

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

**Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії**

«На правах рукопису»
УДК 004.42

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРИКОВ
«__» _____ 2024 р.

Магістерська дисертація

на здобуття ступеня магістра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення інформаційних систем»**

зі спеціальності 121 «Інженерія програмного забезпечення»

**на тему: «Інструмент для реактивної роботи з мережею на
платформах iOS та macOS»**

Виконав:

студент II курсу, групи ІІ-22мп
Жердій Павло Олександрович _____

Керівник:

доцент кафедри ІІІ, к.т.н., доц.,
Ліхоузова Тетяна Анатоліївна _____

Рецензент:

доцент кафедри ІСТ, к.т.н., доц.,
Солдатова Марія Олександрівна _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___» _____ 2023р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Жердію Павлу Олександровичу

1. Тема дисертації «Інструмент для реактивної роботи з мережею на платформах iOS та macOS», науковий керівник дисертації Ліхоузова Тетяна Анатоліївна, доцент кафедри ІІІ, к.т.н. затверджені наказом по університету від «11» листопада 2023 р. № 5168-с
2. Термін подання студентом дисертації «16» січня 2024 р.
3. Об'єкт дослідження – мультиплатформений інструмент для роботи з мережею
4. Предмет дослідження – методи та підходи для реактивної роботи застосунку на платформах iOS та MacOS
5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; побудова структури інструменту; розробка програмного забезпечення; розгортання бібліотеки
6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 4 плакати

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

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

9. Дата видачі завдання «1» листопада 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Огляд літератури	1.11.2022	
2	Порівняльний аналіз існуючих рішень	10.05.2023	
3	Побудова структури інструменту	14.07.2023	
4	Розробка програмного забезпечення	8.09.2023	
5	Розгортання бібліотеки	12.11.2023	
6	Розробка стартап-проєкту	16.12.2023	
7	Оформлення пояснювальної записки	17.12.2023	
8	Подання дисертації на попередній захист	19.12.2023	
9	Подання дисертації на захист	16.01.2024	

Студент

Павло ЖЕРДІЙ

Науковий керівник

Тетяна ЛІХОУЗОВА

РЕФЕРАТ

Розмір пояснювальної записки – 107 аркушів, містить 28 ілюстрацій, 23 таблиці, 4 додатки, 22 посилання на джерела.

Актуальність теми. У роботі розглянуто проблему в області реактивної роботи з мережею, показано основні особливості існуючих рішень для платформ MacOS та iOS, їх переваги та недоліки. Виявлено потребу в забезпеченні адаптивності та ефективності роботи застосунків під час взаємодії з мережею.

Мета дослідження. Основною метою є підвищення зручності роботи з мережевими запитами, асинхронними операціями та управлінням потоками даних.

Об’єкт дослідження: мультиплатформений інструмент для роботи з мережею

Предмет дослідження: методи та підходи для реактивної роботи застосунку на платформах iOS та MacOS

Для реалізації поставленої мети **сформульовані наступні завдання:**

- аналіз існуючих рішень;
- побудова структури інструменту;
- розробка програмного забезпечення;
- розгортання бібліотеки.

Наукова новизна: набув подальшого розвитку реактивний підхід для забезпечення адаптивності та ефективності роботи застосунків під час взаємодії з мережею.

Практичне значення отриманих результатів полягає в тому, що розробникам мобільних та десктопних застосунків на платформах iOS та MacOS надано зручний інструмент для роботи з мережевими запитами, асинхронними операціями та управлінням потоками даних.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на V науково-практичній конференції «SoftTech-2023» та опубліковані у матеріалах конференції.

Ключові слова: SWIFT, РОБОТА З МЕРЕЖЕЮ, MACOS, IOS, РЕАКТИВНЕ ПРОГРАМУВАННЯ.

ABSTRACT

Explanatory note size – 107 pages, contains 28 illustrations, 23 tables, 4 applications, 22 references.

Topicality. Examines the problem in the field of reactive networking, shows the main features of existing solutions for MacOS and iOS platforms, their advantages and disadvantages. The need to improve the methods of working with the network and eliminate existing shortcomings is identified.

The aim of the study. The main target is to provide developers with a convenient tool for working with network requests, asynchronous operations, and data flow management.

The object of research: a multi-platform tool for working with the network.

The subject of research: methods and approaches for reactive application operation on iOS and MacOS platforms.

To achieve this goal, the **following tasks** were formulated:

- analysis of existing solutions;
- building the structure of the tool;
- software development;
- deployment of the library.

The scientific novelty of the results of the master's dissertation is that the reactive approach to ensuring the adaptability and efficiency of applications when interacting with the network has been further developed.

The practical value of the obtained results is that developers of mobile and desktop applications on iOS and MacOS platforms are provided with a convenient tool for working with network requests, asynchronous operations, and data flow management.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering

of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Approbation. The scientific provisions of the dissertation were tested at the fifth Scientific and Practical Conference "SoftTech-2023" and published in the conference proceedings.

Keywords: SWIFT, NETWORKING, MACOS, IOS, REACTIVE PROGRAMMING.

ЗМІСТ

ВСТУП	11
1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	12
1.1 Змістовний опис і аналіз предметної області	12
1.2 Опис функціональної моделі.....	14
1.3 Огляд наявних аналогів	15
1.4 Аналіз вимог до програмного забезпечення.....	17
Висновки до розділу	20
2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	22
2.1 Моделювання та аналіз програмного забезпечення	22
2.2 Концепція розробки мобільного застосунку під iOS.....	25
2.3 Концепція розробки десктопного застосунку під MacOS.....	28
2.4 Порівняння рішень	32
2.5 Рішення для реактивної роботи з мережею	34
Висновки до розділу	41
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	42
3.1 Вибір інструментів для розробки	42
3.2 Розробка інструменту для роботи з VPN.....	44
3.3 Розробка інструменту для роботи з картами	48
3.4 Структура інструменту	51
Висновки до розділу	56
4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ІНСТРУМЕНТУ	58
4.1 Створення та налаштування бібліотеки CocoaPods	58

4.2 Налаштування роботи бібліотеки з Swift Package Manager	61
4.3 Налаштування роботи бібліотеки з Carthage	62
Висновки до розділу	63
5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ	64
5.1 Опис ідеї проєкту.....	64
5.2 Технологічний аудит ідеї проєкту	66
5.3 Аналіз ринкових можливостей запуску стартап-проєкту.....	66
5.4 Розроблення ринкової стратегії проєкту	74
5.5 Розроблення маркетингової програми стартап-проєкту.....	77
Висновки до розділу	80
ВИСНОВКИ.....	81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	83
ДОДАТОК А.....	85
ДОДАТОК Б	86
ДОДАТОК В	87
ДОДАТОК Г	107

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – програмний інтерфейс

ООП – об’єктно орієнтоване програмування

HTTP – протокол передачі гіпертекстових документів

ПЗ – Програмне забезпечення.

MVC — model view controller. Модель, вигляд, контролер.

MVVM — model view viewmodel. Модель, вигляд, вьюмодель.

MVP — minimal valuable product. Продукт з мінімальною кількістю функціоналу, яким можна користуватись.

IDE — integrated development environment. Комплексне програмне рішення для розробки програмного забезпечення.

JSON – запис об’єктів JavaScript

XML – розширена мова розмітки

SPM – менеджер пакетів Swift Package Manager

ВСТУП

У сучасному світі мобільні та десктопні застосунки відіграють важливу роль у повсякденному житті людей, надаючи їм широкий спектр можливостей і зручність використання. Однак, зі зростанням складності та функціональності застосунків виникають нові виклики для розробників. Один із таких викликів полягає в забезпеченні адаптивності та ефективності роботи застосунків під час взаємодії з мережею.

В рамках даної магістерської розглянуто реактивний підхід у використанні мережі при розробці мобільних та десктопних застосунків. Реактивне програмування - це парадигма розроблення програмного забезпечення, заснована на асинхронній та чуйній взаємодії компонентів системи. Вона прагне забезпечити легкість в управлінні змінами та ефективне використання ресурсів.

Застосування реактивного підходу під час розроблення мобільних застосунків має низку переваг. По-перше, він дає змогу створювати адаптивні застосунки, які можуть обробляти асинхронні події, такі як запити до мережі, без блокування користувацького інтерфейсу. По-друге, реактивне програмування спрощує управління станом програми та реагування на зміни даних. Крім того, реактивні бібліотеки та фреймворки надають зручні інструменти для опрацювання потоків даних і управління помилками.

У роботі застосовано реактивний підхід до використання мережі під час розроблення мобільних застосунків для створення більш ефективних і адаптивних застосунків.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Змістовний опис і аналіз предметної області

У цьому розділі буде описана предметна область, пов'язана з використанням реактивного підходу в розробці мобільних застосунків. Розглянемо основні аспекти, виклики та вимоги, з якими стикаються розробники в цій галузі.

Мобільні застосунки стали невід'ємною частиною повсякденного життя людей. Вони надають можливості взаємодії, розваги, управління завданнями та інші функціональності. Розглянемо різноманітні типи мобільних застосунків, їхні цілі та основні характеристики.

Основною метою є надання розробникам зручного інструменту для роботи з мережевими запитами, асинхронними операціями та управлінням потоками даних.

Задачею є створити інструмент для розробки застосунків на платформах iOS та MacOS, який буде оснований на реактивному підході роботи з мережею та допоможе користувачам розробляти зручні мобільні та десктопні застосунки. Інструмент має забезпечити чуйність і гнучкість взаємодії з мережею, даючи змогу легко керувати потоками даних, реагувати на зміни та обробляти помилки.

Однією з ключових функціональностей мобільних застосунків є взаємодія з мережею. Розглядаємо типові сценарії використання мережі, такі як завантаження даних, надсилання запитів на сервер, обмін даними в режимі реального часу тощо. Аналізуємо виклики та проблеми, пов'язані з цими сценаріями.

Введемо основні концепції реактивного програмування, які застосовуються під час використання реактивного підходу в розробці мобільних застосунків. Обговоримо реактивні потоки даних, асинхронність, управління станом і реагування на зміни. Опишемо

переваги, які реактивний підхід може принести при роботі з мережею в мобільних застосунках.

Досліджуємо різні реактивні бібліотеки та фреймворки, які можуть бути використані під час розроблення мобільних застосунків. Розглядаємо популярні інструменти, як-от RxJava, RxSwift або Kotlin Coroutines, та аналізуємо їхні особливості, можливості й обмеження. Наводимо приклади використання реактивного підходу при взаємодії з мережею в мобільних застосунках.

Обговоримо основні виклики, з якими стикаються розробники під час застосування реактивного підходу у використанні мережі в мобільних застосунках. Розглянемо проблеми, такі як управління потоками даних, обробка помилок, управління ресурсами та синхронізація даних. Запропонуємо рішення та найкращі практики для подолання цих викликів, використовуючи реактивний підхід.

Проводимо огляд переваг, які реактивний підхід може принести у використанні мережі під час розроблення мобільних застосунків. Описуємо простоту опрацювання асинхронних подій, управління станом та реагування на зміни. Однак, також обговорюємо деякі недоліки, включно з вивченням і застосуванням нових концепцій, складністю налагодження та можливим збільшенням складності кодової бази.

У прикінцевій частині розділу опис і аналіз предметної області підбиваємо підсумки й основні висновки. Наголошуємо на важливості використання реактивного підходу в розробці мобільних застосунків для ефективної взаємодії з мережею. Зазначаємо, що вибір конкретної реактивної бібліотеки або фреймворка залежить від вимог проєкту та вподобань розробника.

У результаті проведеного аналізу предметної області можна зробити висновок про значущість і застосування.

1.2 Опис функціональної моделі

Інструмент, що розробляється для платформ iOS і macOS, являє собою розширення (extension), яке призначене для полегшення роботи з мережею з використанням реактивного підходу за допомогою бібліотеки Rx.

Одним із ключових функціональних елементів застосунку є інтеграція з бібліотекою Rx, яка надає реактивні потоки даних, оператори для перетворення і фільтрації даних, а також зручні засоби управління підписками та обробки помилок. Застосунок надає користувальницький інтерфейс, що дає змогу розробникам створювати та налаштовувати реактивні запити, керувати потоками даних і спостерігати за результатами.[1]

Призначений для користувача інтерфейс застосунку розроблено з урахуванням особливостей платформ iOS і macOS, щоб забезпечити однаковий та інтуїтивно зрозумілий досвід використання для користувачів обох платформ. Застосунок надає зручні інструменти для створення, редагування та збереження реактивних запитів, а також можливість перегляду результатів запитів у зручному форматі.

Крім того, застосунок забезпечує підтримку різних методів запитів (GET, POST, PUT, DELETE тощо), можливість встановлення заголовків і параметрів запиту, а також можливість роботи з різними форматами даних (наприклад, JSON або XML).

Отже, розроблюване розширення являє собою зручний інструмент для роботи з мережею на платформах iOS і macOS з використанням реактивного підходу за допомогою бібліотеки Rx. Воно дає змогу розробникам легко створювати та керувати реактивними запитами, обробляти асинхронні операції та ефективно працювати з даними, забезпечуючи чуйність і гнучкість взаємодії з мережею[2].

1.3 Огляд наявних аналогів

На ринку існує кілька аналогічних інструментів, які пропонують можливість роботи з мережею з використанням реактивного підходу під час розроблення мобільних застосунків на платформах iOS і macOS. Ось кілька прикладів таких інструментів:

Alamofire: Це одна з найпопулярніших бібліотек для роботи з мережею на платформі iOS і macOS. Alamofire надає зручні засоби для надсилання мережевих запитів, опрацювання відповідей і керування потоками даних[5]. Хоча Alamofire не є суворо реактивною бібліотекою, його можна використовувати в поєднанні з реактивними патернами за допомогою інших інструментів, таких як RxSwift.



Рисунок 1.1 – Фреймворк Alamofire

Moxy: Це ще одна популярна бібліотека для роботи з мережею на платформі iOS і macOS. Moxy пропонує абстракцію над мережевими запитами, дозволяючи визначити API-інтерфейси у вигляді окремих сервісів. Moxy добре інтегрується з реактивними бібліотеками, такими як RxSwift або ReactiveCocoa, і надає зручні можливості для обробки мережевих запитів у реактивному стилі.



Рисунок 1.2 – Фреймворк Moya

Apollo iOS: Це бібліотека для роботи з GraphQL на платформі iOS. Apollo iOS дає змогу визначити схему GraphQL і автоматично генерувати типобезпечні клієнтські запити. Бібліотека інтегрується з реактивними бібліотеками, такими як RxSwift або Combine, і надає зручні можливості для роботи з GraphQL-запитами в реактивному стилі.



Рисунок 1.3 – Фреймворк Apollo

CombineNetworking: Це бібліотека, заснована на фреймворку Combine, що надає засоби для роботи з мережею в реактивному стилі на платформі iOS і macOS. CombineNetworking дає змогу створювати та виконувати мережеві запити, обробляти відповіді та керувати потоками даних за допомогою операторів Combine.

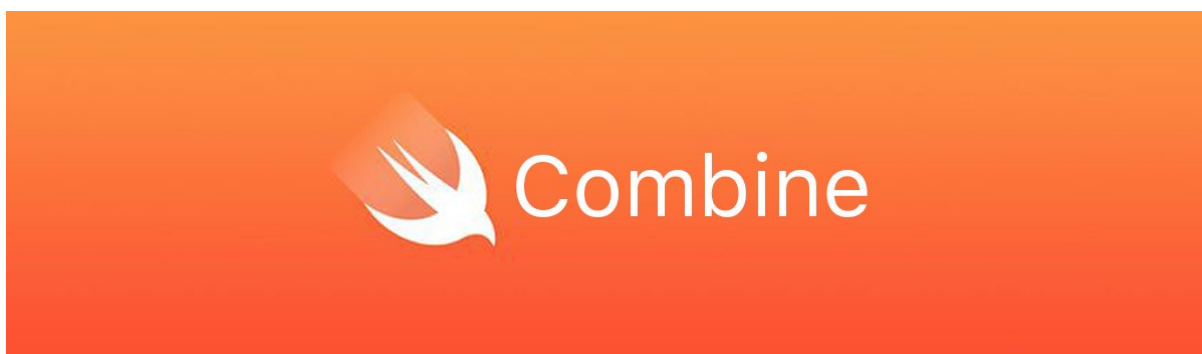


Рисунок 1.4 – Фреймворк Combine

Розроблений інструмент перевершує наявні аналоги, надаючи реактивне рішення для роботи з мережею і підтримку реактивності в розширеннях, як-от Packet Tunnel Provider, App Proxy Provider, Filter Data Provider і Filter Control Provider. Це дає змогу розробникам створювати гнучкіші, ефективніші та чуйніші застосунки, забезпечуючи високу продуктивність і зручність використання.

	Alamofire	Apollo	Моя	Запропоноване Рішення
Підтримка середовищ розробки	UIKit, SwiftUI, AppKit	UIKit, SwiftUI	UIKit	AppKit, UIKit, SwiftUI
Тип API	REST	GraphQL	REST, GraphQL	REST, Websocket
Робота з реактивними потоками	RxAlamofire надає реактивний інтерфейс для роботи з мережею на основі RxSwift	Не має реактивного інтерфейсу, працює з підписками для GraphQL за допомогою RxSwift	Не має реактивного інтерфейсу, працює за допомогою RxSwift	Має зручний вбудований інтерфейс для реактивної роботи з мережею на основі Combine
Підтримка Framework	URLSession	URLSession	URLSession	URLSession, MapKit, WebSocket, Network Extension, NEVPNManager, Packet Tunnel Provider

Рисунок 1.5 – Таблиця порівняння існуючих рішень з запропонованим інструментом

1.4 Аналіз вимог до програмного забезпечення

Інструмент повинен мати такий функціонал:

- мультиплатформеність: Однією з основних вимог є підтримка платформ iOS і macOS. Застосунок має бути розроблений таким чином, щоб він міг бути встановлений і працювати на обох платформах без необхідності переписування коду. Це дає змогу максимально охопити аудиторію користувачів і забезпечити єдиний досвід використання на різних пристроях;
- розширення (extension): Застосунок має бути розроблений у вигляді розширення, щоб він міг інтегруватися з іншими застосунками та надавати функціональність роботи з мережею. Розширення повинно мати доступ до мережі та мати можливість взаємодії з користувацькими запитам та даними;
- робота з мережею: Однією з ключових вимог є забезпечення можливості роботи з мережею. Застосунок має підтримувати різні методи запитів (GET, POST, PUT, DELETE тощо) і давати змогу встановлювати заголовки та параметри запиту. Також важливо мати можливість обробляти асинхронні операції та керувати потоками даних;
- реактивний підхід із використанням Rx: Застосунок має використовувати реактивний підхід під час роботи з мережею за допомогою бібліотеки Rx. Це охоплює роботу з реактивними потоками даних, використання операторів для перетворення і фільтрації даних, а також зручні засоби управління підписками і обробки помилок[8];
- призначений для користувача інтерфейс: Застосунок має надавати зручний та інтуїтивно зрозумілий користувацький інтерфейс для створення та налаштування реактивних запитів, управління потоками даних і перегляду результатів запитів. Інтерфейс має бути розроблено з урахуванням особливостей платформ iOS і macOS, щоб забезпечити однаковий досвід використання на обох платформах;

- обробка різних форматів даних: Застосунок повинен мати можливість працювати з різними форматами даних, такими як JSON або XML. Це дає змогу розробникам обробляти дані, отримані з мережі, і перетворювати їх у потрібний формат для подальшого оброблення або відображення;
- надійність і безпека: Застосунок має бути розроблено з урахуванням вимог до надійності та безпеки. Це охоплює забезпечення стабільної роботи застосунку, опрацювання помилок і виняткових ситуацій, а також захист даних, що передаються мережею. Застосунок має використовувати безпечні протоколи зв'язку і слідувати сучасним рекомендаціям з безпеки даних, щоб захистити інформацію користувачів від несанкціонованого доступу і витоку даних;
- масштабованість: Застосунок має бути розроблено з урахуванням можливості масштабування. Це включає в себе можливість додавання нових функціональних можливостей, підтримку великої кількості користувачів і обробку високого навантаження на мережу. Застосунок має бути здатний ефективно працювати з великим обсягом даних і забезпечувати швидкий відгук користувачеві, навіть за інтенсивного використання;
- тестування та налагодження: Важливою вимогою є можливість тестування та налагодження програми. Застосунок має бути розроблено з урахуванням можливості проведення автоматизованого тестування, щоб забезпечити його стабільну роботу та високу якість[10]. Також необхідні інструменти та механізми для налагодження застосунку, щоб швидко знаходити та виправляти помилки;
- документація та підтримка: Застосунок має бути супроводжений хорошою документацією, яка описує його функціональність, можливості та використання. Документація має бути зрозумілою і

доступною для розробників, щоб полегшити інтеграцію застосунку в наявні проєкти. Крім того, необхідно надати підтримку розробникам, щоб допомогти їм розібратися в застосунку і розв'язати питання або проблеми, що виникають.

Представлений у вигляді Pod-розширення інструмент для реактивної роботи з мережею на платформах iOS і macOS має такий функціонал.

По-перше, він забезпечує зручну інтеграцію з мережевими запитами і відповідями, даючи змогу використовувати реактивні потоки для їх обробки.

По-друге, інструмент надає можливість автоматичного перетворення даних, отриманих із мережі, в об'єкти моделей за допомогою операторів маппінгу.

По-третє, він підтримує реактивне кешування даних, що дає змогу підвищити продуктивність і зменшити навантаження на мережу. Крім того, інструмент має механізми обробки помилок і перехоплення винятків, забезпечуючи надійну роботу з мережею. Нарешті, він надає можливість зручного налаштування і конфігурації запитів, включно з параметрами, заголовками і типами даних.

У результаті аналізу вимог до програмного забезпечення, можна виділити важливі аспекти, такі як мультиплатформеність, інтеграція з бібліотекою Rx, робота з мережею, користувальницький інтерфейс, надійність і безпека, масштабованість, тестування і налагодження, а також документація і підтримка. Врахування цих вимог дасть змогу розробити функціональний та ефективний застосунок для роботи з мережею з використанням реактивного підходу на платформах iOS і macOS.

Висновки до розділу

На даному етапі було описане предметне середовище, сформульовано постановку задачі та розглянуто існуючі аналоги. Було розглянуто групу схожих інструментів – Alamofire, Мойа, Apollo які теж використовуються для роботи з мережею, але в ході дослідження було виявлено, що інструменти надають не повний функціонал для роботи з мережею, або робота з мережею не є реактивною, або погано імпортується в застосунки які використовують реактивний підхід.

Також було сформульована постановка задачі, призначення, цілі та задачі розробки. Визначено призначення та мету розробки, були досліджені проблеми, які вирішуватиме інструмент.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Моделювання програмного забезпечення - це процес створення абстрактних моделей, які описують різні аспекти застосунку, його компоненти, їхню взаємодію та поведінку. Під час розроблення застосунку, написаного з використанням архітектури MVVM (Model-View-ViewModel) та реактивного програмування (Rx), моделювання є важливим етапом, що допомагає розробникам краще зрозуміти та структурувати застосунок.

В архітектурі MVVM модель представляє бізнес-логіку та стан застосунку. Вона відповідає за обробку даних, взаємодію із зовнішніми джерелами даних, а також надає інтерфейси для отримання та оновлення даних. У моделюванні програмного забезпечення для застосунка з архітектурою MVVM слід створити модель, яка абстрагує основні компоненти бізнес-логіки та визначає, як ці компоненти взаємодіятимуть між собою та із зовнішніми компонентами[2].

Крім моделі, в архітектурі MVVM присутні подання (View) і моделі подання (ViewModel). Подання відповідають за відображення даних і призначений для користувача інтерфейс, а моделі подання забезпечують зв'язок між поданнями та моделлю, обробляючи користувацькі дії та оновлюючи модель і подання відповідно.

Реактивне програмування з використанням Rx (Reactive Extensions) надає інструменти та патерни для роботи з асинхронними та подієво-орієнтованими потоками даних. У контексті моделювання програмного забезпечення для застосунку з використанням Rx, моделювання сфокусовано на визначенні потоків даних, їхніх джерел та обробки.

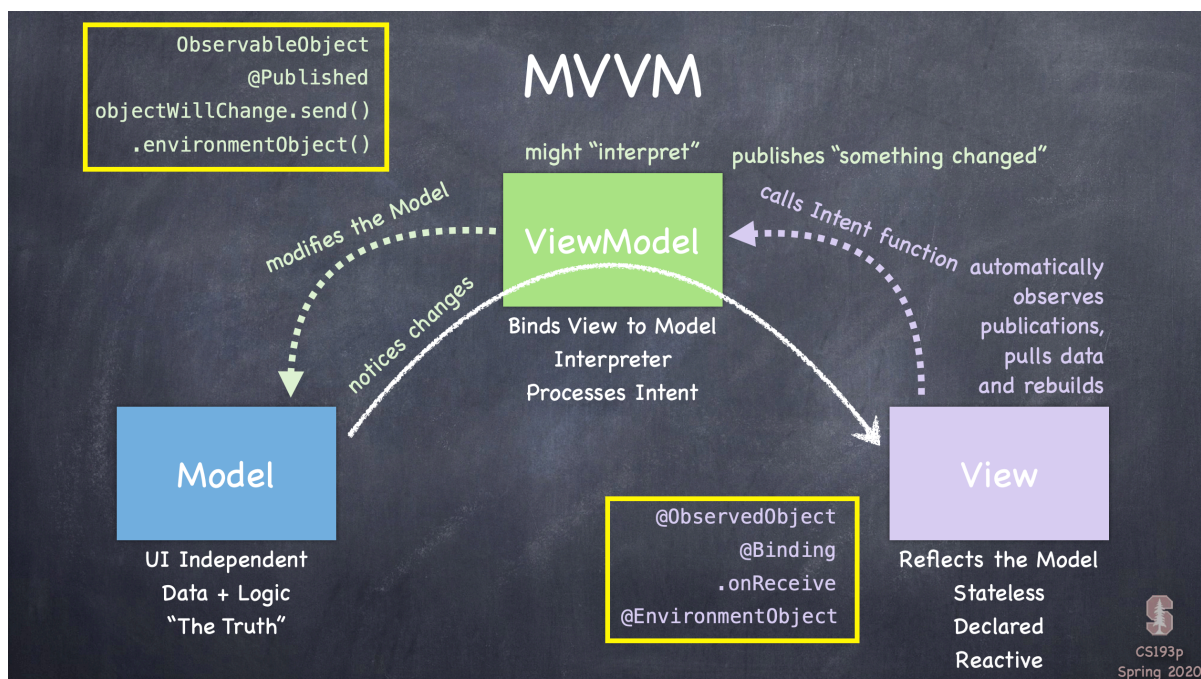


Рисунок 2.1 – Архітектура MVVM

Під час моделювання застосунку, що базується на архітектурі MVVM і використанні Rx, необхідно визначити потоки даних між моделлю, моделями подання та поданнями, а також обробку та перетворення цих даних за допомогою операторів Rx. Це дає змогу створити реактивні ланцюжки обробки даних, які автоматично оновлюють уявлення під час зміни моделі та забезпечують реактивну поведінку застосунку.

Таким чином, моделювання програмного забезпечення для програми, написаної з використанням архітектури MVVM і реактивного програмування з використанням Rx, включає визначення моделі, що описує бізнес-логіку та стан застосунку, моделей подання, що пов'язують модель із поданнями та обробляють користувацькі дії, а також подань, що відображають дані та користувацький інтерфейс.

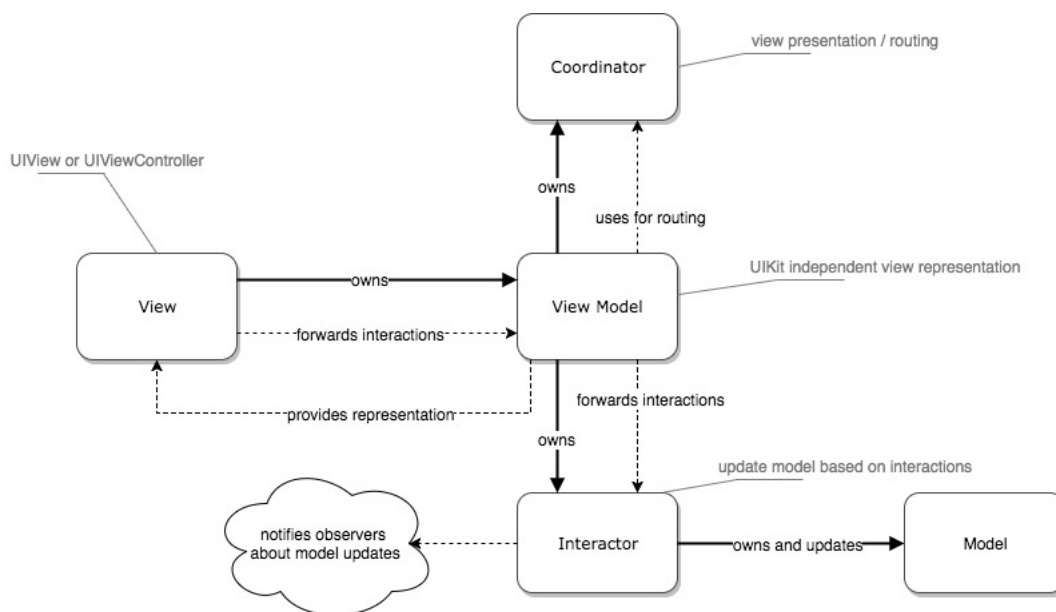


Рисунок 2.2 – Діаграма архітектури MVVM з патерном Coordinator

Під час моделювання необхідно також визначити потоки даних з використанням Rx, визначити джерела даних, як-от мережеві запити або призначені для користувача вводи, і створити реактивні ланцюжки оброблення даних з використанням операторів Rx. Це дає змогу автоматично оновлювати уявлення під час зміни даних, підтримувати реактивну поведінку та керувати асинхронними операціями.

Моделювання програмного забезпечення із застосуванням архітектури MVVM і реактивного програмування допомагає розробникам створювати більш гнучкі, масштабовані та чуйні застосунки. Воно спрощує розподіл відповідальності між компонентами застосунку, забезпечує легкість тестування і повторного використання коду, а також сприяє поліпшенню підтримованості застосунку.

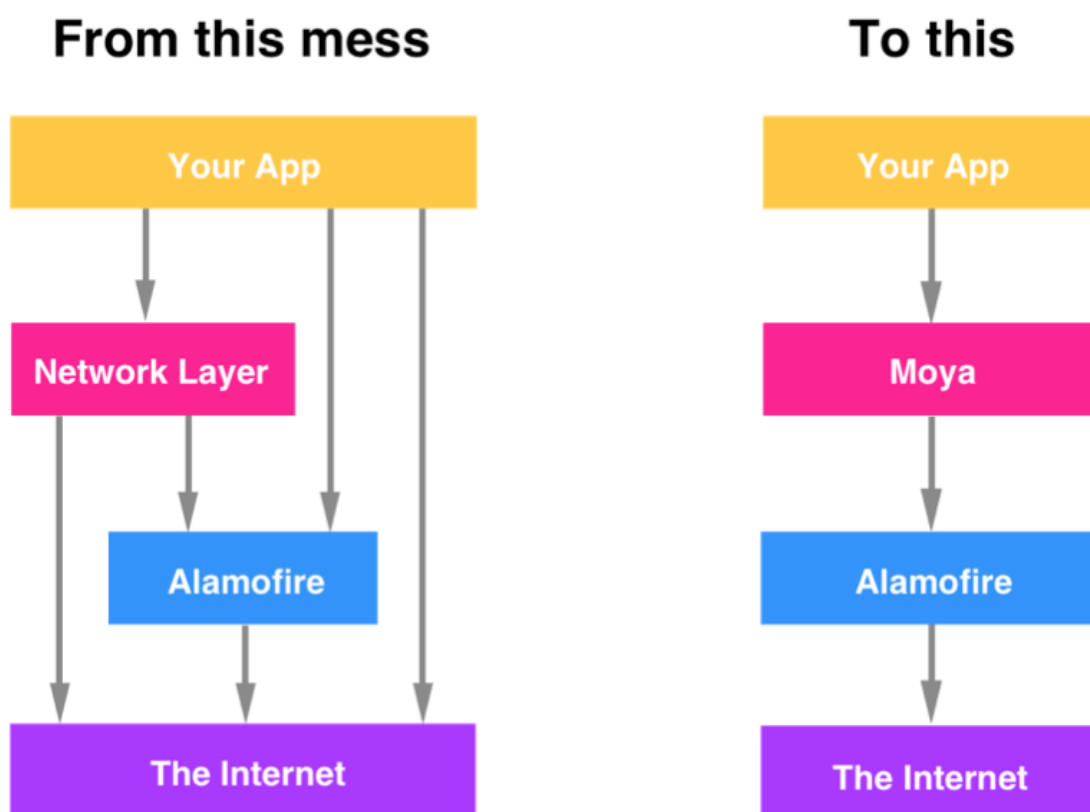


Рисунок 2.3 – Діаграма моделювання покращеної архітектури з використанням бібліотеки для роботи з мережею

У підсумку, моделювання програмного забезпечення для застосунку, що базується на архітектурі MVVM і реактивному програмуванні з використанням Rx, допомагає розробникам створити ефективне й елегантне рішення, що забезпечує чудовий користувацький досвід і спрощує розробку та підтримку застосунку[11].

2.2 Концепція розробки мобільного застосунку під iOS

Swift – мова програмування для IOS, macOS, watchOS, tvOS. Він був створений компанією Apple у 2014 році. Оскільки розробкою займалася одна з найбільших технологічних компаній у світі, то ця мова

програмування зайняла широку частину ринку розробки для пристроїв Apple.

Особливості мови програмування Swift в тому що це open-source а значить велике співтовариство розробників стежить за розвитком мови, вносить свої актуальні застосунки ґрунтуючись на досвіді програмування під платформу Apple[12]. Ця мова також є набагато безпечнішою в порівнянні з іншими мовами програмування, написання коду мовою Swift вважається «суворим» проте це призводить до того, що число помилок при компіляції мінімальне, а також покращується читання коду.

Мова Swift є більш швидким мовою програмування ніж його попередник - мова Objective-C, яка використовувалася розробниками Apple до презентації Swift і використовується досі. Мова Objective-C, майже втричі повільніша за Swift. Swift так само майже в 5 разів швидше за мову Python. Ці особливості роблять вибір у бік мови Swift популярним.

Мова програмування Swift змінюється та покращується, у 2019 році була представлена версія Swift 5 – останній стандарт використання, Apple мотивує розробників постійно оновлювати кодову базу для нової версії Swift та фреймворків. Це допомагає тримати навіть старі продукти актуальними та працюючими під будь-яку версію операційних систем пристроїв Apple, від мобільних гаджетів до телевізійної приставки.

Swift можна використовувати в одному файлі з Objective-C кодом, а значить, кілька мов програмування можна об'єднати в одному проекті. Це додає гнучкості для розробників продуктів.

Один з дуже важливих пунктів для вибору мови Swift як інструмент розробки є ARC - Automatic Reference Counting, який є менеджером пам'яті, ця технологія покликана реалізувати збірник сміття, робота з пам'яттю дозволяє підняти продуктивність при роботі програми на 20%. До розробки ARC розробникам Apple доводилося вручну дбати про пам'ять, що призводило до витоків пам'яті, необхідно було вважати об'єкти та

посилання на об'єкти в кожному класі, щоб не забувати звільняти пам'ять, зараз цим займається мова Swift.

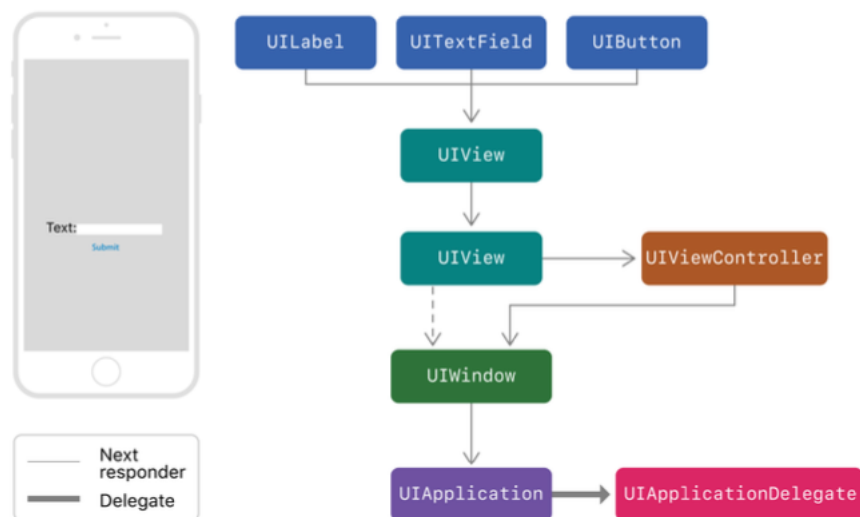


Рисунок 2.4 – Спадкування класів UIKit

Swift є протокольною мовою, що означає розробка ПЗ мовою Swift і архітектура застосунків спирається на протоколи - шаблони методів, властивостей які присвоюються класам, структурам і перерахуванням для того, щоб ті дотримувалися загальних правил для всіх присвоєних класів. Протоколи є аналогом інтерфейсу інших мов програмування. Базою розробки програми з візуальним інтерфейсом використовуючи мову Swift є фреймворк CocoaTouch, він дозволяє створювати застосунки під пристрої з інтерактивним екраном - iPhone, iPad, фреймворк містить компоненти для обробки івентів отриманих від інтерактивного екрану, а також роботу з елементами візуального інтерфейсу мобільного застосунка[13]. Як шаблон проектування у фреймворці CocoaTouch використовується Model-View- Controller. Інструменти фреймворку CocoaTouch включені до IOS SDK, яке було випущено в 2009 році для розробників для мобільної платформи IOS. Можливості CocoaTouch фреймворку включають роботу з анімацією – CoreAnimation,

багатозадачність та розпізнавання мультитач-жестів. У CocoaTouch входять фреймворки Foundation і UIKit, бібліотеки, що містять специфічні IOS класи. Також фреймворк зберігає GameKit для реалізації взаємодії з Game Center (реалізовано в частині освітніх мобільних ігор) та MapKit – фреймворк для роботи з картами[14]. Фреймворк UIKit це view архітектура для імплементації інтерфейсу.

Базою фреймворка UIKit є view – базовий візуальний елемент, від якого успадковуються всі подальші інтерактивні елементи інтерфейсу. Спадкування дозволяє використовувати спільний функціонал view у кожному елементі з бібліотеки елементів або у створеному вручну елементі. Наприклад, кнопка UIButton може мати колір і ширину межі так само, як і таблиця UITableView.

2.3 Концепція розробки десктопного застосунку під MacOS

Розробка під macOS має свої особливості порівняно з розробкою під iOS. Ось кілька ключових аспектів і порівняння бібліотек, які використовуються для розробки під обидві платформи:

Архітектура і UI: macOS і iOS мають різні архітектури застосунків і призначені для користувача інтерфейси. Розробляючи під macOS, вам необхідно враховувати великі екрани, багатозадачність і застосункові можливості, такі як меню та панелі інструментів. Водночас iOS фокусується на мобільних пристроях з обмеженим екраном і взаємодії через сенсорний екран.

Бібліотеки: Для розробки під macOS використовуються бібліотеки, які спеціально призначені для цієї платформи, такі як AppKit і SwiftUI. AppKit є основною бібліотекою для створення застосунків під macOS і надає широкий набір інструментів для роботи з інтерфейсом користувача, файловою системою та іншими системними функціями. SwiftUI - це

сучасна декларативна бібліотека для створення користувацького інтерфейсу на основі Swift, яка надає спільний код для iOS, macOS та інших платформ Apple[15].

Фреймворки та сервіси: У розробці під macOS можна використовувати різні фреймворки та сервіси, які надаються Apple для розширення функціональності застосунку. Наприклад, для роботи з мережею можна використовувати фреймворк URLSession, а для роботи з базою даних - Core Data.

Доступ до функцій macOS: macOS надає застосункові функції, як-от робота з файловою системою, взаємодія з іншими застосунками та використання розширень. Це відкриває застосункові можливості для розробників застосунків під macOS[16].

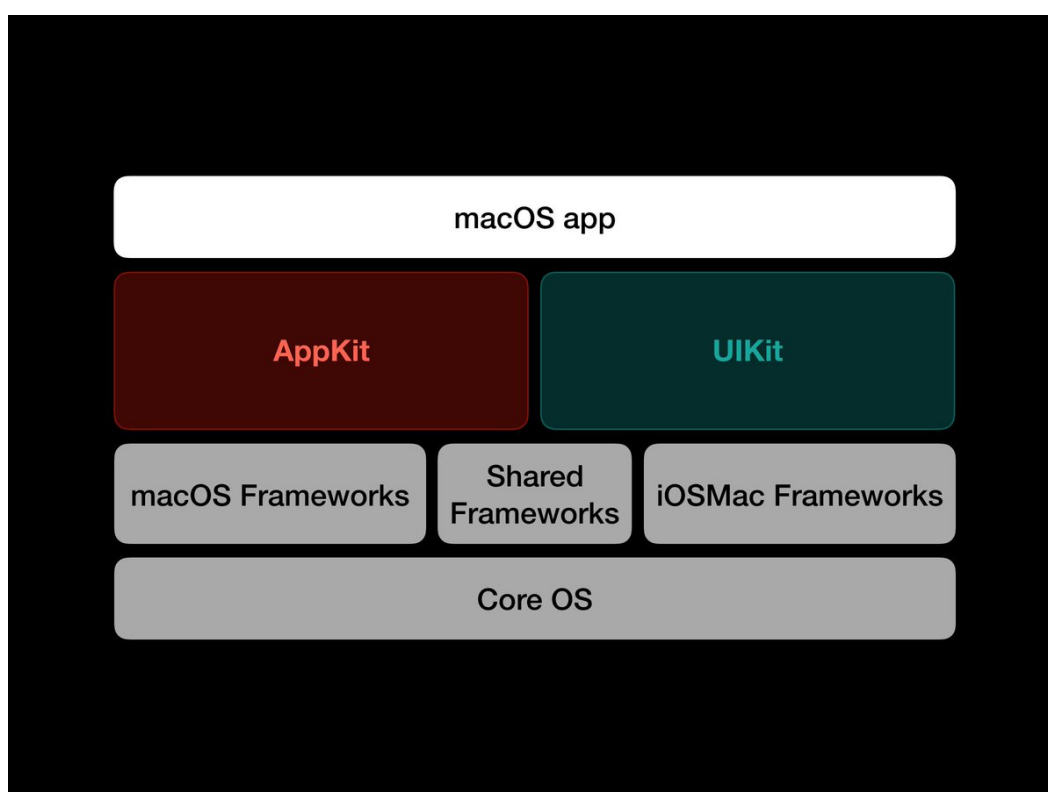


Рисунок 2.5 – Бібліотека AppKit для розробки застосунку під MacOS

Спільна кодова база: Важливою перевагою розробки під macOS та iOS є можливість використовувати загальну кодову базу, особливо під час

використання фреймворка SwiftUI. Це дає змогу ефективно розробляти і підтримувати застосунки для обох платформ.

Загалом, розробка під macOS вимагає врахування особливостей платформи та використання відповідних бібліотек і фреймворків. Однак, із загальними концепціями та інструментами, такими як мова Swift і Xcode, розробники можуть використовувати свій досвід у розробці під iOS і застосовувати його у створенні застосунків для macOS. Це спрощує процес переходу між платформами та підвищує ефективність розробки.

Однак, важливо зазначити, що розробка під macOS вимагає врахування особливостей цієї платформи та адаптації інтерфейсу застосунку під великі екрани і різні ввідні пристрої. Також необхідно враховувати відмінності в поведінці та вподобаннях користувачів між мобільними та настільними пристроями.

Порівнюючи бібліотеки для розробки під macOS та iOS, можна сказати, що основні концепції та інструменти є схожими. Однак, існують специфічні бібліотеки та фреймворки, як-от AppKit, які оптимізовані для розробки під macOS і надають розширені можливості для роботи з інтерфейсом користувача та системними функціями[17].

У підсумку, розробка під macOS має свої особливості, але з використанням відповідних бібліотек та інструментів, розробники можуть створювати потужні та інтуїтивно зрозумілі застосунки для настільних комп'ютерів, поєднуючи їх з уже наявними iOS-застосунками і забезпечуючи єдиний досвід для користувачів в екосистемі Apple.

На поточний момент, безпосередньо інтегрувати AppKit з бібліотекою RXSwift може бути непросто, оскільки RXSwift в основному орієнтований на роботу з платформою iOS і фреймворком UIKit. Однак, існують деякі інструменти та розширення, які дають змогу додати реактивність у розробку з використанням AppKit.

Один із таких інструментів - ReactiveCocoa, який є реактивним фреймворком, сумісним з AppKit. ReactiveCocoa надає реактивні оператори та інструменти, які можна використовувати для оброблення подій і потоків даних у застосунках, заснованих на AppKit.

Інший інструмент - RxSwiftCommunity/RxAppState, надає інтеграцію між RXSwift і AppKit для обробки змін стану програми. Він дає змогу реагувати на події, як-от активація або деактивація застосунку, зміна фокусу та інші, використовуючи реактивні підходи.

Також варто зазначити, що замість безпосередньо інтегрувати RXSwift з AppKit, можна використовувати загальну логіку і підходи, засновані на реактивності, і адаптувати їх для роботи з AppKit. Наприклад, ви можете використовувати реактивні оператори і шаблони проєктування, як-от MVVM, для спрощення управління даними і подіями в застосунку на базі AppKit[19].

Загалом, інтеграція RXSwift з AppKit може потребувати деяких застосункових зусиль і підходів. Однак, за допомогою інструментів, як-от ReactiveCocoa і RxAppState, а також адаптації реактивних підходів для роботи з AppKit, можна додати реактивність і спростити розроблення застосунків на основі AppKit.

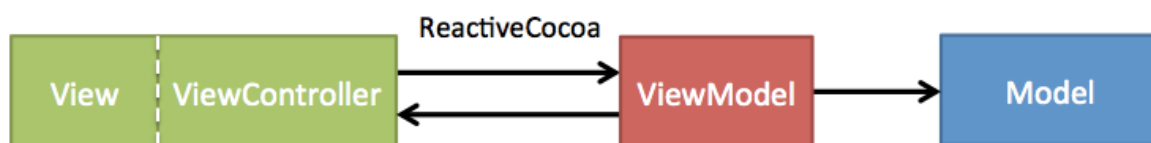


Рисунок 2.6 – Архітектура MVVM з використанням ReactiveCocoa для MacOS

2.4 Порівняння рішень

Combine, RxSwift і ReactiveCocoa є трьома популярними реактивними фреймворками для розробки iOS і macOS застосунків. Ось порівняння цих фреймворків за деякими ключовими аспектами:

Платформа підтримки:

Combine: Розроблений Apple і призначений для роботи на платформах iOS, macOS, watchOS і tvOS.

RxSwift: Незалежний від Apple проєкт, який підтримує платформи iOS, macOS, watchOS, tvOS і Linux.

ReactiveCocoa: Незалежний проєкт, розроблений GitHub, який підтримує платформи iOS і macOS.

Інтеграція з платформою:

Combine: Надає нативну інтеграцію з фреймворками Apple, такими як SwiftUI, Foundation і UIKit.

RxSwift: Також надає інтеграцію з фреймворками Apple, але вимагає використання своїх власних типів даних і операторів.

ReactiveCocoa: Інтегрується з фреймворком AppKit для розробки застосунків під macOS, а також підтримує інтеграцію з іншими фреймворками Apple.

Синтаксис і API:

Combine: Має новий синтаксис, заснований на об'єднанні функцій і операторів, а також надає типобезпечні потоки даних.

RxSwift: Має синтаксис, що ґрунтується на спостережуваних послідовностях (observables) і операторах, що робить його зрозумілішим для розробників, знайомих із реактивним програмуванням.

ReactiveCocoa: Має синтаксис, схожий на RxSwift, але також пропонує свої власні концепції, такі як сигнали (signals) і сигнальні значення (signal values).

Підтримка архітектурних шаблонів:

Combine, RxSwift і ReactiveCocoa надають підтримку архітектурних шаблонів, таких як MVVM (Model-View-ViewModel), що полегшує розробку реактивних застосунків і обмін даними між компонентами.

Екосистема та спільнота:

Combine: Оскільки Combine є фреймворком Apple, він має велику підтримку з боку Apple і широку екосистему, включно з документацією, прикладами коду та інструментами розробки.

RxSwift: володіє великою і активною спільнотою розробників, які активно вкладаються в розвиток і підтримку фреймворку. Є безліч ресурсів, форумів і репозиторіїв із загальнодоступним кодом, де можна знайти корисні приклади і рішення для роботи з RxSwift.

ReactiveCocoa: Хоча спільнота ReactiveCocoa не така широка, як у Combine і RxSwift, однаково існує певна кількість активних розробників, які підтримують і розвивають фреймворк. Документація та приклади коду також доступні для використання та вивчення.

Залежності та сумісність:

Combine: Повністю інтегрований з іншими фреймворками Apple і використовує новий тип даних Publisher для роботи з потоками даних.

RxSwift: Може бути використаний разом з іншими фреймворками Apple і не вимагає великих змін в існуючому коді.

ReactiveCocoa: Також може бути використаний у спільній роботі з іншими фреймворками Apple, включно з AppKit та іншими.

Загалом, усі три фреймворки - Combine, RxSwift і ReactiveCocoa - надають інструменти та концепції для реактивної розробки на платформі iOS і macOS. Вибір конкретного фреймворка залежить від ваших уподобань, досвіду та вимог проєкту. Combine є нативним фреймворком Apple з широкою підтримкою та інтеграцією, в той час як RxSwift і

ReactiveCocoa пропонують більш гнучкий синтаксис і більш широку підтримку платформи macOS.

2.5 Рішення для реактивної роботи з мережею

Проект SimpleTunnel містить робочі приклади чотирьох точок розширення, що надаються фреймворком Network Extension:

Провайдер пакетного тунелю

Пункт розширення Packet Tunnel Provider використовується для реалізації клієнтської частини користувацького протоколу мережевого тунелювання, який інкапсулює мережеві дані у вигляді IP-пакетів. Ціль PacketTunnel у проекті SimpleTunnel створює зразок розширення Packet Tunnel Provider.

App Proxy Provider

Точка розширення App Proxy Provider використовується для реалізації клієнтської частини користувацького мережевого проксі-протоколу, який інкапсулює мережеві дані у вигляді потоків мережевих даних застосунків. Підтримуються як потоки даних на основі TCP або потоків, так і потоки даних на основі UDP або дейтаграм. Ціль AppProxy у проекті SimpleTunnel створює зразок розширення App Proxy Provider.

Два розширення Filter Provider використовуються для реалізації динамічного фільтрації мережевого вмісту на пристрої. Розширення Filter Data Provider відповідає за перевірку мережевих даних і прийняття рішень про пропуск/блокування. Пісочниця розширення Filter Data Provider не дозволяє розширенню взаємодіяти з мережею або записувати на диск, щоб запобігти витоку мережевих даних. Розширення Filter Control Provider може обмінюватися даними через мережу і записувати на диск. Він відповідає за оновлення правил фільтрації від імені розширення Filter Data Provider.

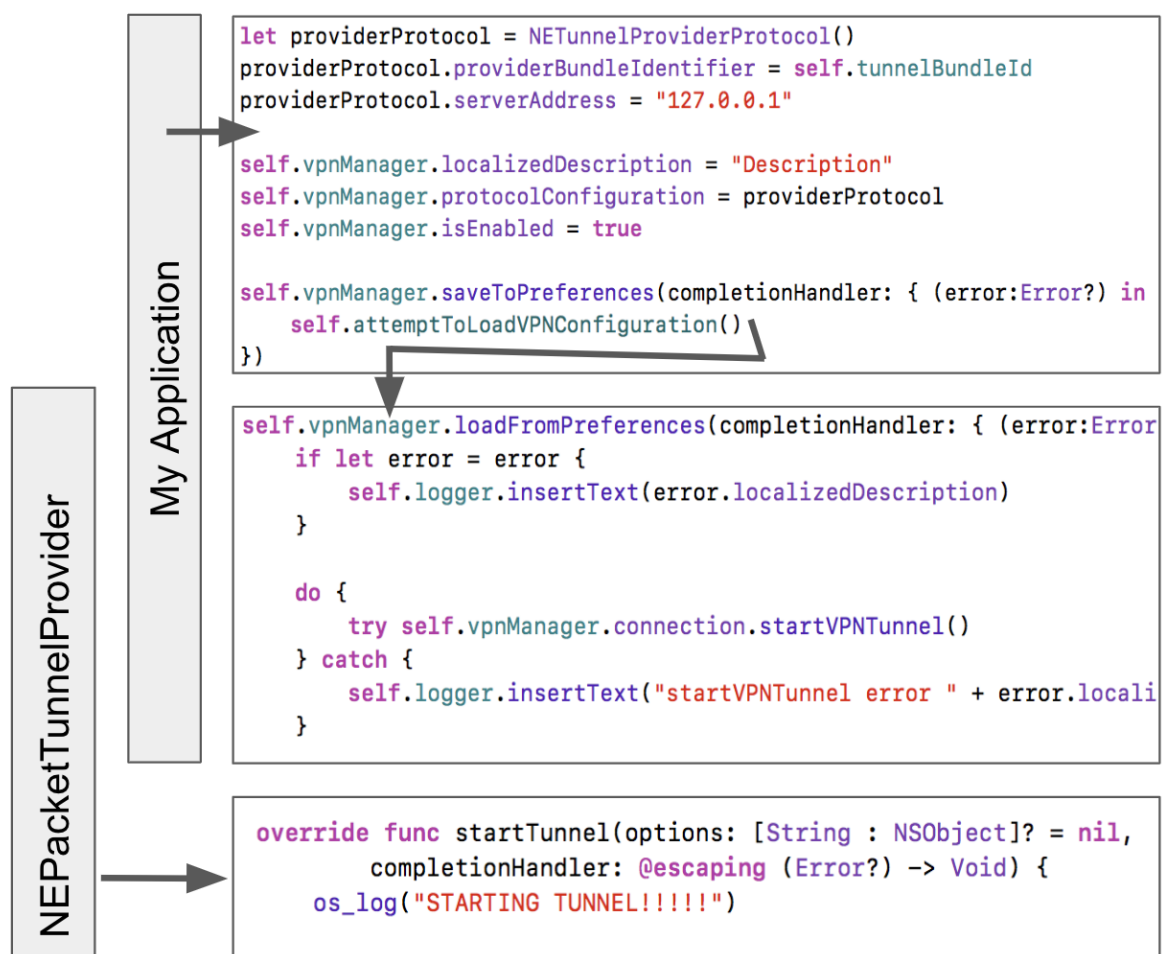


Рисунок 2.7 – Packet Tunnel Provider для мережевих викликів

Ціль `FilterDataProvider` у проєкті `SimpleTunnel` створює зразок розширення `Filter Data Provider`. Ціль `FilterControlProvider` у проєкті `SimpleTunnel` створює зразок розширення `Filter Control Provider`.

Усі зразки розширень упаковано у застосунок `SimpleTunnel`. Застосунок `SimpleTunnel` містить код, який демонструє, як налаштовувати і контролювати різні типи постачальників мережевих розширень. Ціль `SimpleTunnel` у проєкті `SimpleTunnel` створює застосунок `SimpleTunnel` і всі зразки розширень.

Проект `SimpleTunnel` містить як клієнтську, так і серверну частини користувацького протоколу мережевого тунелювання. Розширення `Packet Tunnel Provider` і `App Proxy Provider` реалізують клієнтську частину. Ціль

tunnel_server створює двійковий файл командного рядка OS X, який реалізує серверну частину. Сервер налаштовується за допомогою списку. Зразок списку включено до вихідного коду tunnel_сервера. Щоб запустити сервер, скористайтеся цією командою: `sudo tunnel_server`

У RxSwift, бібліотеці реактивного програмування для мови Swift, існує кілька інструментів, які ефективно підтримують архітектуру MVVM (Model-View-ViewModel). Ці інструменти допомагають керувати потоками даних, реагувати на зміни та пов'язувати моделі подання з поданнями.

Ось деякі ключові інструменти RxSwift для архітектури MVVM:

Observables: Observables представляють джерела даних, які можуть випромінювати значення під час виконання програми. У контексті MVVM, модель надає Observables для надання даних уявленням. Подання підписуються на Observables, щоб отримувати оновлення даних.

Bindings: Біндинги (bindings) використовуються для зв'язування Observables з елементами користувацького інтерфейсу. Вони дають змогу автоматично оновлювати подання під час зміни значень у Observables. Наприклад, біндинг може пов'язувати текстове поле з Observables, щоб відображати й автоматично оновлювати дані в реальному часі.

Subjects: Subjects є особливим різновидом Observables, які можуть динамічно генерувати та надсилати значення. Вони використовуються для взаємодії між моделлю подання та поданнями. Модель подання може створювати Subject і надсилати значення в нього, а подання можуть підписуватися на Subject, щоб отримувати і реагувати на ці значення.

Transformations: RxSwift пропонує широкий набір операторів, які дають змогу перетворювати, фільтрувати та комбінувати Observables. Вони можуть бути використані в моделях подання для обробки і перетворення даних, а також для реагування на різні події та умови.

Command: Команда (Command) представляє дію, яку може бути виконано у відповідь на введення користувача або інші події. Команди

можуть бути прив'язані до елементів призначеного для користувача інтерфейсу і виконуватися з використанням Observables і біндингів. Це зручний інструмент для реалізації логіки обробки користувацьких дій у моделях подання.

Ці інструменти дають змогу розробникам створювати більш реактивні та гнучкі моделі подання в архітектурі MVVM. Вони спрощують управління потоками даних, зв'язування з поданнями та реагування на зміни.

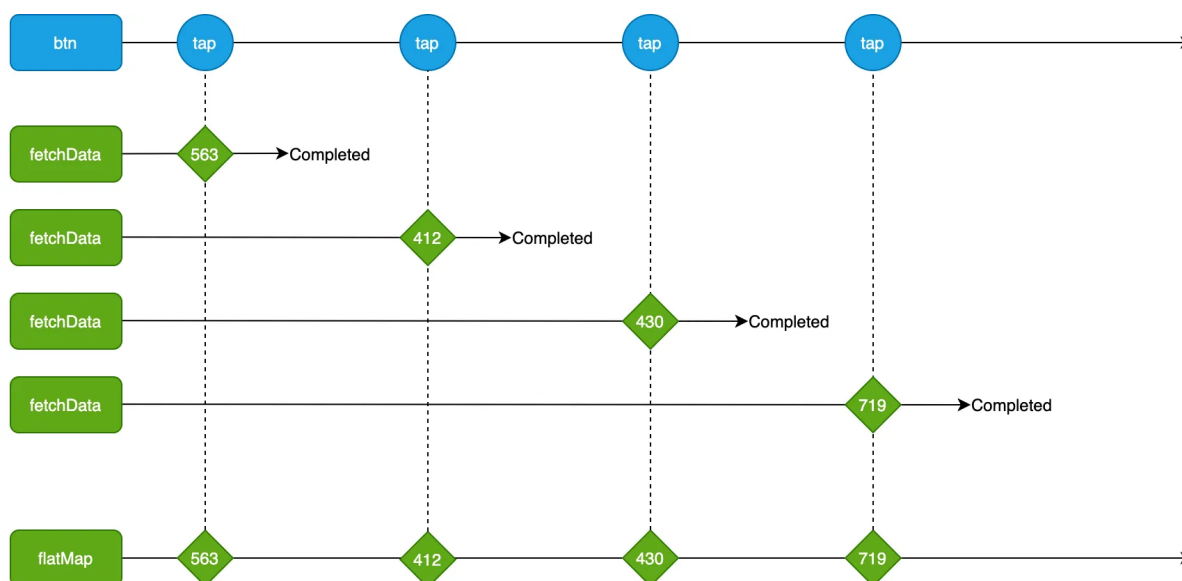


Рисунок 2.8 – Діаграма управління потоками з використанням реактивних методів

Під час роботи з реактивним програмуванням мережеві запити є найпопулярнішою логікою, з якою ми часто працюємо. Сьогодні ми обговоримо, як правильно обробляти послідовність мережевих запитів, виправимо деякі поширені помилки, а також запропонуємо деякі загальні підходи, які більшість розробників часто використовують у реальних проектах. Ми будемо використовувати RxSwift для демонстрації коду, але оскільки ReactiveX є крос-платформенним, то можна також задіяти RxJava, RxKotlin або навіть Combine.

Почнемо із загальної вимоги, у нас є UIButton, яка буде запускати мережевий запит для отримання даних з сервера. Ми відобразимо дані у вигляді виводу Observable за допомогою UITextView. Ви можете запустити завантажений проект, щоб побачити інтерфейс і потік:

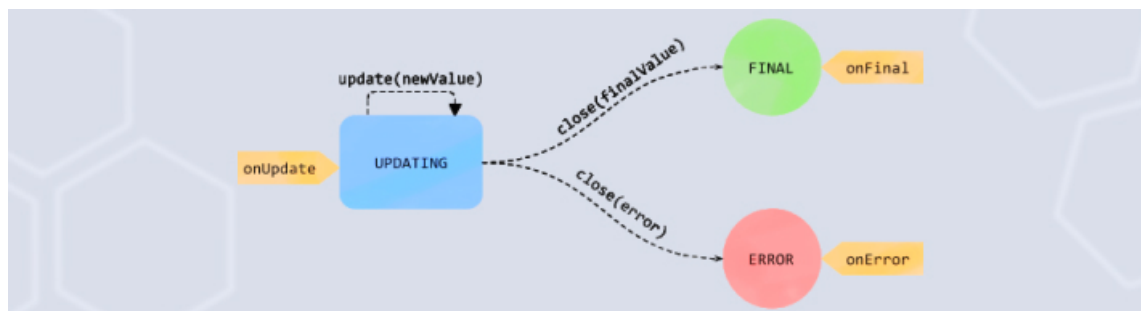


Рисунок 2.9 – Асинхронна обробка подій

RxSwift - це бібліотека для написання асинхронного та подієвого коду за допомогою спостережуваних послідовностей та операторів функціонального стилю, що дозволяє параметризувати виконання за допомогою планувальників. RxSwift за своєю суттю спрощує розробку асинхронних програм, дозволяючи коду реагувати на нові дані та обробляти їх послідовно та ізольовано.

RxSwift та реактивне програмування

Розуміння концепцій реактивного програмування спочатку може здатися складним, але коли ви познайомитеся з реактивним програмуванням та RxSwift, це допоможе вам писати кращі застосунки. Особливо при написанні асинхронного та подієвого коду, RxSwift допоможе вам підтримувати кодову базу з мінімальними зусиллями.

Objects graph of Producer-based subscription

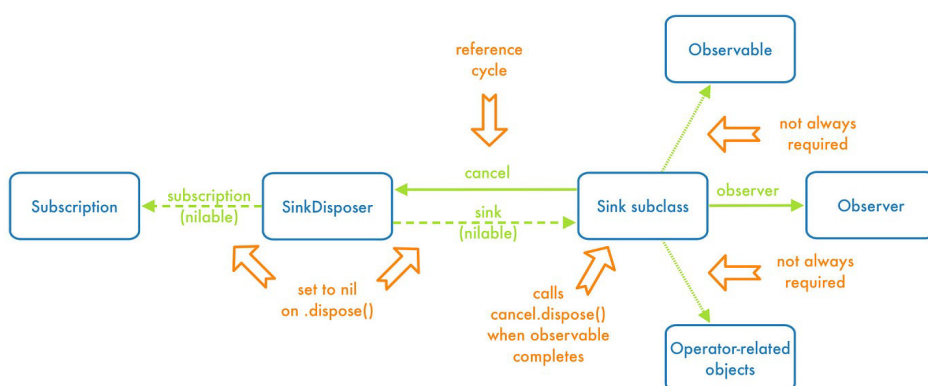


Рисунок 2.10 – Граф обробки подій на основі producer subscription

Асинхронне програмування - це спосіб паралельного програмування, при якому частина роботи виконується окремо від основного потоку програми і повідомляє викликаючий потік про своє завершення, збій або прогрес. Вам може бути цікаво, коли варто використовувати асинхронне програмування і які його переваги та проблеми.

Основні переваги, які можна отримати від використання асинхронного програмування, - це покращена продуктивність та швидкість реакції програми. Особливо добре асинхронний патерн підходить для створення адаптивного інтерфейсу в клієнтському застосунку під час виконання ресурсомістких операцій, що вимагають великих обчислень.

Щоб пояснити асинхронне програмування простим способом, давайте наведемо приклад з застосунку для iOS. У застосунку для iOS в будь-який момент ми можемо виконувати одну з наступних дій:

- реагування на натискання кнопок;
- анімація переходів у UIView;
- завантаження бітів даних з віддаленого джерела;
- відтворення аудіо або відео;
- збереження бітів даних на диск.

Здається, що всі ці дії відбуваються одночасно. Щоразу, коли подання анімується, звук у вашому застосунку не призупиняється, поки анімація не завершиться. Різні частини вашої програми не блокують виконання одна одної. Існують різні види API, які дозволяють виконувати різні частини роботи в різних потоках, у різних контекстах виконання і розподіляти їх між різними ядрами процесора пристрою.

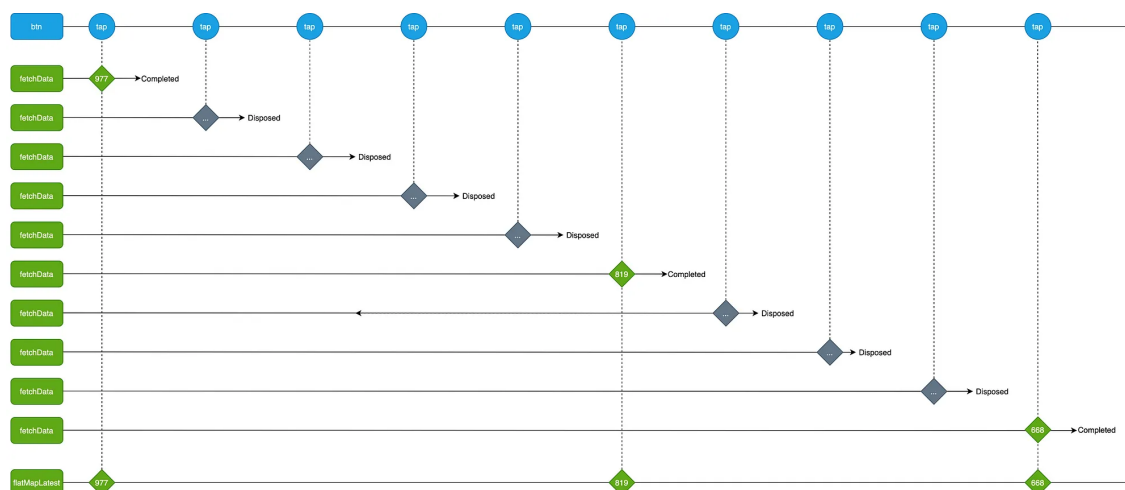


Рисунок 2.11 – Діаграма управління потоками з використанням реактивних методів

Драйвер ніколи не видає помилок, тому я припустив, що його також не можна завершити, але я помилився. Його потенційно можуть завершити інші помилки у його послідовності. Ця діаграма пояснює, чому функціонал крана припиняє працювати, коли виникає помилка мережі:

Отже, щоб послідовність продовжувала працювати в цьому випадку, ми повинні перехопити помилку. Я з'ясував, що якщо дочірній Observable всередині операторів `flatMap` або `flatMapLatest` надсилає завершену подію (оскільки Observable завершується як помилка), то це не призведе до завершення всієї послідовності.

Висновки до розділу

Отже, у цьому розділі було змодельоване та проаналізоване програмне забезпечення, для того, щоб зрозуміти з яких частин воно буде складатися. Було розглянуто предметну область, та розглянуто тонкості розробки мобільних застосунків під iOS та десктопних застосунків під MacOS.

Також у цьому розділі було розглянуто вибір архітектури MVVM і використання бібліотеки RxSwift. Розробка велася з урахуванням особливостей платформ та інтеграції з AppKit і UIKit. Було описано інструменти, що надають інтеграцію AppKit з RxSwift, а також порівняння між різними реактивними бібліотеками. Результатом розробки став потужний і гнучкий інструмент, який надає реактивне рішення для роботи з мережею і має низку переваг порівняно з аналогами.

Висновком є опис процесу розробки інструменту, здатного значно спростити та покращити процес роботи з мережею на платформах iOS та macOS.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментів для розробки

Спершу визначимось з інструментами, які ми будемо використовувати для реалізації. Для реалізації реактивної роботи я обрав два важливих інструменти - RxSwift і Combine. Вони надають потужні засоби для роботи з асинхронними подіями, даючи змогу розробникам створювати чуйні та ефективні застосунки. У цьому розділі буде детально розглянуто вибір цих інструментів і класів, які будуть адаптовані для реактивної роботи.

Вибір між RxSwift і Combine був заснований на низці факторів. RxSwift - це одна з найпопулярніших бібліотек для реактивного програмування на платформі iOS, і вона має широкую спільноту розробників і велику документацію. З іншого боку, Combine - це новий набір API, розроблений Apple, який надає нативну підтримку реактивного програмування на платформі iOS і MacOS, починаючи з версії iOS 13 і macOS 10.15.

Для нашої магістерської роботи ми вирішили використовувати обидва інструменти. Це дасть нам змогу порівняти їхню продуктивність, зручність використання та інтеграцію в наш застосунок. Такий підхід також дасть змогу забезпечити максимальну сумісність із різними версіями операційних систем.

Для успішної реактивної роботи наших застосунків ми вибрали низку ключових класів, які будуть адаптовані з використанням реактивних обгорт. Ці класи включають в себе:

NEVPNManager: NEVPNManager - це клас, наданий Apple для управління віртуальними приватними мережами (VPN) на пристроях iOS і MacOS. Адаптація NEVPNManager до реактивної роботи дасть нам змогу

гнучкіше керувати VPN-з'єднаннями і реагувати на зміни та події в їхньому стані.

Packet Tunnel Provider: Packet Tunnel Provider - це клас, призначений для реалізації власних VPN-серверів на пристроях iOS і MacOS. Адаптація цього класу до реактивної роботи буде корисною для створення високопродуктивних і чуйних VPN-сервісів.

MapKit: MapKit - це фреймворк Apple для інтеграції карт і геолокації в застосунки. Використання реактивних обгорт для MapKit дасть нам змогу ефективно відстежувати зміни місця розташування, інтеракції з картами та інші події, пов'язані з картами.

WebSocket: WebSocket - це клас, призначений для обміну даними за протоколом WebSocket. Реактивна адаптація WebSocket дасть змогу зручно обробляти вхідні та вихідні повідомлення, а також керувати з'єднанням через реактивні потоки.

У рамках процесу розробки реактивного інструменту для платформ iOS і MacOS буде створено демонстраційний застосунок, що призначений для наочної демонстрації функціональності інструменту. Цей демо-застосунок міститиме кілька ключових екранів, кожен з яких продемонструє переваги реактивного програмування і роботу інструменту.

Екран VPN-підключення:

На цьому екрані користувач зможе взаємодіяти з VPN-сервісом, керованим нашим інструментом. За допомогою реактивних потоків, ми продемонструємо можливості підключення до різних VPN-серверів і відстеження стану підключення в реальному часі. Користувач зможе вмикати та вимикати VPN-підключення, а також обирати країну для підключення.

Екран карти:

На цьому екрані буде продемонстровано інтеграцію з MapKit, використовуючи реактивне програмування. Користувач зможе вибрати

місце розташування на карті, а застосунок реагуватиме на зміни вибору місця розташування в режимі реального часу. Ми також можемо інтегрувати дані про VPN-сервери на карту, щоб користувач міг легко обирати країну для підключення.

Екран виміру швидкості інтернету:

Для демонстрації реактивних можливостей у роботі з даними, ми створимо екран для вимірювання швидкості інтернету. Користувач побачить зміни швидкості завантаження і вивантаження в реальному часі, і застосунок реагуватиме на зміни швидкості.

Усі ці екрани працюватимуть у реальному часі, оновлюючи дані та взаємодіючи з користувачем із використанням реактивних принципів. Такий підхід дасть змогу продемонструвати переваги реактивного програмування та роботу інструменту на практиці.

3.2 Розробка інструменту для роботи з VPN

Розробка реактивних обгорт для методів класів NEVPNManager і Packet Tunnel Provider з використанням Combine надає зручний спосіб інтеграції реактивного програмування у ваш проєкт. Це дає змогу керувати асинхронними операціями та подіями, пов'язаними з VPN-сервісами, а також створювати застосунки, здатні ефективно реагувати на зміни стану.

Для початку розглянемо, як можна створити реактивну обгортку для методів NEVPNManager з використанням Combine. Припустимо, що у нас є метод `connectToVPNServer` у NEVPNManager, який асинхронно встановлює VPN-з'єднання. Ми хочемо створити реактивний інтерфейс для цього методу.

```

class VPNManagerWrapper {
    private let vpnManager = NEVPNManager.shared()

    func connectToVPNServer() -> AnyPublisher<Bool, Error> {
        return Future<Bool, Error> { [weak self] promise in
            self?.vpnManager.loadFromPreferences { error in
                if let error = error {
                    promise(.failure(error))
                } else {
                    self?.vpnManager.connection.startVPNTunnel { error in
                        if let error = error {
                            promise(.failure(error))
                        } else {
                            promise(.success(true))
                        }
                    }
                }
            }
        }
        .eraseToAnyPublisher()
    }
}

```

Рисунок 3.1 – Реактивна обгортка для класу NEVPNManager

У цьому прикладі використано Future і AnyPublisher для створення реактивної обгортки для методу connectToVPNServer. Тепер можна підписатися на цей обгорнутий метод за допомогою Combine і обробляти результати та помилки у застосунку.

Аналогічно, створимо реактивну обгортку для методів Packet Tunnel Provider. Припустимо, у вас є метод startTunnel для початку роботи VPN-сервера.

Таким чином, методи NEVPNManager і Packet Tunnel Provider у реактивні обгортки, які повертають AnyPublisher. Це дасть змогу інтегрувати їх у наш проєкт, підписуватися на результати операцій і реагувати на зміни стану VPN-сервісів за допомогою Combine. Застосунок стане більш чуйним і легше керованим в умовах асинхронних операцій, характерних для роботи з VPN.

```

14
15 import NetworkExtension
16 import Combine
17
18 class PacketTunnelProviderWrapper {
19     private let tunnelProvider = NETunnelProviderManager.shared()
20
21     func startTunnel() -> AnyPublisher<Bool, Error> {
22         return Future<Bool, Error> { [weak self] promise in
23             self?.tunnelProvider.loadFromPreferences { error in
24                 if let error = error {
25                     promise(.failure(error))
26                 } else {
27                     do {
28                         try self?.tunnelProvider.connection.startVPNTunnel()
29                         promise(.success(true))
30                     } catch {
31                         promise(.failure(error))
32                     }
33                 }
34             }
35         }
36         .eraseToAnyPublisher()
37     }
38 }
39

```

Рисунок 3.2 – Реактивна обгортка для класу Packet Tunnel Provider

Аналогічно створимо реактивну обгортку для інших методів та властивостей.

shared(): Статичний метод для отримання екземпляра NETunnelProviderManager. Повертає спільний екземпляр менеджера для керування Packet Tunnel Provider.

loadFromPreferences(completionHandler:): Метод для завантаження налаштувань Packet Tunnel Provider із системних налаштувань. Приймає замикання, яке буде викликано після завантаження налаштувань.

saveToPreferences(completionHandler:): Метод для збереження поточних налаштувань Packet Tunnel Provider у системних налаштуваннях. Приймає замикання, яке буде викликано після збереження.

removeFromPreferences(completionHandler:): Метод для видалення налаштувань Packet Tunnel Provider із системних налаштувань. Приймає замикання, яке буде викликано після видалення.

connection: Властивість, що надає доступ до об'єкта NETunnelProviderSession, який дозволяє керувати станом Packet Tunnel Provider.

protocolConfiguration: Властивість, що надає доступ до налаштувань протоколу для Packet Tunnel Provider.

isEnabled: Властивість, що вказує, чи ввімкнено Packet Tunnel Provider.

За допомогою розробленого інструменту для реактивного підключення до VPN тепер можна розробити застосунок. На головному екрані буде 4 кнопки: кнопка вибору країни для підключення, кнопка “Connect” для підключення до вибраної країни, кнопка переходу до екрану з заміром швидкості мережі та кнопка переходу до екрани з інтерактивною картою вибору країни для підключення.

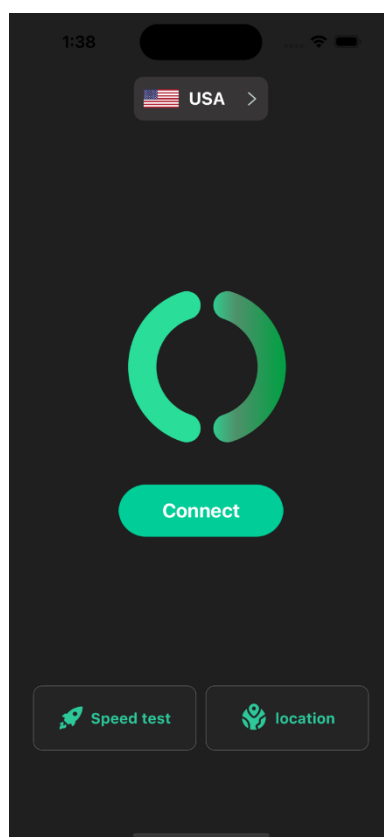


Рисунок 3.3 – Головний екран мобільного застосунку у стані, коли не підключено до мережі

У головного екрану є 2 стани: перший коли користувач не підключений до VPN, та другий, коли підключений. Після підключення на головному екрані запускається анімація, та після встановлення зв'язку вона припиняється, з'являється лічильник часу підключення, та плашки з поточним IP та локацією користувача.

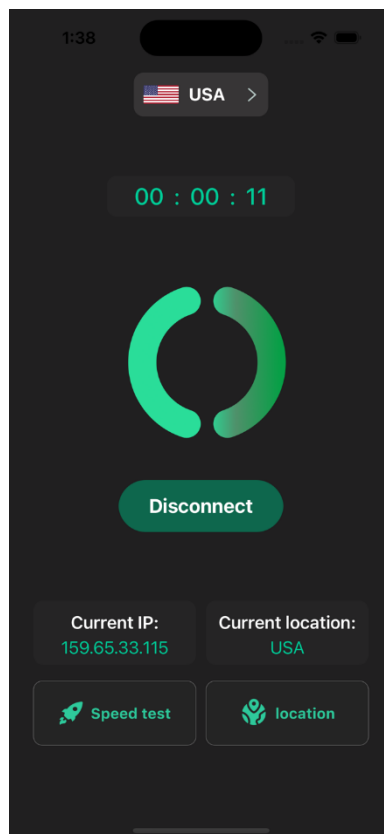


Рисунок 3.4 – Головний екран мобільного застосунку у стані, коли підключено до мережі

3.3 Розробка інструменту для роботи з картами

Розробка реактивної обгортки для інтеграції MapKit з використанням Combine дає змогу створити чуйний і керований інтерфейс для роботи з картами та геолокацією. Це полегшує опрацювання змін місця розташування і взаємодію з картами в реальному часі. Ось як можна створити таку обгортку:

```

248 class MapKitWrapper: NSObject, ObservableObject, MKMapViewDelegate, CLLocationManagerDelegate {
249     @Published var userLocation: CLLocationCoordinate2D = CLLocationCoordinate2D()
250     @Published var selectedLocation: CLLocationCoordinate2D?
251
252     private var mapView = MKMapView()
253     private var locationManager = CLLocationManager()
254     private var cancellable: [AnyCancellable] = []
255
256     override init() {
257         super.init()
258
259         mapView.delegate = self
260         locationManager.delegate = self
261         locationManager.desiredAccuracy = kCLLocationAccuracyBest
262         locationManager.requestWhenInUseAuthorization()
263
264         $userLocation
265             .receive(on: DispatchQueue.main)
266             .sink { [weak self] location in
267                 self?.mapView.setCenter(location, animated: true)
268             }
269             .store(in: &cancellable)
270     }
271
272     func setupMapView(in view: UIView) {
273         mapView.frame = view.bounds
274         view.addSubview(mapView)
275     }
276
277     func selectLocation(coordinate: CLLocationCoordinate2D) {
278         selectedLocation = coordinate
279     }
280
281     func mapView(_ mapView: MKMapView, regionDidChangeAnimated animated: Bool) {
282         userLocation = mapView.centerCoordinate
283     }
284
285     func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
286         if let newLocation = locations.last?.coordinate {
287             userLocation = newLocation
288         }
289     }
290 }

```

Рисунок 3.5 – Реактивна обгортка для класу MapKit

У цьому прикладі створюємо MapKitWrapper, який є ObservableObject, що дає змогу спостерігати за змінами всередині обгортки. Ми використовуємо @Published властивості для відстеження місця розташування користувача та обраної користувачем локації на карті.

Обгортка ініціалізує MKMapView і CLLocationManager для роботи з картою і геолокацією. Ми також встановлюємо спостереження за змінами місця розташування користувача та оновленнями області карти.

Щоб використовувати цю реактивну обгортку у проєкті, треба створити екземпляр MapKitWrapper і використовувати його методи та властивості для управління MapKit. Також можна підписатися на зміни

всередині обгортки, наприклад, спостерігати за `userLocation` або `selectedLocation`, щоб реагувати на зміни в реальному часі та оновлювати інтерфейс застосунку відповідним чином.

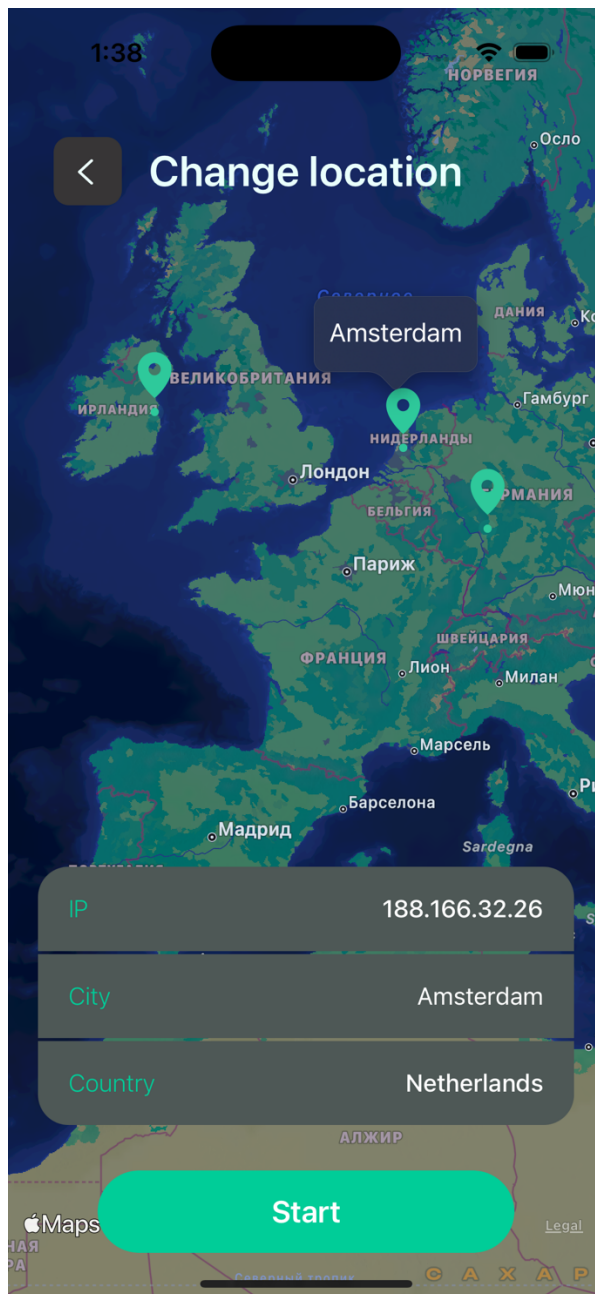


Рисунок 3.6 – Екран з інтерактивною картою для вибору країни

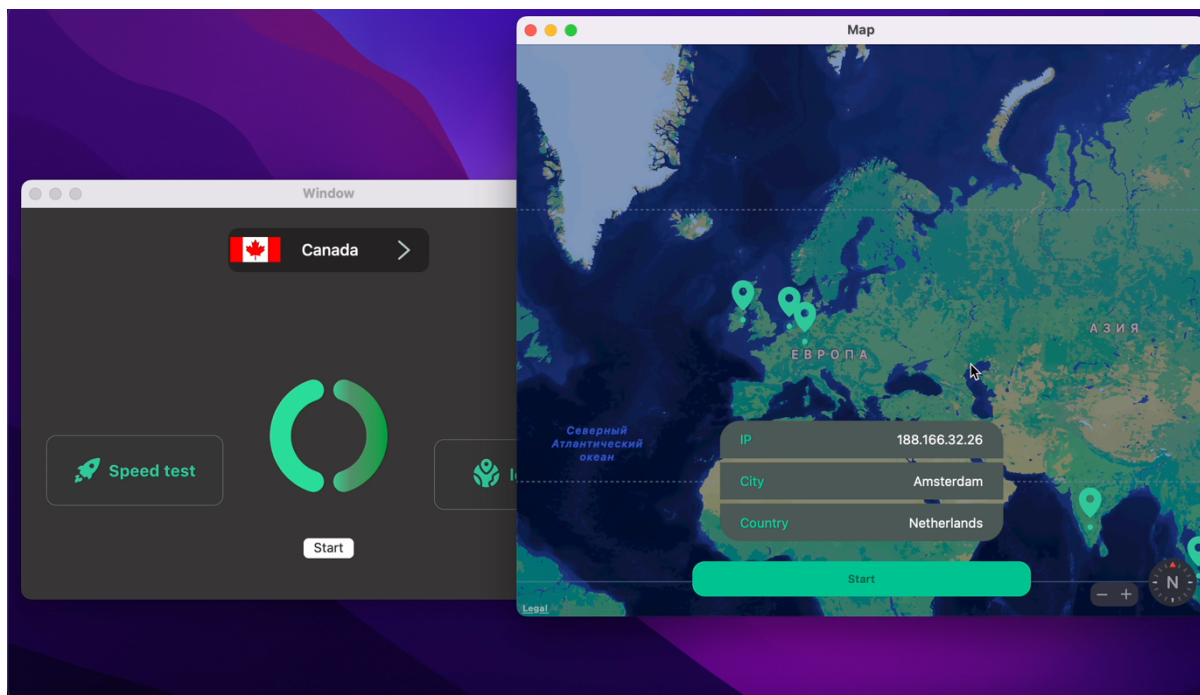


Рисунок 3.7 – Інтерфейс десктопного застосунку на платформі
MacOS

3.4 Структура інструменту

Для описання структури інструменту було використано UML діаграму класів, де основні класи-обгортки для реактивної роботи з мережею будуть представлені з урахуванням їх зв'язків і співвідношень. У кожному класі будуть описані ключові методи і змінні. Діаграма також відображає залежності між класами, що використовуються для різних аспектів реактивної роботи з мережею, наприклад, серіалізацію даних або управління сесіями.

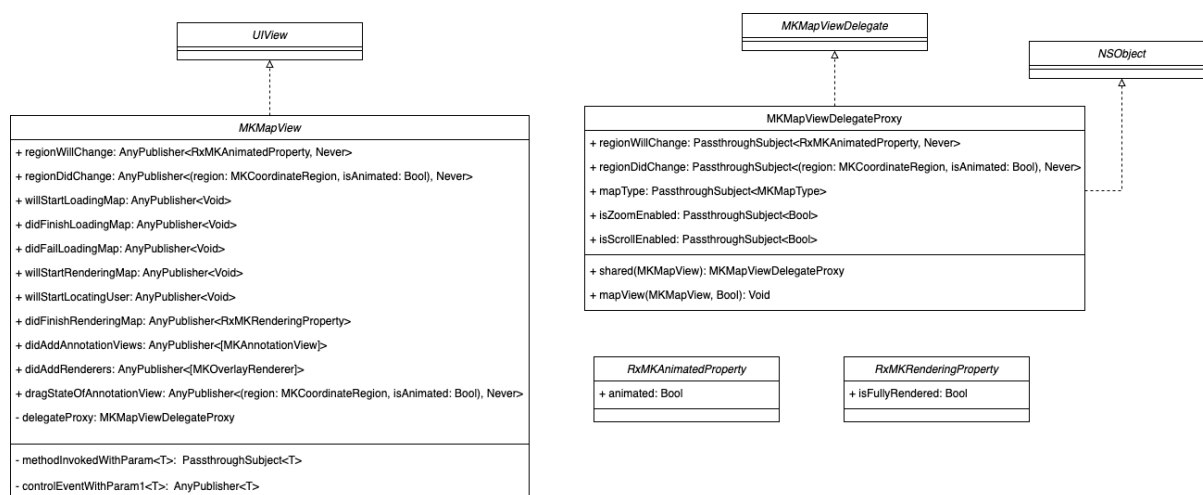


Рисунок 3.8 – UML діаграма класів для реактивної роботи з картами MapKit

В діаграмі на рисунку 3.8 продемонстровано основні класи які використовуються для реактивної роботи з картами MapKit. Основний клас MKMapView містить такі змінні та функції:

- `handleViewForAnnotation` - дозволяє визначити спосіб обробки анотацій на мапі;
- `handleRendererForOverlay` - визначає спосіб обробки накладення на мапі;
- `methodInvokedWithParam1` - внутрішня функція для спрощення виклику методів делегата з одним параметром;
- `controlEventWithParam1` - створює подію управління для відповідного виклику методу делегата;
- `regionWillChange` - реактивна обгортка для події зміни регіону мапи;
- `regionDidChange` - реактивна обгортка для події, що вказує на закінчення зміни регіону;
- `willStartLoadingMap` - реактивна обгортка для події початку завантаження мапи;
- `didFinishLoadingMap` - вказує на закінчення завантаження мапи.

- `didFailLoadingMap` - обгортка для обробки помилок при завантаженні мапи;
- `willStartRenderingMap` - індикатор початку рендерингу мапи;
- `didFinishRenderingMap` - вказує на завершення рендерингу мапи;
- `didAddAnnotationViews` - обгортка для події додавання відображень анотацій на мапі;
- `didSelectAnnotationView` - обробляє вибір відображення анотації;
- `didDeselectAnnotationView` - обробляє відміну вибору відображення анотації;
- `willStartLocatingUser` - індикатор початку відслідковування місцезнаходження користувача;
- `didStopLocatingUser` - індикатор закінчення відслідковування місцезнаходження користувача;
- `didUpdateUserLocation` - оновлює інформацію про місцезнаходження користувача;
- `didFailToLocateUser` - обгортка для обробки помилок при визначенні місцезнаходження користувача;
- `dragStateOfAnnotationView` - стан перетягування відображення анотації;
- `didAddRenderers` - обгортка для події додавання рендерів на мапу.

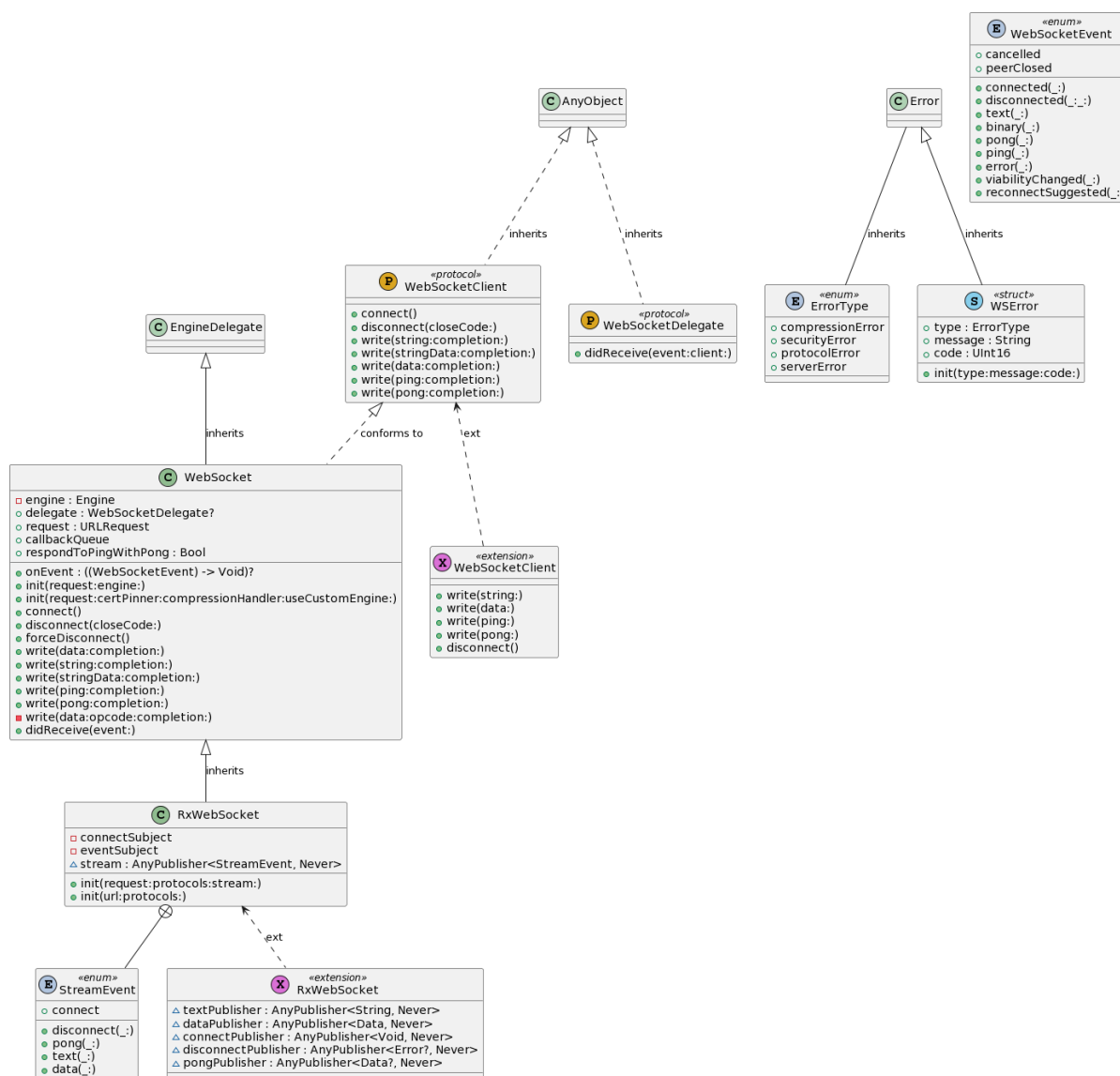


Рисунок 3.9 – UML діаграма класів для реактивної роботи з WebSocket

На рисунку 3.9 продемонстровано основні класи для роботи з WebSocket. Основні змінні за функції для реактивної роботи представлені у класі RxWebSocket. Він забезпечує реактивний інтерфейс для роботи з вебсокетами, даючи змогу підписуватися на різні типи подій (текстові повідомлення, дані, стан під'єднання тощо) у зручному для Combine стилі. Це робить його потужним інструментом для реалізації вебсокет-клієнтів у застосунках, що використовують архітектурні патерни, які ґрунтуються на реактивному програмуванні.

Клас `RxWebSocket`, що розширює `WebSocket` (з бібліотеки `Starscream`), є реалізацією вебсокета з використанням парадигми реактивного програмування в стилі `Combine`.

Тип `StreamEvent`, описує різні події, пов'язані з вебсокетом:

`connect`: Подія підключення до вебсокета.

- `disconnect(Error?)`: Подія відключення від вебсокета з можливою помилкою;
- `pong(Data?)`: Подія отримання `pong`-повідомлення від сервера, яке може містити дані;
- `text(String)`: Подія отримання текстового повідомлення;
- `data(Data)`: Подія отримання даних (не тексту)

`RxWebSocket` містить дві основні властивості для керування потоками подій:

- а) `connectSubject`: `CurrentValueSubject`, який зберігає і випускає останню подію підключення/відключення;
- б) `eventSubject`: `PassthroughSubject`, який випускає текстові, дані, і `pong`-події.

В ініціалізаторі класу `RxWebSocket` налаштовуються обробники подій вебсокета:

- `onConnect`: У разі підключення вебсокета випускається подія `connect`;
- `onText`: При отриманні текстового повідомлення випускається подія `text`;
- `onData`: При отриманні даних випускається подія `data`;
- `onPong`: У разі отримання `pong`-відповіді випускається подія `pong`;
- `onDisconnect`: У разі вимкнення вебсокета випускається подія `disconnect`;
- Потік `stream`.

Властивість `stream` являє собою об'єднання `connectSubject` і `eventSubject` і надає уніфікований потік усіх подій, що відбуваються у вебсокеті.

У розширенні `RxWebSocket` визначено спеціалізовані `Publisher Combine`:

- `textPublisher`: Фільтрує `stream` і витягує тільки текстові повідомлення;
- `dataPublisher`: Фільтрує `stream` і витягує тільки події з даними;
- `connectPublisher`: Фільтрує `stream` і випускає подію під час під'єднання до вебсокета;
- `disconnectPublisher`: Фільтрує `stream` і випускає подію під час відключення від вебсокета, включно з можливою помилкою;
- `pongPublisher`: Фільтрує `stream` і випускає `pong`-події.

Висновки до розділу

У цьому розділі ми розглянули важливий етап розроблення програмного забезпечення, пов'язаний зі створенням реактивних обгорт для різних класів, включно з `NEVPNManager`, `Packet Tunnel Provider` та інтеграцією з `MapKit`, з використанням фреймворка `Combine`. Реактивне програмування стає дедалі актуальнішим у світі розроблення, даючи змогу створювати більш чуйні та ефективні застосунки, здатні ефективно керувати асинхронними подіями та потоками даних.

У розділі було розроблено реактивні обгортки для кожного з обраних класів. Для `NEVPNManager` і `Packet Tunnel Provider` було створено методи, що дають змогу асинхронно керувати VPN-сервісами, а для `MapKit` створили обгортку, яка дає змогу відстежувати місце розташування на карті та взаємодіяти з нею реактивно.

Використання фреймворку Combine дає змогу спростити обробку асинхронних операцій, управління даними та реагування на зміни в реальному часі. Розроблені реактивні обгортки можуть бути легко інтегровані в проєкти, що робить їх більш чуйними, гнучкими і зручними для розробників.

Крім того, у розділі було зазначено, що створено демонстраційний застосунок, який дасть змогу наочно продемонструвати роботу реактивного інструменту на платформах iOS і MacOS. Цей застосунок міститиме в собі екрани для роботи з VPN, картою і виміром швидкості інтернету, забезпечуючи візуальне уявлення про переваги реактивного програмування.

Загалом, процес розроблення програмного забезпечення з використанням реактивних обгортки і фреймворка Combine збагатить функціональність вашого застосунку, зробить його більш чуйним і забезпечить більш гнучкий і зручний інтерфейс для взаємодії з різними компонентами системи.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ІНСТРУМЕНТУ

4.1 Створення та налаштування бібліотеки CocoaPods

CocoaPods - це менеджер залежностей для проектів Swift і Objective-C, який спрощує процес інтеграції зовнішніх бібліотек і фреймворків у розробку iOS і macOS. Він автоматизує процес завантаження, конфігурації та оновлення бібліотек, що значно скорочує час розробки. CocoaPods дозволяє легко визначити залежності проекту в одному файлі (Podfile), після чого він автоматично встановлює та налаштовує їх. Це забезпечує кращий контроль версій, уникнення конфліктів залежностей та легке впровадження сторонніх рішень, роблячи процес розробки більш ефективним і структурованим.

Створення бібліотеки CocoaPods можна поділити на такі етапи:

- а) налаштування проекту Xcode та необхідних targets;
- б) розміщення на Github;
- в) імплементація бібліотеки;
- г) написання тестів;
- д) налаштування Travis CI та Codecov;
- е) публікація pod.

В попередніх розділах було описано процес створення інструменту-бібліотеки. Юніт-тестування дуже важливе для забезпечення надійності вашої бібліотеки. Так як вона розроблена на основі реактивних потоків, її не так просто покрити тестами стандартними бібліотеками XCTest або Quick. Тому для зручності буде використовуватись пакет Testable Combine Publishers. Він дозволяє спростити написання модульних тестів для Combine Publishers за допомогою методу `.expect(...)` для ланцюгів Publisher Subject. Результируючий тип `PublisherExpectation` збирає різні очікування, а потім за допомогою виклику `.waitForExpectations(timeout: 1)` надає можливість після вказаного таймауту перевірити результат.

Під капотом `PublisherExpectation` використовує стандартні API фреймворку `XCTest` і пересилає результати перевірки у відповідні рядки коду, які оголошували очікування. Це дозволяє швидко побачити, яке саме очікування в ланцюжку очікувань не спрацювало в юніт-тестах, як у `Xcode`, так і у виведенні консолі.

Наступним кроком буде налаштування CI/CD за допомогою `Travis CI` та `Codecov`. Щоразу, коли будуть викладені нові зміни на `Github`, ми повинні протестувати наш `pod` і переконатися, що всі тести пройдено успішно.

`Travis CI` - це хостинговий розподілений сервіс безперервної інтеграції, який використовується для створення та тестування програмних проєктів, розміщених на `GitHub`.

Він буде використовуватись для запуску тестових кейсів та надсилання звіту про покриття коду до `Codecov`.

`Codecov` - це інструмент для створення звітів, який призначений для обробки будь-якого формату звіту про покриття у формат, який є стандартним для `Codecov`.

```

1 language: objective-c
2
3 osx_image: xcode14.1
4 env:
5   matrix:
6     - TEST_SDK=iphonesimulator12.1 OS=16.1 NAME='iPhone 14'
7     - TEST_SDK=iphonesimulator12.1 OS=16.1 NAME='iPhone 14 Pro'
8
9   script:
10    - set -o pipefail && xcodebuild test -enableCodeCoverage YES -project
      Combine_Rx_Helper.xcodeproj -scheme Combine_Rx_Helper -sdk $TEST_SDK -destination
      "platform=iOS Simulator,OS=$OS,name=$NAME" ONLY_ACTIVE_ARCH=YES
11

```

Рисунок 4.1 – Налаштування для `Travis CI`

Останнім кроком буде публікація `pod` для того, щоб користувачі змогли встановити бібліотеку за допомогою `CocoaPods`.

```

1 Pod::Spec.new do |spec|
2
3   spec.name           = "Combine-Rx-Helper"
4   spec.version        = "0.0.1"
5   spec.summary        = "A CocoaPods library written in Swift"
6
7   spec.description    = <<-DESC
8   This CocoaPods library helps you perform calculation.
9   DESC
10
11   spec.homepage       = "https://github.com/PashaZherdiy/Combine-Rx-Helper"
12   spec.license        = { :type => "MIT", :file => "LICENSE" }
13   spec.author         = { "PashaZherdiy" => "pacha120431@gmail.com" }
14
15   spec.ios.deployment_target = "16.1"
16   spec.swift_version = "5.2"
17
18   spec.source         = { :git => "https://github.com/PashaZherdiy/Combine-Rx-Helper", :tag =>
19     "#{spec.version}" }
20   spec.source_files   = "Combine-Rx-Helper/**/*.{h,m,swift}"
21 end

```

Рисунок 4.2 – Налаштування CocoaPods файлу .podspec

Файл .podspec визначає метадані для CocoaPods бібліотеки. Ось що представляє кожне поле:

- spec.name: Назва CocoaPods бібліотеки;
- spec.version: Версія бібліотеки;
- spec.summary: Короткий опис бібліотеки;
- spec.description: Докладний опис, який може бути багаторядковим.
- spec.homepage: URL домашньої сторінки проекту, зазвичай це репозиторій GitHub;
- spec.license: Інформація про ліцензію бібліотеки, включаючи тип і розташування файлу;
- spec.author: Ім'я та контактні дані автора(ів) бібліотеки;
- spec.ios.deployment_target: Мінімальна версія iOS, яку підтримує бібліотека;
- spec.swift_version: Версія Swift, з якою сумісна бібліотека;
- spec.source: Джерело, звідки можна завантажити бібліотеку, зазвичай Git репозиторій;

- `spec.source_files`: Шляхи та типи файлів, що включені до бібліотеки, зазвичай файли Swift, Objective-C, та заголовочні файли.

4.2 Налаштування роботи бібліотеки з Swift Package Manager

CocoaPods не єдиний спосіб встановлення бібліотек, зараз багато сучасних проектів використовують в розроці менеджер пакетів Swift Package Manager.

Swift Package Manager (SPM) - це інструмент для управління залежностями та збирання проектів Swift, який автоматизує завантаження, компіляцію та лінування зовнішніх бібліотек. Він вбудований прямо в Swift і дозволяє визначати залежності через простий `Package.swift` файл, що робить його ідеальним для модульної розробки. SPM підтримує використання Git для версіонування та як основного джерела для залежностей, не вимагаючи реєстрації або використання централізованого репозиторію, на відміну від CocoaPods. SPM полегшує роботу зі збиранням і тестуванням бібліотек, забезпечуючи високу інтеграцію з Xcode, що робить його зручним варіантом для розробників на Swift.

```

1 // swift-tools-version: 5.9
2 // The swift-tools-version declares the minimum version of Swift required to build this package.
3
4 let package = Package(
5     name: "Combine Rx Helper",
6     products: [
7         // Products define the executables and libraries a package produces, making them visible to
8         // other packages.
9         .library(
10             name: "Combine Rx Helper",
11             targets: ["Combine Rx Helper"]),
12     ],
13     targets: [
14         // Targets are the basic building blocks of a package, defining a module or a test suite.
15         // Targets can depend on other targets in this package and products from dependencies.
16         .target(
17             name: "Combine Rx Helper"),
18         .testTarget(
19             name: "Combine Rx HelperTests",
20             dependencies: ["Combine Rx Helper"]),
21     ]
22 )

```

Рисунок 4.3 – Конфігурація Swift Package Manager

Файл `Package.swift` в `Swift Package Manager` визначає конфігурацію пакета. У ньому описано ім'я пакета, продукти, які він генерує, та цілі (targets), які включають кодову базу. Продукти можуть бути бібліотеками, які можуть використовуватися іншими пакетами. Цілі визначають модулі або тестові набори, і можуть мати залежності від інших цілей у пакеті або від продуктів, які надаються залежностями.

4.3 Налаштування роботи бібліотеки з Carthage

Також багато розробників використовують замість `CocoaPods` та `Swift Package Manager` менеджер залежностей `Carthage`.

`Carthage` - це децентралізований менеджер залежностей для проектів на мові програмування `Swift`. Він дозволяє розробникам збудувати свої залежності та вирішити, як інтегрувати їх у проект. Переваги `Carthage` включають простоту використання та менш інвазивний вплив на проекти, оскільки він не створює і не змінює `Xcode` проекти за вас. Однак, це також може бути недоліком, оскільки розробникам потрібно вручну включати зібрані бінарники в свої `Xcode` проекти. `Carthage` використовують для того, щоб мати більше контролю над процесом збирання залежностей.

`Carthage` відрізняється від `CocoaPods`, оскільки не залежить від централізованого репозиторія і не вносить зміни в проект `Xcode`, забезпечуючи більше гнучкості. Він скачує код напряму з `GitHub`, лише збирає бібліотеки і залишає інтеграцію на розсуд розробника. У той час як `Carthage` підтримує лише динамічне лінування, `CocoaPods` пропонує як статичне, так і динамічне лінування фреймворків. `Carthage` можна використовувати разом із `Swift Package Manager`, але необхідно звертати увагу на типи бібліотек та їх сумісність.

Щоб бібліотеку можна було встановити за допомогою `Carthage`, потрібно:

- зробити бібліотеку загальнодоступною на GitHub;
- у бібліотеці має бути Xcode проект з Shared схемами;
- задокументувати в інструкції для користувачів, як додати бібліотеку в Carthage файл;
- опублікувати pre-built framework через GitHub Releases.

Висновки до розділу

У цьому розділі було описано процес розгортання розробленого інструменту. Бібліотека підтримує усі сучасні менеджери керування залежностями, такі як: CocoaPods, Carthage та Swift Package Manager.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка інструменту для реактивної роботи з мережею на платформах iOS та MacOS	ІТ-компанії	Написання мобільних та десктопних застосунків з використанням реактивного підходу буде зручнішим
	ІТ-спільнота	

Таблиця 5.2 — Опис ідеї стартап-проєкту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Alamofire	Opollo	Moya			
1	Можливість інтеграції з різними мережевими сервісами та API	+	+	-	+			+
2	Підтримка асинхронного запуску мережових запитів	+	+	+	+			+

Продовження таблиці 5.2

3	Вбудовані засоби для моніторингу та діагностики мережевої активності	-	-	-	-	+		
4	Гнучке керування кешуванням даних для ефективного мережевого взаємодії	-	-	-	-	+		
5	Інструменти для безпеки даних при мережевих запитах	+	-	-	-			+
6	Автоматизація тестування мережевих викликів і обробки відповідей	+	-	-	+		+	

На основі аналізу можливостей та перспектив ринку, мій стартап-проект інструменту для реактивної роботи з мережею на платформах iOS та MacOS демонструє сильні сторони та високий потенціал конкурентоспроможності. Забезпечуючи відповідність до вимог сучасного розробництва програмного забезпечення та впровадження інноваційних технологій, проєкт володіє усіма необхідними функціями для ефективної реактивної взаємодії з мережевими сервісами.

5.2 Технологічний аудит ідеї проєкту

Таблиця 5.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка інструменту для реактивної роботи з мережею	XCode, Swift, AppKit, UIKit	Наявна	Доступна
2	Розробка застосунку на основі інструменту	XCode, Swift, AppKit, UIKit	Наявна	Доступна

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 5.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн./ум.од	Невідома
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі або по ринку, %	Невідомо

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Інструмент для реактивної роботи з мережею на платформах iOS та MacOS	ІТ-компанії	Відсутні	Зручність, мультиплатформеність
2		ІТ-спільнота	Відсутні	

Таблиця 5.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність конкурентів, котрі надають схожі рішення	Зменшення ціни на поставлену послугу; Розробка унікальних характеристик товару; Надання ліцензій на обслуговування
2	Кошти на розробку та підтримку продукту	Закінчення грошей та недостатнє фінансування	Залучення додаткових інвесторів, мотивація роботи на перспективу; Ітеративна розробка продукту задля покрокового виведення продукту на ринок та отримання відповіді користувачів

Таблиця 5.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Вихід на ринок, Зменшення монополії, Надання нових рішень у сфері	Розробка нової функціональності; Вихід нової продукції на ринок; Надання різноманітних типів ліцензій в залежності від потреб користувача \ замовника.
2	Вихід аналогу	Надати продукт з певними характеристиками та можливостями що відсутні у компаній конкурентів	Аналіз ринку та користувачів задля задоволення їх потреб та надання функціональності у найкоротші строки за ціну, котра є дешевшою ніж у продуктів-замінників.
3	Зворотній зв'язок від користувачів	Можливість отримання необхідної інформації для вдосконалення продукту	Наявність вхідних даних та реакція на них з боку команди розробників задля задоволення потреб та бажань кінцевих користувачів системи кешування даних.
4	Грошова винагорода за рекламу	При достатньому попиту на систему кешування даних можлива комерціалізація продукту на основі реклами задля отримання грошової винагороди для подальшого розвитку продукту та оплати заробітної плати працівникам	Точкова комерціалізація продукту; Введення реклами; Ведення додаткових коштів у проєкт задля його подальшого розвитку.

Таблиця 5.9 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична	Товар від кожної компанії на ринку, являється недосконалим заміником товару, реалізованого іншими фірмами; На ринку є умови для входу та виходу; Ціна корелює між суперниками;	Розробка продукту з характеристиками, які покривають сфери вживання що не покривають інші товари-замінники; Кореляція цін у відповідності до товарів заміників; Різні типи ліцензій.
2	Рівень конкурентної боротьби: світовий	Всі продукти заміники розроблялись інтернаціональними командами з різних куточків світу, продукти не належать до певної держави, а належать команді розробників	Вихід на ринок збуту продукту з клієнто-необхідною функціональністю; Налагодження маркетингу на основних Інтернет ресурсах задля охоплення великої кількості потенційних користувачів; Надання бета-версій продукту.
3	Галузева ознака: внутрішньогалузева	Даний тип продукту може використовуватися тільки у сфері розробки ІТ додатків \ продуктів	Надання зручного, інтуїтивно зрозумілого інтерфейсу; Підтримка всім відомих методів взаємодії з середовищем розробки; Наявність документації та он-лайн підтримки.

Продовження таблиці 5.9

4	Конкуренція за видами товарів: товарно-видова	Дана конкуренція – конкуренція між товарами одного виду.	Впровадження функціональності яка відсутня у товарів-замінників; Спрощення інтерфейсів; Надання підтримки.
5	Характер конкурентних переваг: цінова та не цінова	Цінові переваги – точкова комерціалізація; Не цінова – надання функціональності, що відсутня у товарах-замінниках.	Надання платних ліцензій лише на критично важливу функціональність для клієнта з певним строком підтримки, що зазначена у відповідній ліцензії; Впровадження унікальної функціональності.
6	За інтенсивністю: марочна	Наявність унікального знаку що відрізняє даний продукт продуктів-замінників	Впровадження власної назви та власного знаку.

Таблиця 5.10 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Alamofire, Apollo, Moya	Невідомо	Набір функціоналу	ІТ компанії	Відсутні
Висновки	Існують	Дані відсутні	Чим більше функціоналу пропонує постачальник, тим краще рішення	Клієнти хочуть отримати максимально зручне та ефективне рішення	Відсутні

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок: оскільки кожний з існуючих продуктів не впливає у великій мірі на поточну ситуацію на ринку в цілому, кожний з існуючих продуктів має свою специфічну сферу використання та свої позитивні та негативні сторони щодо рішення певних типів задач, то робота та вихід на даний ринок є можливою і реалізованою задачею [22].

Для виходу на ринок продукт повинен мати функціонал що відсутній у продуктів-аналогів, повинен задовольняти потреби користувачів, мати необхідний та достатній функціонал з конфігурування, підтримку зі сторони розробників та можливість розробки спеціального функціоналу за відповідною ліцензією.

Таблиця 5.11 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Швидкість та ефективність обробки	Інструмент забезпечує швидку обробку мережових запитів, що підвищує продуктивність розробників
2	Сумісність з багатьма платформами	Розроблено так, щоб працювати на iOS та MacOS, забезпечуючи широку сумісність
3	Інтеграція з сучасними фреймворками	Підтримка популярних фреймворків підвищує гнучкість використання
4	Простота використання	Інтуїтивно зрозумілий інтерфейс спрощує навчання та використання
5	Розширені можливості тестування та дебагінгу	Включення інструментів для діагностики допомагає виявити та вирішити проблеми швидше
6	Висока масштабованість	Здатність обробляти зростаючий об'єм даних та запитів без втрати продуктивності
7	Безпека та конфіденційність	Вбудовані функції безпеки захищають від зовнішніх загроз та витоку даних

Продовження таблиці 5.11

8	Підтримка спільноти та оновлення	Активна спільнота та регулярні оновлення забезпечують тривалу актуальність інструменту
---	----------------------------------	--

Таблиця 5.12 — Порівняльний аналіз сильних та слабких сторін стартап-проекту

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Швидкість та ефективність обробки	15					+		
2	Сумісність з багатьма платформами	18				+			
3	Інтеграція з сучасними фреймворками	18					+		
4	Простота використання	16						+	
5	Розширені можливості тестування та дебагінгу	14				+			
6	Висока масштабованість	13					+		
7	Безпека та конфіденційність	14					+		
8	Підтримка спільноти та оновлення	5							+

Таблиця 5.13 — SWOT аналіз стартап-проєкту

Сильні сторони (S): – Інноваційність та актуальність технології для сучасного ринку. – Висока адаптивність до потреб розробників на платформах Apple. – Можливість інтеграції з широким спектром API та сервісів.	Слабкі сторони (W): – Обмеження ресурсів на ранньому етапі розробки. – Висока залежність від великих оновлень платформ iOS та MacOS. – Потенційні складнощі з управлінням великою кількістю мережових запитів.
Можливості (O): – Розширення ринку за рахунок збільшення популярності мобільних та десктопних додатків. – Партнерство з іншими розробниками програмного забезпечення. – Внесення інновацій у сферу мережової безпеки та приватності.	Загрози (T): – Стрімке оновлення технологій та поява нових конкурентів. – Зміни в політиці конфіденційності та безпеки, що можуть вплинути на доступність API. – Ризики, пов'язані з захистом інтелектуальної власності в рамках розробки ПЗ.

Таблиця 5.14 — Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, даний ресурс - наявний	2-3 місяці

Продовження таблиці 5.14

2	Реклама	Залучення власних коштів для реклами товару	1-2 місяці
3	Написання статей та опис товару на відомих ресурсах	Головний ресурс – час, даний ресурс – наявний	2-3 тижні
4	Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	1-3 місяці

5.4 Розроблення ринкової стратегії проєкту

Для розробки ринкової стратегії, потрібно визначити охоплення ринку за допомогою визначення груп потенційних споживачів. В таблиці 5.15 оглянути вибір цільових груп потенційних користувачів.

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є працівники ІТ сфери, ІТ компанії в цілому та будь-які підприємства котрі використовують програмні продукти побудовані на мові програмування Swift, та використовують реактивний підхід у розробці застосунків. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки для підприємств, ІТ працівників та ІТ компаній у цілому надається стандартизований продукт з можливістю розширення функціональності за домовленістю (відповідно до ліцензії).

Таблиця 5.15 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	ІТ компанії, що розробляють мобільні та десктопні застосунки під iOS та MacOS	Висока готовність	Високий попит	Середня конкуренція	Легко
2	ІТ спільнота	Висока готовність	Високий попит	Середня конкуренція	Легко
Які цільові групи обрано: ІТ компанії, ІТ спільнота					

Таблиця 5.16 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності, що відсутня у товарів-замінників, підтримка клієнтів	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали, контакт напряду з споживачами; формування лояльності і прихильності споживачів	Зниження ступеню заміності товару; Прихильність клієнтів; Відмітні властивості товару; Відмітні характеристики товару;	Стратегія диференціації

Таблиця 5.17 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні, оскільки є товари-замінники, але дані товари-замінники не мають деякого необхідного функціоналу	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики товару конкурента, основна ціль компанії розробка нового унікального функціоналу, з підтримкою основного функціоналу конкурентів	Стратегія заняття конкурентної ніші

Таблиця 5.18 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Висока продуктивність та низьке споживання ресурсів	Фокус на розробці високопродуктивного інструменту, що відповідає сучасним трендам і технологічним стандартам.	Реактивна архітектура, що забезпечує швидку відповідь на мережеві події.	Надійність, безпека та інноваційність.

Продовження таблиці 5.18

2	Гнучкість у налаштуванні для задоволення специфічних потреб розробників	Надання інтегрованих рішень для підвищення ефективності розробки та тестування.	Інтуїтивний інтерфейс та легкість використання.	Стійкість до змін у технологічному ландшафті та висока масштабованість.
---	---	---	---	---

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття конкурентної ніші.

5.5 Розроблення маркетингової програми стартап-проєкту

Першим кроком до розроблення маркетингової програми стартап-проєкту є формування маркетингової концепції товару, який отримує споживач. Для цього у таблиці 5.19 підсумуємо результати попереднього аналізу конкурентоспроможності товару

Таблиця 5.19 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Ефективна реактивна обробка мережевих запитів	Зниження часу відгуку додатків на мережеві події	Новітні реактивні підходи, які перевищують традиційні блокуючі I/O моделі
2	Інтуїтивні інструменти для розробки на iOS та MacOS	Покращення досвіду розробки завдяки інтегрованим інструментам дебагінгу та тестування	Гнучкість та масштабованість, які дозволяють адаптуватися до змінних вимог сучасного ринку програмного забезпечення

Таблиця 5.20 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1. Товар за задумом	Інструмент для реактивної роботи з мережею на платформах iOS та MacOS		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Якість	Нм	Тл
	Масштабованість	М	Тл
	Зрозумілість	М	Е
	Швидкість роботи	Нм	Тх
	Універсальність	М	Тх
	Спосіб взаємодії: додаток у мережі інтернет		
	Якість: ISO 19649:2017		
3. Товар із підкріпленням	Ознайомлення клієнта з усіма сутностями та функціоналу проєкта для зрозумілого користування		

Таблиця 5.21 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
0\$	0\$	ІТ компанії від 100 000\$ за рік	Не визначена
		ІТ спільнота від 1000\$ за рік	

Таблиця 5.22 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Задоволення потреб людей, що розробляють мобільні та десктопні застосунки	Надати можливість встановити інструмент та імпортувати в проект	Від виробника до споживача	Прямий канал збуту

Таблиця 5.23 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Професійні розробники, які шукають ефективні рішення для оптимізації реактивної роботи з мережею.	Інтернет, електронна пошта	Надійність, продуктивність та інноваційність як головні властивості продукту.	Підкреслити унікальність продукту та його переваги у вирішенні специфічних задач розробників.	Створення образу інструменту, який є необхідним помічником у повсякденній роботі розробника, з акцентом на інноваційність та ефективність.

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

Висновки до розділу

В п'ятому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку, як висновок було вказано що існує можливість ринкової комерціалізації проєкту. Розглянувши потенційні групи клієнтів, бар'єри входження, стан конкуренції та конкурентоспроможність проєкту було встановлено що проєкт є перспективним. Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

ВИСНОВКИ

Магістерська дисертація присвячена застосуванню реактивного підходу у використанні мережі при розробці мобільних та десктопних застосунків. Аналіз показав, що є потреба в забезпеченні адаптивності та ефективності роботи застосунків під час взаємодії з мережею. Даний факт обумовлює актуальність теми роботи.

Метою роботи є підвищення зручності роботи з мережевими запитами, асинхронними операціями та управлінням потоками даних.

У роботі описано процес розробки інструменту на мові програмування Swift, ґрунтованого на принципах реактивного програмування, для ефективної роботи з мережею. Цей інструмент надає реактивне рішення, яке дає змогу розробникам створювати гнучкі, адаптивні та ефективні мобільні застосунки.

В ході виконання роботи проведено аналіз існуючих рішень реактивної роботи з мережею для роботи з мобільними платформами. Результати аналізу показали, що аналоги мають недоліки, такі як: підтримка не усіх популярних платформ та середовищ розробки, вузький функціонал, не достатній для повноцінної розробки.

З метою усунення цих недоліків був розроблений мультиплатформений інструмент, що покриває усі потреби розробників під час роботи з мережею. Розроблений інструмент представляє собою бібліотеку, яка підтримує усі сучасні менеджери керування залежностями, такі як: CocoaPods, Carthage та Swift Package Manager.

Проведений маркетинговий аналіз стартап-проєкту. Проведено опис ідеї проєкту, технологічний аудит ідеї проєкту, аналіз ринкових можливостей запуску стартап-проєкту. Розроблено ринкову стратегію проєкту та маркетингову програму стартап-проєкту.

Наукова новизна: набув подальшого розвитку реактивний підхід для забезпечення адаптивності та ефективності роботи застосунків під час взаємодії з мережею.

Практичне значення отриманих результатів полягає в тому, що розробникам мобільних та десктопних застосунків на платформах iOS та MacOS надано зручний інструмент для роботи з мережевими запитами, асинхронними операціями та управлінням потоками даних.

Результати роботи пройшли апробацію на науково-практичній конференції «SoftTech-2023» та опубліковані у матеріалах конференції.

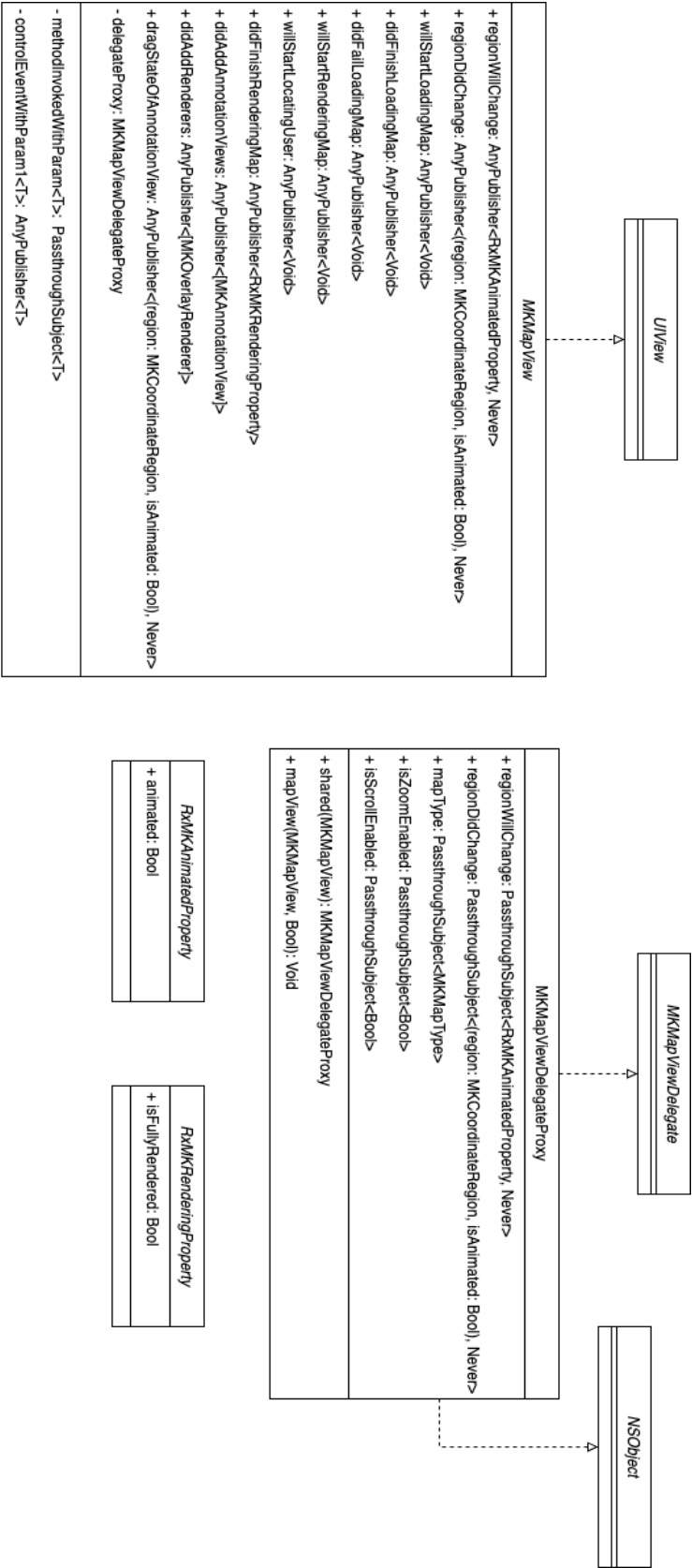
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Example of Clean Architecture of iOS app using RxSwift [Електронний ресурс] – Режим доступу до ресурсу: <https://iosexample.com/example-of-clean-architecture-of-ios-app-using-rxswift/>
- 2) Cocoapods [Електронний ресурс] – Режим доступу до ресурсу: <https://cocoapods.org/>
- 3) Swift Package Manager [Електронний ресурс] – Режим доступу до ресурсу: <https://www.swift.org/package-manager/>
- 4) Carthage [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/Carthage/Carthage>
- 5) Alamofire [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/Alamofire/Alamofire>
- 6) MVVM in iOS [Електронний ресурс] – Режим доступу до ресурсу: <https://cheesecakelabs.com/blog/mvvm-in-ios/>
- 7) An Intro to Combine and the Reactive Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/capital-one-tech/ios-bites-an-intro-to-combine-and-the-reactive-pattern-24da924a6863>
- 8) UIKit [Електронний ресурс] – Режим доступу до ресурсу: <https://evergreens.com.ua/ru/articles/ui-kit.html>
- 9) Why RxSwift Is Worth Using [Електронний ресурс] – Режим доступу до ресурсу: <https://intersog.com/blog/is-rx-still-relevant-in-modern-mobile-ios-development-today/>
- 10) Building a command line tool using a Swift Package Manager [Електронний ресурс] – Режим доступу: <https://www.swiftbysundell.com/articles/building-a-command-line-tool-using-the-swift-package-manager/>
- 11) MapKit [Електронний ресурс] – Режим доступу: <https://developer.apple.com/documentation/mapkit/>

- 12) Shabalin, D. (2019). Combine: Asynchronous Programming with Swift. Ray Wenderlich Media. 1-384.
- 13) Young, M. (2020). Reactive Programming in Swift. O'Reilly Media. 1-350.
- 14) Farag, M. (2019). Learning RxSwift: Reactive Programming. Packt Publishing. 1-178
- 15) Duncan DeVore, Sean Walsh , Brian Hanafee (2018) Reactive Application Development. 121-128
- 16) Jesse Feiler (2018), Beginning Reactive Programming with Swift: Using RxSwift, Amazon Web Services, and JSON with iOS and macOS. 135-140
- 17) Matt Neuburg (2021), iOS 15 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics. 204-250
- 18) Tjeerd in 't Veen (2019), Swift in Depth. 351-390
- 19) Wallace Wang (2023), Pro iPhone Development with SwiftUI: Design and Manage Top-Quality Apps 4th ed. Edition. 310-430
- 20) Jon Hoffman (2019), Mastering Swift 5. 110-112
- 21) David Kopec (2018), Classic Computer Science Problems in Swift. 140
- 22) Розроблення стартап-проєкту: Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ», 2016. – 28 с.

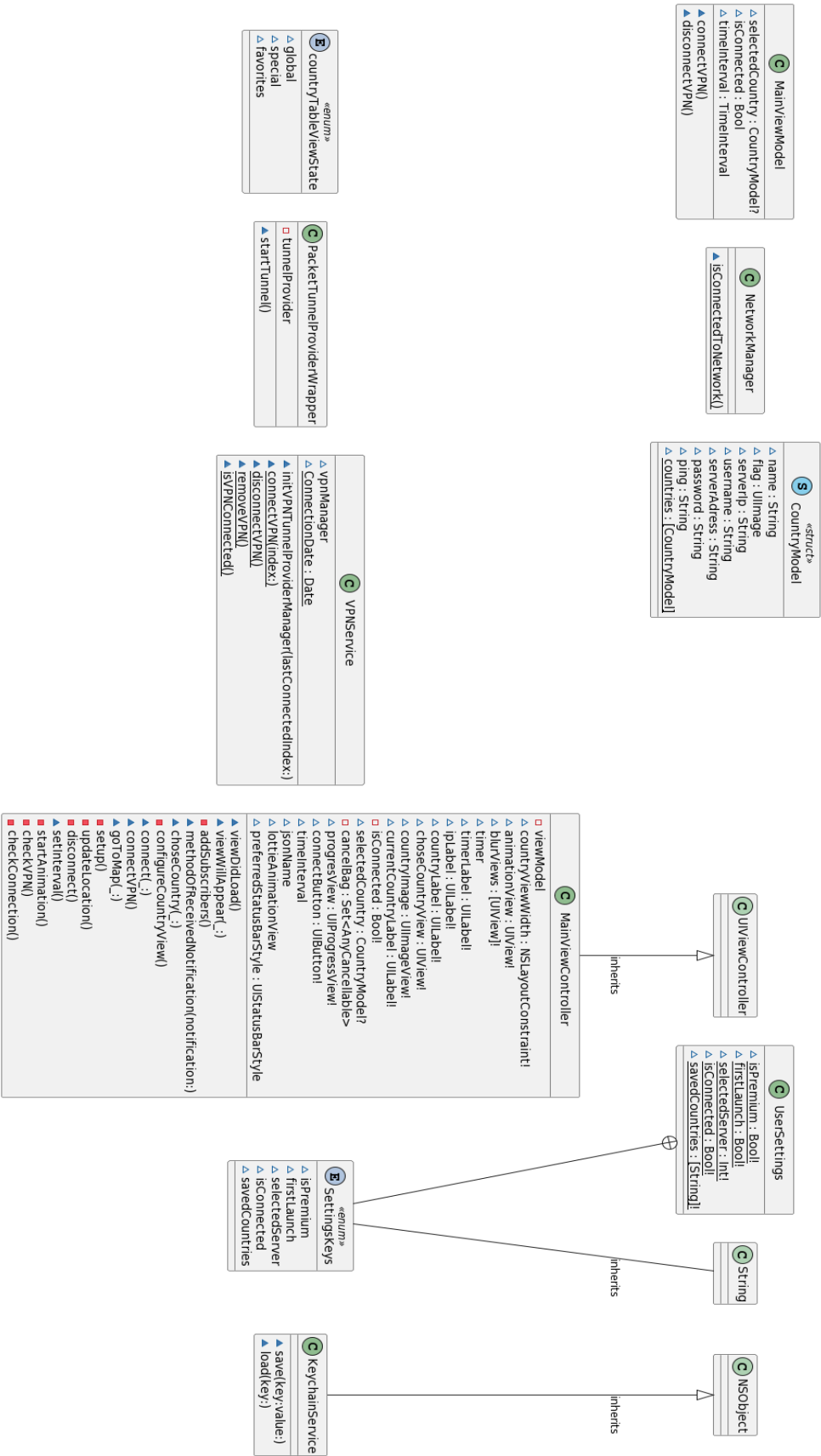
ДОДАТОК А

Діаграма класів бібліотеки



ДОДАТОК Б

Діаграма класів застосунку



ДОДАТОК В

Лістинг програмного коду

MainViewController.swift

```

class MainViewController: UIViewController {

    private var viewModel = MainViewModel()

    @IBOutlet weak var countryViewWidth: NSLayoutConstraint!

    @IBOutlet var animationView: UIView!
    @IBOutlet var blurViews: [UIView]!
    var timer = Timer()

    @IBOutlet weak var timerLabel: UILabel!
    @IBOutlet weak var ipLabel: UILabel!
    @IBOutlet weak var countryLabel: UILabel!

    @IBOutlet weak var choseCountryView: UIView!
    @IBOutlet weak var countryImage: UIImageView!
    @IBOutlet weak var currentCountryLabel: UILabel!

    private var isConnected: Bool! = false

    var selectedCountry: CountryModel? = CountryModel.countries[0]

    private var cancelBag: Set<AnyCancellable> = []

    @IBOutlet weak var progresView: UIProgressView!
    @IBOutlet weak var connectButton: UIButton!

    var timeInterval = TimeInterval() {
        didSet {
            timerLabel.text = timeInterval.stringTime
        }
    }

    let jsonName = "Processing Circle"
    var lottieAnimationView = LottieAnimationView()

    override var preferredStatusBarStyle: UIStatusBarStyle {
        return .lightContent
    }
}

```

```

}

override func viewDidLoad() {
    super.viewDidLoad()
    let animation = LottieAnimation.named(jsonName)
    lottieAnimationView = LottieAnimationView(animation: animation)

    lottieAnimationView.frame = CGRect(x: 0, y: 0, width:
animationView.frame.width, height: animationView.frame.height)
    animationView.addSubview(lottieAnimationView)
    lottieAnimationView.loopMode = .loop

    NotificationCenter.default.addObserver(self, selector:
#selector(self.methodOfReceivedNotification(notification:)), name:
Notification.Name("alert"), object: nil)
    addSubscribers()

    UserSettings.firstLaunch = true
    progresView.isHidden = true
    connectButton.layer.cornerRadius = connectButton.frame.height / 2

    blurViews.forEach {
        $0.layer.cornerRadius = 10
        $0.transform = CGAffineTransform(scaleX: 0.0, y: 0.0)
    }
    if let savedIndex = UserSettings.selectedServer {
        selectedCountry = CountryModel.countries[savedIndex]
    }
    let tapToChoseCountry = UITapGestureRecognizer(target: self, action:
#selector(self.choseCountry(_:)))
    choseCountryView.addGestureRecognizer(tapToChoseCountry)
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    configureCountryView()
    choseCountryView.layoutIfNeeded()
    countryViewWidth.constant = countryLabel.frame.width + 100
    checkConnection()
}

private func addSubscribers() {

```

```

NotificationCenter
    .Publisher(center: .default,
               name: UIApplication.willEnterForegroundNotification)
    .sink { [weak self] _ in
        self?.checkVPN()
    }
    .store(in: &cancelBag)
}

@objc func methodOfReceivedNotification(notification: Notification) {
    let vc = UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: "AlertVC") as!
AllowAlertViewController
    self.addChild(vc)
    vc.view.frame = self.view.frame
    self.view.addSubview(vc.view)
    vc.didMove(toParent: self)
}

@objc func choseCountry(_ sender: UITapGestureRecognizer) {
    let vc = storyboard?.instantiateViewController(withIdentifier:
"CountryVC") as! CountriesViewController
    if !UserSettings.isPremium {
        vc.selectedIndex = 0
    } else {
        vc.selectedIndex = CountryModel.countries.firstIndex(where: { $0.name
== selectedCountry?.name ?? nil })
    }
    navigationController?.pushViewController(vc, animated: true)
}

private func configureCountryView() {
    let tapToChoseCountry = UITapGestureRecognizer(target: self, action:
#selector(self.choseCountry(_:)))
    choseCountryView.addGestureRecognizer(tapToChoseCountry)
    choseCountryView.layer.cornerRadius = 10
    countryImage.layer.cornerRadius = 5

    countryImage.image = selectedCountry?.flag
    countryLabel.text = selectedCountry?.name
}

```

```

@IBAction func connect(_ sender: Any) {
    if isConnected {
        disconnect()
    } else {
        connectVPN()
    }
}

func connectVPN() {
    lottieAnimationView.play()
    if let selectedCountry = selectedCountry {
        let index = CountryModel.countries.firstIndex(where: { $0.name ==
selectedCountry.name })
        VPNService.connectVPN(index: index!)
    }
    isConnected = true
    timer = Timer.scheduledTimer(timeInterval: 1, target: self, selector:
#selector(setInterval), userInfo: nil, repeats: true)
    setup()
    startAnimation()
    DispatchQueue.main.async {
        UIView.animate(withDuration: 4) { [self] in
            progresView.setProgress(1, animated: true)
        }
    }

    DispatchQueue.main.asyncAfter(deadline: .now() + 5) { [self] in
        lottieAnimationView.stop()
        DispatchQueue.main.asyncAfter(deadline: .now() + 0.2) { [self] in
            UIView.animate(withDuration: 1) { [self] in
                blurViews.forEach { $0.transform = CGAffineTransform(scaleX:
1.0, y: 1.0) }
            }
            updateLocation()

            UIView.animate(withDuration: 1) { [self] in

                } completion: { [self] _ in
                    connectButton.isHidden = false
                    progresView.isHidden = true
                    connectButton.backgroundColor = UIColor(red: 0.054, green:
0.404, blue: 0.3, alpha: 1)
                    connectButton.setTitle("Disconnect", for: .normal)

```

```

    }

    }
}

@IBAction func goToMap(_ sender: Any) {
    let vc = storyboard?.instantiateViewController(withIdentifier: "MapVC")
as! MapViewController
    vc.selectedIndex = CountryModel.countries.firstIndex(where: { $0.name
== selectedCountry?.name ?? nil })
    navigationController?.pushViewController(vc, animated: true)
}

private func setup() {

    view.bringSubviewToFront(connectButton)

}
private func updateLocation() {
    currentCountryLabel.text = selectedCountry?.name
    ipLabel.text = selectedCountry?.serverIp
}

private func disconnect() {
    VPNService.disconnectVPN()

    isConnected = false
    TimeInterval = 0
    connectButton.backgroundColor = UIColor(red: 0.18, green: 0.788, blue:
0.608, alpha: 1)
    connectButton.setTitle("Connect", for: .normal)
    timer.invalidate()
    UIView.animate(withDuration: 1) { [self] in
        blurViews.forEach { $0.transform = CGAffineTransform(scaleX: 0.0, y:
0.0) }
    }
}

@objc func setInterval() {
    TimeInterval += 1
}

```

```

private func startAnimation() {

}

private func checkVPN() {

    DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) {
        if !VPNService.isVPNConnected() {
            self.disconnect()
        }
    }
}

    timeInterval = Date().timeIntervalSinceReferenceDate -
VPNService.ConnectionDate.timeIntervalSinceReferenceDate
}

private func checkConnection() {
    if !NetworkManager.isConnectedToNetwork() {
        let vc = UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: "NoInternetVC") as!
InternetAlertViewController
        self.addChild(vc)
        vc.view.frame = self.view.frame
        self.view.addSubview(vc.view)
        vc.didMove(toParent: self)
        return
    }
}
}

```

InternetAlertViewController.swift

```

class InternetAlertViewController: UIViewController {

    @IBOutlet weak var tryAgainButton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()

        showAnimate()
    }
}

```

```

override func viewDidLoadLayoutSubviews() {
    super.viewDidLoadLayoutSubviews()
    tryAgainButton.layer.cornerRadius = tryAgainButton.frame.height / 2
}

@IBAction func tryAgain(_ sender: Any) {
    let parentVC = parent as! MainViewController
    removeAnimate()
    if !NetworkManager.isConnectedToNetwork() {
        if !NetworkManager.isConnectedToNetwork() {
            let vc = UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: "NoInternetVC") as!
InternetAlertViewController
            parentVC.addChild(vc)
            vc.view.frame = parentVC.view.frame
            parentVC.view.addSubview(vc.view)
            vc.didMove(toParent: parent)
            return
        }
    }
}

```

```

func showAnimate()
{
    self.view.transform = CGAffineTransform(scaleX: 1.3, y: 1.3)
    self.view.alpha = 0.0
    UIView.animate(withDuration: 0.25, animations: {
        self.view.alpha = 1.0
        self.view.transform = CGAffineTransform(scaleX: 1.0, y: 1.0)
    })
}

```

```

func removeAnimate()
{
    UIView.animate(withDuration: 0.25, animations: {
        self.view.transform = CGAffineTransform(scaleX: 1.3, y: 1.3)
        self.view.alpha = 0.0
    }, completion: {(finished : Bool) in
        if (finished)
        {
            self.view.removeFromSuperview()
        }
    })
}

```

```

    }
  })
}
}

```

MapViewController.swift

```

class MapViewController: UIViewController {

    @IBOutlet weak var countryIP: UILabel!
    @IBOutlet weak var city: UILabel!
    @IBOutlet weak var country: UILabel!

    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var infoView: UIView!
    @IBOutlet weak var beginButton: UIView!

    var selectedIndex: Int? = 1

    private var artworks: [Artwork] = [] {
        didSet {
            mapView.addAnnotations(artworks)
        }
    }

    let initialLocation = CLLocation(latitude: 50.40856715875, longitude:
30.54299045393)
    override func viewDidLoad() {
        super.viewDidLoad()
        mapView.delegate = self
        initMap()
        initInfo()
        self.navigationController?.interactivePopGestureRecognizer?.isEnabled =
false

        if #available(iOS 13.0, *) {
            self.overrideUserInterfaceStyle = .dark
        }
    }

    override var supportedInterfaceOrientations: UIInterfaceOrientationMask {

```

```

    return .portrait
}

```

```

private func LoadData() {
    artworks = [Artwork(title: "Bengaluru", country: "India", coordinate:
CLLocationCoordinate2D(latitude: 12.972442, longitude: 77.580643)),
                Artwork(title: "Vancouver", country: "Canada", coordinate:
CLLocationCoordinate2D(latitude: 49.246292, longitude: -123.116226)),
                Artwork(title: "Frankfurt", country: "Germany", coordinate:
CLLocationCoordinate2D(latitude: 50.110924, longitude: 8.682127)),
                Artwork(title: "Dublin", country: "United Kingdom", coordinate:
CLLocationCoordinate2D(latitude: 53.350140, longitude: -6.266155)),
                Artwork(title: "Amsterdam", country: "Netherlands", coordinate:
CLLocationCoordinate2D(latitude: 52.377956, longitude: 4.897070)),
                Artwork(title: "New York", country: "USA", coordinate:
CLLocationCoordinate2D(latitude: 40.730610, longitude: -73.935242)),
                Artwork(title: "Singapore", country: "Singapore", coordinate:
CLLocationCoordinate2D(latitude: 1.290270, longitude: 103.851959)),]
}

```

```

override func viewWillLayoutSubviews() {
    super.viewDidLayoutSubviews()

    infoView.clipsToBounds = true
    infoView.layer.cornerRadius = 20

    beginButton.layer.cornerRadius = beginButton.frame.height / 2
    beginButton.clipsToBounds = true
}

```

```

func initMap(){

    mapView.centerToLocation(initialLocation)

    let Center = CLLocationCoordinate(latitude: 50.408, longitude: 30.542)
    let region = MKCoordinateRegion(
        center: Center.coordinate,
        latitudinalMeters: 500,
        longitudinalMeters: 600)
    // mapView.setCameraBoundary(
    //     MKMapView.CameraBoundary(coordinateRegion: region),
    //     animated: true)
}

```

```

        //let zoomRange =
        MKMapView.CameraZoomRange(maxCenterCoordinateDistance: 200)
        let zoomRange =
        MKMapView.CameraZoomRange(minCenterCoordinateDistance: 40000000)

        //mapView.setCameraZoomRange(zoomRange, animated: true)

        mapView.setRegion(MKCoordinateRegion(center: Center.coordinate,
        latitudinalMeters: 2000000, longitudinalMeters: 2000000), animated: true)
        LoadData()

        mapView.register(ArtworkView.self,
        forAnnotationViewWithReuseIdentifier:
        MKMapViewDefaultAnnotationViewReuseIdentifier)

        mapView.addAnnotations(artworks)
    }

    func initInfo() {
        countryIP.text = CountryModel.countries[selectedIndex!].serverIp
        country.text = CountryModel.countries[selectedIndex!].name
        city.text = artworks.first(where: { $0.country ==
        CountryModel.countries[selectedIndex!].name })?.title
    }

    @IBAction func goBack(_ sender: Any) {
        navigationController?.popViewController(animated: true)
    }

    @IBAction func beginConnection(_ sender: Any) {
        navigationController?.popViewController(animated: true)
        let vc = navigationController?.topViewController as! MainViewController

        guard let selectedIndex = selectedIndex else {
            return
        }
        UserSettings.selectedServer = selectedIndex
        vc.selectedCountry = CountryModel.countries[selectedIndex]
    }

```

```
}
```

```
private extension MKMapView {
  func centerToLocation(
    _ location: CLLocation,
    regionRadius: CLLocationDistance = 500000
  ) {
    let coordinateRegion = MKCoordinateRegion(
      center: location.coordinate,
      latitudinalMeters: regionRadius,
      longitudinalMeters: regionRadius)
    setRegion(coordinateRegion, animated: true)
  }
}
```

```
extension MapViewController: MKMapViewDelegate {
```

```
  func mapView(_ mapView: MKMapView, annotationView view:
    MKAnnotationView, calloutAccessoryControlTapped control: UIControl) {
```

```
    guard let artwork = view.annotation as? Artwork else {
      return
    }
    let launchOptions = [MKLaunchOptionsDirectionsModeKey:
    MKLaunchOptionsDirectionsModeDriving]
  }
```

```
  func mapView(_ mapView: MKMapView, didSelect view:
    MKAnnotationView) {
    if let view = view as? ArtworkView {
      countryIP.text = CountryModel.countries.first(where: { $0.name ==
view.artwork.country })?.serverIp
      country.text = view.artwork.country
      city.text = view.artwork.title

      selectedIndex = CountryModel.countries.firstIndex { $0.name ==
view.artwork.country }
    }
  }
```

```
  func mapView(_ mapView: MKMapView, didAdd views:
    [MKAnnotationView]) {
```

```

    }
}
Artwork.swift

class Artwork: NSObject, MKAnnotation {
    let title: String?
    let country: String?
    let coordinate: CLLocationCoordinate2D

    init(
        title: String?,
        country: String?,
        coordinate: CLLocationCoordinate2D
    ) {
        self.title = title
        self.country = country
        self.coordinate = coordinate

        super.init()
    }
    var image: UIImage {
        guard let name = country else {
            return imageLiteral(resourceName: "USA")
        }

        return UIImage(named: "countryLocationMark")!
        //return UIImage(named: name) ?? UIImage()
    }
}

```

CurrentLocationView.swift

```

class CurrentLocationView: MKAnnotationView {
    private var centerImage: UIImageView!
    private var largeCircleImage: UIView!
    private var smallCircleImage: UIView!

    override init(annotation: MKAnnotation?, reuseIdentifier: String?) {
        super.init(annotation: annotation, reuseIdentifier: reuseIdentifier)
    }
}

```

```

required init?(coder: NSCoder) {
    super.init(coder: coder)
    setup()
}

private func setup() {
    centerImage.image = UIImage(named: "locationMark")

    largeCircleImage.frame = CGRect(x: 0, y: 0, width: 38, height: 38)
    largeCircleImage.layer.cornerRadius = largeCircleImage.frame.height / 2
    largeCircleImage.backgroundColor = UIColor(red: 0.204, green: 0.596,
blue: 0.859, alpha: 1)
    largeCircleImage.alpha = 0.2
    largeCircleImage.center = centerImage.center

    smallCircleImage.frame = CGRect(x: 0, y: 0, width: 27, height: 27)
    smallCircleImage.layer.cornerRadius = largeCircleImage.frame.height / 2
    smallCircleImage.backgroundColor = UIColor(red: 0.204, green: 0.596,
blue: 0.859, alpha: 1)
    smallCircleImage.alpha = 0.4
    smallCircleImage.center = centerImage.center

    largeCircleImage.addSubview(smallCircleImage)
    smallCircleImage.addSubview(centerImage)

}
}
CountryModel.swift

```

```

struct CountryModel {
    var name: String
    var flag: UIImage
    var serverIp: String
    var username: String
    var serverAdress: String
    var password: String
    var ping: String

    static var countries: [CountryModel] = []
}

```

```

enum countryTableViewState {

```

```

case global
case special
case favorites
}

```

CountriesViewController.swift

```

class CountriesViewController: UIViewController {

    @IBOutlet weak var tableView: UITableView!

    @IBOutlet weak var searchBar: UISearchBar!
    @IBOutlet weak var goPremiumButton: UIButton!
    var selectedIndex: Int?
    @IBOutlet var stateButtons: [UIButton]!

    var savedCountries: [CountryModel]!

    var filteredCountries: [CountryModel] = CountryModel.countries
    var savedFilteredCountries: [CountryModel]!

    var state: countryTableViewState = .global {
        didSet {
            tableView.reloadData()
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        updateSavedCountries()
        searchBar.delegate = self

        if let textField = searchBar.value(forKey: "searchField") as? UITextField {
            textField.backgroundColor = UIColor(red: 0.718, green: 0.788, blue:
0.776, alpha: 1)
            //textField.font = myFont
            //textField.textColor = myTextColor
            //textField.tintColor = myTintColor
            // And so on...

            let backgroundView = textField.subviews.first

```

```

        if #available(iOS 11.0, *) { // If `searchController` is in
`navigationItem`
            backgroundColor =
UIColor.white.withAlphaComponent(0.3) //Or any transparent color that
matches with the `navigationBar` color`
            backgroundColor?.subviews.forEach({ $0.removeFromSuperview() })
// Fixes an UI bug when searchBar appears or hides when scrolling
        }
        backgroundColor?.layer.cornerRadius = 10.5
        backgroundColor?.layer.masksToBounds = true
        //Continue changing more properties...
    }

    clearButtons()
    goToGlobal(self)
    tableView.delegate = self
    tableView.dataSource = self
    if selectedIndex != nil {
        tableView.selectRow(at: IndexPath(row: selectedIndex!, section: 0),
animated: true, scrollPosition: .top)
    }
}

func updateSavedCountries() {
    var result: [CountryModel] = []

    for countryName in UserSettings.savedCountries {
        let country = CountryModel.countries.first { $0.name == countryName
    }!
        result.append(country)
    }
    savedCountries = result
    savedFilteredCountries = savedCountries
}
override func viewDidLoadSubviews() {
    //clearButtons()
}

private func clearButtons() {
    stateButtons.forEach { button in
        button.layer.cornerRadius = 10
    }
}

```

```

        button.layer.borderWidth = 1
        button.layer.borderColor = UIColor(red: 0.529, green: 0.529, blue:
0.529, alpha: 1).cgColor
        button.tintColor = UIColor(red: 0.737, green: 0.757, blue: 0.804, alpha:
1)
        button.backgroundColor = .clear
    }
}

```

```

private func selectButton(index: Int) {
    stateButtons[index].backgroundColor = UIColor(red: 0.18, green: 0.788,
blue: 0.608, alpha: 1)
    stateButtons[index].layer.borderColor = UIColor(red: 0.18, green: 0.788,
blue: 0.608, alpha: 1).cgColor
    stateButtons[index].layer.cornerRadius = 10

    stateButtons[index].tintColor = UIColor(red: 0.129, green: 0.122, blue:
0.125, alpha: 1)
}

```

```

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    tableView.reloadData()
    if UserSettings.isPremium {
        goPremiumButton.isHidden = true
    }
}

```

```

@IBAction func goBack(_ sender: Any) {
    navigationController?.popViewController(animated: true)
    let vc = navigationController?.topViewController as! MainViewController

    guard let selectedIndex = selectedIndex else {
        return
    }
    vc.selectedCountry = CountryModel.countries[selectedIndex]
}
@IBAction func goPremium(_ sender: Any) {
}

```

```

@IBAction func goToGlobal(_ sender: Any) {

```

```

        clearButtons()
        filteredCountries = CountryModel.countries
        selectButton(index: 0)
        state = .global
    }
    @IBAction func goToSpecial(_ sender: Any) {
        clearButtons()
        filteredCountries = CountryModel.countries
        selectButton(index: 1)
        state = .special
    }
    @IBAction func goToFavorites(_ sender: Any) {
        clearButtons()
        savedFilteredCountries = savedCountries
        selectButton(index: 2)
        state = .favorites
    }
}

```

```

extension CountriesViewController: UITableViewDelegate,
UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {

```

```

        switch state {
        case .global:
            return filteredCountries.count
        case .special:
            return filteredCountries.count
        case .favorites:
            return savedFilteredCountries.count
        }
    }
}

```

```

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
CountryCell.identifier, for: indexPath) as? CountryCell
        guard let tableViewCell = cell else { return UITableViewCell() }
        cell?.delegate = self
        switch state {

```

```

case .global:
    tableViewCell.state = .global

    tableViewCell.arrowImage.image = UIImage(named: "arrowImage")
    if savedCountries.contains(where: { $0.name ==
filteredCountries[indexPath.row].name }) {
        tableViewCell.saveButton.setImage(UIImage(named: "activeStar"),
for: .normal)
    } else {
        tableViewCell.saveButton.setImage(UIImage(named: "star"), for:
.normal)
    }
    tableViewCell.countryFlagImage.image =
filteredCountries[indexPath.row].flag
    tableViewCell.countryNameLabel.text =
filteredCountries[indexPath.row].name
    case .special:
        tableViewCell.state = .special

        tableViewCell.arrowImage.image = UIImage(named: "arrowImage")
        tableViewCell.saveButton.setImage(UIImage(named:
filteredCountries[indexPath.row].ping), for: .normal)
        tableViewCell.countryFlagImage.image =
filteredCountries[indexPath.row].flag
        tableViewCell.countryNameLabel.text =
filteredCountries[indexPath.row].name
        case .favorites:
            tableViewCell.state = .favorites

            tableViewCell.arrowImage.image = UIImage(named:
savedFilteredCountries[indexPath.row].ping)
            tableViewCell.saveButton.setImage(UIImage(named: "activeStar"), for:
.normal)
            let country = savedFilteredCountries[indexPath.row]
            tableViewCell.countryFlagImage.image = country.flag
            tableViewCell.countryNameLabel.text = country.name

        }

    return tableViewCell

}

```

```

func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
    return 68
}

```

```

func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
    navigationController?.popViewController(animated: true)
    let vc = navigationController?.topViewController as! MainViewController

    if !(state == .favorites) {
        vc.selectedCountry = CountryModel.countries[indexPath.row]
    } else {
        vc.selectedCountry = savedCountries[indexPath.row]
    }

}

}

```

```

extension CountriesViewController: UISearchBarDelegate {

```

```

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        searchBar.endEditing(true)
    }

```

```

    func searchBar(_ searchBar: UISearchBar, textDidChange searchText:
String) {
        filteredCountries = []
        savedFilteredCountries = []

        if !(state == .favorites) {
            if searchText == "" {
                filteredCountries = CountryModel.countries
            }

        } else {
            if searchText == "" {
                savedFilteredCountries = savedCountries
            }
        }
    }
}

```

```

    if !(state == .favorites) {
        for country in CountryModel.countries {
            if country.name.lowercased().contains(searchText.lowercased()) {
                filteredCountries.append(country)
            }
        }
    } else {
        for country in savedCountries {
            if country.name.lowercased().contains(searchText.lowercased()) {
                savedFilteredCountries.append(country)
            }
        }
    }

    self.tableView.reloadData()

}

}

extension CountriesViewController: CountryCellDelegate {
    func updateTableView() {
        updateSavedCountries()
        tableView.reloadData()
    }
}

extension UIViewController {
    open override func touchesBegan(_ touches: Set<UITouch>, with event:
    UIEvent?) {
        self.view.endEditing(true)
    }
}

```

ДОДАТОК Г

Результати перевірки роботи на співпадіння



Ім'я користувача:
Лісовиченко Олег Іванович

Дата перевірки:
08.01.2024 17:23:17 EET

Дата звіту:
08.01.2024 17:29:04 EET

ID перевірки:
1016049273

Тип перевірки:
Doc vs Internet + Library

ID користувача:
76913

Назва документа: ІП-22мп_Жердій_ПЗ

Кількість сторінок: 57 Кількість слів: 8053 Кількість символів: 62372 Розмір файлу: 11.12 MB ID файлу: 1015749390

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

9.24%
Схожість

Найбільша схожість: 6.67% з джерелом з Бібліотеки (ID файлу: 1011466185)

1.75% Джерела з Інтернету	53	Сторінка 59
9.11% Джерела з Бібліотеки	144	Сторінка 59

0% Цитат

- Вилучення цитат вимкнене
- Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 10 сторінок