func findCoordinates(_ coordinate: CLLocationCoordinate2D, r: Double, n: Int) ->
    [CLLocationCoordinate2D] {
    // Convert inputs to radians
    let phi1 = deg2rad(coordinate.latitude)
    let lambda1 = deg2rad(coordinate.longitude)
    let d = r / R // angular distance

    // Initialize an empty array to store the coordinates
    var coordinates: [CLLocationCoordinate2D] = []

    // Loop over different bearings
    for i in 0..<n {
        // Convert bearing to radians
        let theta = deg2rad(Double(i) * 360 / Double(n))

        // Apply the formula for destination point
        let phi2 = asin(sin(phi1) *  cos(d) + cos(phi1) * sin(d) * cos(theta))
        let lambda2 = lambda1 + atan2(sin(theta) * sin(d) * cos(phi1), cos(d) - sin(phi1) *
            sin(phi2))

        // Convert outputs to degrees
        let lat2 = rad2deg(phi2)
        let lon2 = rad2deg(lambda2)

        // Append the coordinate to the array
        coordinates.append(.init(latitude: lat2, longitude: lon2))
    }


        // Return the array of coordinates
        return coordinates
    }
}
```

Рисунок 4.9 – Використання функції findCoordinates

CircleViewAqi описує компонент, який використовується для відображення інформації про якість повітря (AQI) у вигляді круга з текстом. Цей компонент є частиною представлення користувацького інтерфейсу (View) у додатку, реалізованому за допомогою SwiftUI.

SwiftUI забезпечує інструменти для створення користувацького інтерфейсу, а XCAAQI містить структури та дані для роботи з API якості повітря

Приклад бібліотек для реалізації кольорових кружечків (рисунок 4.10) .

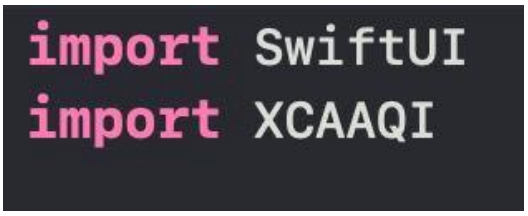


Рисунок 4.10 – Імпортування необхідних бібліотек

Константа aqi – це об'єкт типу AQIResponse, який містить дані про якість повітря, включаючи індекс якості повітря та кольорову індикацію. Зміна isSelected - булеве значення, яке визначає, чи обрана аннотація. Використовується для зміни вигляду компонента, коли він обраний. Зміна size - розмір круга, який відображає індекс якості повітря (рисунок 4.11).

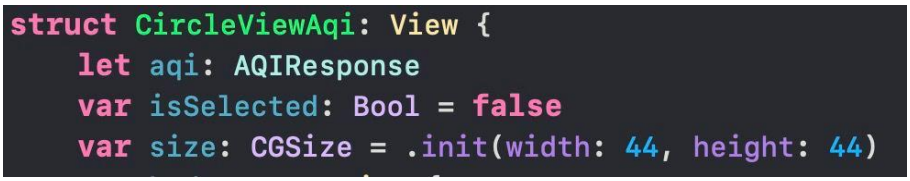


Рисунок 4.11 – Структура CircleViewAqi

В основній частині використовуємо наступні методи:

– Circle() – створює круг як основну форму компонента;

– .stroke – задає колір обведення круга. Колір визначається значеннями червоного, зеленого та синього компонентів, що беруться з даних aqi.color. Товщина обведення змінюється в залежності від того, чи обраний компонент;

– .frame – встановлює розмір круга;

– .overlay – додає текстове відображення індексу якості повітря (aqi.aqiDisplay) поверх круга. Текст має білий колір та жирний шрифт;

– .scaleEffect – змінює масштаб компонента в залежності від того, чи обраний компонент. Якщо обраний, розмір збільшується на 50%;

Приклад роботи коду основної частини (рисунок 4.12).

```
    var body: some View {
        Circle()
            .stroke(Color(red: aqi.color.red, green: aqi.color.green,
                blue: aqi.color.blue), lineWidth: isSelected ? 4 : 3)
            .frame(width: size.width, height: size.height)
            .overlay {
                Text(aqi.aqiDisplay)
                    .foregroundStyle(.white)
                    .fontWeight(.bold)
            }
            .scaleEffect(isSelected ? CGSize(width: 1.5, height: 1.5) :
                CGSize(width: 1, height: 1))
    }
}
```

Рисунок 4.12 – Код основної частини CircleViewAqi

Блок коду забезпечує прев'ю компонента CircleViewAqi у Xcode, що дозволяє швидко переглянути, як виглядатиме компонент з конкретними даними. Компонент CircleViewAqi створює кругову аннотацію для відображення індексу якості повітря (AQI) з використанням кольору, що індиксує якість повітря. Основні функції компонента включають:

– Відображення круга з кольоровою обведенням, що вказує на рівень якості повітря;

– Відображення індексу AQI в середині круга;

– Зміна вигляду компонента, коли він обраний (товстіша обведення та збільшений розмір);

Цей компонент є частиною архітектури MVVM і використовується в View для представлення даних якості повітря на карті (рисунок 4.13).

```
#Preview {
    CircleViewAqi(aqi: .init(aqiDisplay: "23", color: .init(red: 0.2,
        green: 0.5, blue: 0.5)), isSelected: true)
}
```

Рисунок 4.13 – Перегляд CircleViewAqi

4.3 Опис користувацького інтерфейсу (View)

У цьому розділі коду описано представлення користувацького інтерфейсу (View) у додатку. Ми використовуємо архітектуру MVVM (Model-View-ViewModel) для організації коду, що забезпечує чіткий розподіл відповідальності між моделлю, представленням та логікою бізнесу.

Цей розділ включає в себе наступні компоненти:

– Імпортовані бібліотеки;

– Оголошення структури ContentView;

– Основна частина користувацького інтерфейсу;

– Вибір анотації;

– Користувацька діяльність;

– Перегляд користувацького вмісту;

Імпортовані бібліотеки дозволяють працювати з картами (MapKit), створювати інтерфейс користувача (SwiftUI) та взаємодіяти з API для якості повітря XCAAQI (рисунок 4.14).

```
import MapKit
import SwiftUI
import XCAAQI
```

Рисунок 4.14 – Бібліотеки для розділу View

Ця структура є основним представленням користувацького інтерфейсу. Тут також створюється стан (State) для ViewModel, який буде відповідати за логіку бізнесу та дані (рисунок 4.15).

```
struct ContentView: View {
    @State var vm = ViewModel()
```

Рисунок 4.15 – Структура ContentView

Карта (Map):

– Використовується компонент Map для відображення карти;

– Карта отримує позицію та вибір з ViewModel через двостороннє зв'язування ($vm.position та $vm.selection);

– ForEach проходиться по масиву annotations з ViewModel та створює аннотації на карті;

– Кожна аннотація відображає дані якості повітря за допомогою CircleViewAqi;

Стиль карти:

– Встановлюється гібридний стиль карти (.hybrid), який включає в себе елементи зображень з супутника та традиційної карти;

Код, що відповідає цьому переліку (рисунок 4.16).

```
var body: some View {

    Map(position: $vm.position, selection: $vm.selection) {
        ForEach(vm.annotations) { aqi in
            Annotation(aqi.aqiDisplay, coordinate: aqi.coordinate) {
                CircleViewAqi(aqi: aqi, isSelected: aqi ==
                    vm.selection)
            }
            .tag(aqi)
            .annotationTitles(.hidden)
        }
    }
    .mapStyle(.hybrid(elevation: .flat, pointsOfInterest: .all,
        showsTraffic: false))
```

Рисунок 4.16 – Використання властивостей карти

Sheet (додаткове вікно):

Використовується sheet для відображення додаткової інформації в окремому вікні. Приклад коду відображення інформації у окремому вікні (рисунок 4.17).

```
    .sheet(isPresented: .constant(true)) {
        ScrollView {
            VStack{
                if let selection = vm.selection {
                    selectedAQIView(aqi: selection)
                } else {
                    if vm.locationStatus != .requestLocation &&
                        vm.locationStatus != .requestAQIConditions {
                        locationFromView
                    }
```

Рисунок 4.17 – Код інформаційного вікна

Якщо є вибрана аннотація (vm.selection), то відображається детальна інформація про неї за допомогою selectedAQIView, якщо аннотація не вибрана, то відображаються різні статуси запиту місця розташування та умов якості повітря з використанням ProgressView та текстових повідомлень.

Це реалізується завдяки if, else та else if надаючи варіативність розвитку подій.

Також тут відбуваються різні різні методи для покращення та спрощення інтерфейсу, такі як заголовок додатка та заднього фону (рисунок 4.18).

```
                if vm.locationStatus == .requestAQIConditions {
                    ProgressView("Requesting AQI conditions ...
                        just wait ...")
                }
                if vm.locationStatus == .requestLocation {
                    ProgressView("Requesting current locations ...
                        just wait ...")
                }
                if case let .locationAutorized(text) =
                    vm.locationStatus {
                    Text(text)
                }
                if case let .error(text) = vm.locationStatus {
                    Text(text)
                }
            }
        }
    }
        .padding()
        .safeAreaPadding(.top)
        .presentationDetents([.height(24), .height(176)], selection:
            $vm.presentationD)
        .presentationBackground(.ultraThinMaterial)
        .presentationBackgroundInteraction(.enabled(upThrough:
            .height(176)))
        .interactiveDismissDisabled()
    }
    .onChange(of: vm.selection) { oldValue, newValue in
        if oldValue == nil && newValue != nil {
            vm.presentationD = .height(176)
        }
    }
    .navigationTitle("Air Quality")
    .navigationBarTitleDisplayMode(.inline)
    .toolbarBackground(.ultraThinMaterial, for: .navigationBar)
}
```

Рисунок 4.18 – Використання логіки if, else

Функція для відображення вибраної аннотації, відображає детальну інформацію про вибрану аннотацію якості повітря.

Надає саме значення якості та головний забруднюючий елемент та змогу передивлятись ці показники у різних точках світу завдяки координатам ширини та довготи.

Всі ці дані у окремому розробленому вікні для покращення загальної візуалізації додатка (рисунок 4.19).

```
func selectedAQIView(aqi: AQIResponse) -> some View {
    HStack(spacing: 16) {
        CircleViewAqi(aqi: aqi, size: CGSize(width: 80, height: 80))
        VStack(alignment: .leading)  {
            Text("Coordinate: \(aqi.coordinate.latitude),
                \(aqi.coordinate.longitude)")
            Text(aqi.category)
            Text("Dominant Pollutant: \(aqi.dominantPollutant)")
            Text(aqi.displayName)
        }
    }
    .padding(.top)
    .padding(.horizontal)
    .frame(maxWidth: .infinity)
}
```

Рисунок 4.19 – Функція вибору анотації

Блок коду, що відповідає за форму введення координат для отримання даних якості повітря:

– Користувач може ввести широту та довготу вручну;

– Можна також використати поточне місцезнаходження за допомогою кнопки "Use current location";

– Кнопка "Refresh AQI" оновлює дані якості повітря для введених координат;

Дані широти та довготи вводяться вручну в спеціальному для цього вікні, яке виповнене у формі прямокутника.

Самі ж кнопки мають також дизайн прямокутника зеленій кольоровій палітрі, яка чудово доповнює в цілому стиль карти.

Загалом весь дизайн виповнен в одному єдиному стилі, що дає привабливий вигляд та не порте логістику і ідею застосунку.

Використовуємо метод await, щоб ці значення обновлялись в той момент, коли це дійсно потрібно.

Таке рішення позитивно вплине на оптимізацію додатка, що наддасть змогу запускати його на більш старих девайсах, таких як:

– IPhone 7;
– IPhone XS;
– IPhone 11;
– IPhone 12;
– IPhone 13;

Код для реалізації цієї функції (рисунок 4.20).

```
@ViewBuilder
var locationFromView: some View {
    Text("Get AQI current (around a coordinate)")
        .font(.headline)
        .padding(.bottom, 8)

    HStack {
        Text("Latit")
        TextField("Enter latitude", value: $vm.lat, format: .number)
        Text("Long")
        TextField("Enter longitude", value: $vm.long, format: .number)
    }
    .keyboardType(.decimalPad)
    .textFieldStyle(.roundedBorder)
    .padding(.bottom, 8)

    HStack{
        Button("Use current location") {
            vm.lat = vm.currentLocation?.latitude ?? 0
            vm.long = vm.currentLocation?.longitude ?? 0
            Task {
                await vm.handleCoordinateChange(.init(latitude:
                    vm.lat, longitude: vm.long))

            }
        }
         .frame(width: 170, height: 35)
             .background(.green)
             .foregroundColor(.black)
             .clipShape(Rectangle())

         Button("Refresh AQI") {
             Task {
                 await vm.handleCoordinateChange(.init(latitude:
                     vm.lat, longitude: vm.long))
             }
         }
         .frame(width: 120, height: 35)
             .background(.green)
             .foregroundColor(.black)
             .clipShape(Rectangle())
     }
```

Рисунок 4.20 – Код для діяльності користувача

Блок коду дозволяє переглянути ContentView у NavigationStack під час розробки, що забезпечує зручний перегляд та тестування інтерфейсу (рисунок 4.21).

```
#Preview {
    NavigationStack {
        ContentView(vm: .init(radiusArray: []))
    }
}
```

Рисунок 4.21 – Користувацький вміст

Розробка мобільного додатка для моніторингу якості повітря з використанням архітектури MVVM (Model View View - Model) забезпечила чітке розділення відповідальності між різними компонентами додатка, що сприяє кращій організації коду, підвищенню його читабельності та підтримуваності.

Використання архітектури MVVM для розробки мобільного додатку з моніторингу якості повітря виявилося ефективним підходом, який забезпечує не лише технічні переваги, але й сприяє підвищенню загальної якості програмного забезпечення. Даний підхід дозволяє створювати модульні, тестовані та легко підтримувані додатки, що є важливим фактором для успішної розробки та подальшого розвитку програмних рішень.

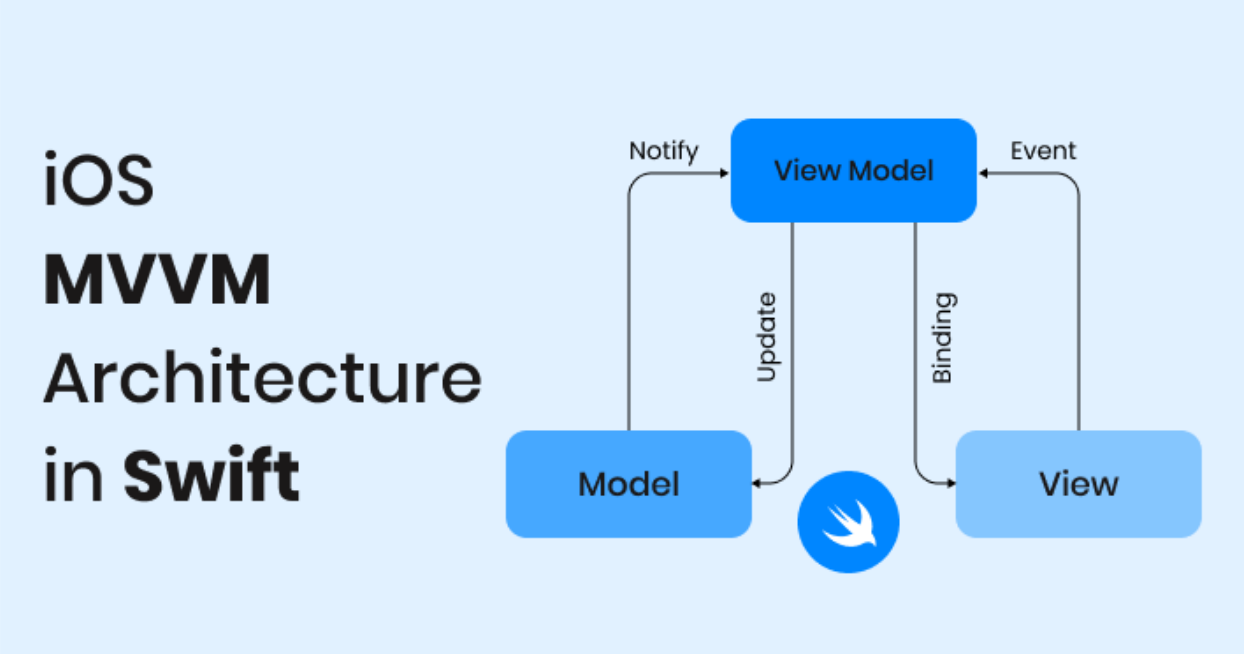


Рисунок 4.22 – Наглядний приклад архітектури MVVM

У цьому підрозділі ми розібрали роботу коду по частинам (налаштування додатку за допомогою Model, функціонування за допомогою View Model, перегляд додатку View та додатковий розділ CircleViewAqi), якщо ви бажаєте переглянути цей код повністю, ви зможете знайти його у додатку Б.

ВИСНОВКИ

У цій дипломній роботі ми розглянули розробку комплексної системи для моніторингу якості повітря, яка включає апаратну платформу, серверну частину та мобільний додаток для iOS. Система дозволяє користувачам отримувати актуальні дані про якість повітря в режимі реального часу, використовуючи архітектуру MVVM (Model View ViewModel) для забезпечення чіткого розділення логіки, даних та представлення.

На початку ми реалізували апаратну частину системи, яка включає датчик якості повітря, підключений до плати Arduino ESP32. Датчик вимірює індекс якості повітря (AQI), а плата збирає ці дані та передає їх на сервер через Wi-Fi. Сервер обробляє отримані дані, зберігає їх у базі даних та надає API для доступу до цих даних.

Мобільний додаток для iOS був розроблений з використанням SwiftUI та архітектури MVVM. Це забезпечило зручну структуру для розробки та підтримки додатку. Модель (Model) відповідала за структуру даних і взаємодію з API, тоді як ViewModel обробляв дані та взаємодіяв з Model для отримання інформації про якість повітря, використовуючи CLLocationManager для отримання поточної геолокації користувача та запитів до API. Представлення (View) забезпечувало відображення інтерфейсу користувача, включаючи інтерактивну карту з аннотаціями, що показують рівень якості повітря в різних локаціях.

Для відображення даних ми використовували MapKit, що дозволило створити інтерактивну карту з круговими аннотаціями, які вказують на рівень якості повітря. Кожна аннотація містить інформацію про індекс AQI, відображену у вигляді кольорової обведення та числового значення всередині. Користувачі можуть бачити детальну інформацію про якість повітря, включаючи координати, категорію забруднення та домінуючий забруднювач, для обраної аннотації.

Загалом, ця система демонструє ефективний підхід до моніторингу та

візуалізації якості повітря, використовуючи сучасні технології. Реалізація охоплює всі етапи від збору даних до їх відображення на мобільному пристрої, забезпечуючи користувачів актуальною інформацією про стан повітря у їхньому регіоні. У майбутньому можлива інтеграція з власним сервером для покращення точності даних та розширення функціональності додатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Smith, J.; Brown, L. Analyzing Air Quality Data in Urban Environments Using IoT Sensors. IEEE Access, 2021, 9, 87456-87469. [CrossRef]

2. Johnson, P.; Wang, T.; Lee, S. A Fuzzy Logic Approach to Air Quality Monitoring and Control in Smart Cities. Appl. Soft Comput., 2022, 107, 107581. [CrossRef]

3. Patel, M.; Chen, H. Hybrid Machine Learning Models for Predicting Air Quality Index in Real-Time. J. Environ. Manage., 2023, 255, 109846. [CrossRef]

4. Kumar, R.; Sharma, V.; Gupta, A. Optimizing Sensor Deployment for Large-Scale Air Quality Monitoring Networks Using Swarm Intelligence. AIP Conf. Proc., 2021, 2456, 130021.

5. Singh, K.; Fong, S.; Park, J.; Vasilakos, A.V. Adaptive Data Processing Techniques for Real-Time Air Quality Monitoring Using IoT Sensors. Symmetry, 2020, 12, 310. [CrossRef]

6. Ali, S.; Yuan, D.; Jin, J.; Gao, L.; Yu, S.; Dong, Z.Y. A Cybersecurity Framework for IoT-Based Air Quality Monitoring Systems. Future Gener. Comput. Syst., 2021, 114, 140-150. [CrossRef]

7. Khanna, V.; Patel, P.; Baredar, P. Optimization of Hybrid Air Quality Monitoring Systems Using Machine Learning and IoT. Int. J. Sustain. Environ., 2020, 45, 789-802. [CrossRef]

8. Rasheed, M.; Omar, R.; Sulaiman, M.; Halim, W.A. Air Quality Prediction Using Particle Swarm Optimization in Sensor Networks. Indones. J. Electr. Eng. Comput. Sci., 2022, 18, 223-234. [CrossRef]

9. Hasan, M.Z.; Al-Rizzo, H.; Al-Turjman, F.; Rodriguez, J.; Radwan, A. IoT-Based Task Scheduling for Efficient Air Quality Monitoring in Cloud Environments. J. Cloud Comput., 2021, 9, 45-58. [CrossRef]