МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ТУРИСТИЧНОГО ДОДАТКУ НА ПЛАТФОРМІ IOS

Здобувач освіти гр. ІН-81. Андрій ШКУРАТОВ

Науковий керівник, Сергій ШАПОВАЛОВ
кандидат ф.-м. наук, доцент

Завідувач кафедри Анатолій ДОВБИШ
доктор технічних наук,професор.

Суми 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи бакалавра

Студента четвертого курсу, групи ІН-81 спеціальності "Інформатика" денної форми навчання Шкуратова Андрія Олександровича.

Тема: "**Інформаційна технологія туристичного додатку на платформі iOS**"

Затверджена наказом по СумДУ

№_____ від _____2022 р.

Зміст пояснювальної записки: 1) огляд інструментів імплементації залежностей до проєкту; 2) налаштування залежностей проєкту; 3) розробка інтерфейсу додатку; 4) методологія розробки програмного продукту;

Дата видачі завдання "_____"_____2022 р.

Керівник випускної роботи _____ Шаповалов С.П.

Завдання приняв до виконання _____ Шкуратов А.О.

РЕФЕРАТ

Записка: 58 стор., 20 рис., 1 додаток, 6 джерел, 1 таблиця.

Об’єкт дослідження - розробка мобільного додатку.

Мета роботи — розробка информацийного та програмного забезпечення мобільного додатку.

Методи дослідження - метод реактивного програмування, середовище розробки xcode.

Результати — розроблено інформаційне та програмне забезпечення для мобільного додатку на платформі iOS. Інтерфейс додатку, та бізнес логіка розроблені за допомогою мови програмування swift. Дизайн розроблено за допомогою інстременту figma. Основним середовищем розробки програмного забезпечення є XCode.

СЕРЕДОВИЩЕ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ IOS,
XCODE, SWIFT, REACTIVE EXTENSION, STORYBOARD,
FIREBASE, MVVM, SNAPKIT.

ЗМІСТ

ВСТУП

Сучасний світ є відкритим, в якому всі можуть безперешкодно переміщатися сфера туризму стала однією з найприбутковіших. [8] Для полегшення пошуку квитків, бронювання готелів створюються окремі сервіси, що полегшують ці процеси. А інформаційне представлення країни є задатком її туристичного успіху. Україна має безліч всесвітньо відомих туристичних пам'яток, проте значно більша кількість не менш визначних пам'яток є приховані від публічного загалу. Через це розвиток інформаційного та туристичного напрямку є дуже важливим, особливо зараз. Мобільний додаток, що представляє визначні пам'ятки нашої Батьківщини в інформаційному наразі є необхідністю. [7]

Метою даної роботи є розробка туристичному додатку, що зображає визначні пам’ятки та представляє візитні картки Українських міст, на платформу iOS з урахуванням недоліків основних представників ринку надання цифрових послуг у сфері туризму. При створенні додатку необхідно використовувати основні тренди, запроваджені лідерами ринку в даній сфері.

Найпопулярнішими представниками в даній сфері є мобільний застосунок Google Maps, що доступний для платформ iOS та Android, та має веб версію, а також TripAdvisor, що є доступним, також, на всіх трьох платформах.

Перелічені додатки є найбільш популярними в туристичному сегменті, тож їх порівняння допоможе визначити їх недоліки та переваги, що вплине на розробку додатку.

1. ІНФОРМАЦІЙНИЙ АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Порівняльний аналіз аналогічних додатків

Для аналізу схожих додатків обрано топові представники в цьому напрямку: Google Maps та TripAdvisor.

Для початку порівняння необхідно визначити критерії за якими воно буде вестись. Визначимо 7 критеріїв: інформативність, легкість користування, можливість бронювати готелі, побудова маршрутів, наявність туристичних турів, можливість обрати активність, вибір ресторанів. Кожен критерій оцінюється за п’ятибальною шкалою за зручністю, де 1 - не зручно та 5 - дуже зручно. Після оцінки всіх критерії підбивається загальний висновок з коментарями, на основі висновків з порівняння буде сформульовано функціонал майбутнього додатку.

Почнемо з аналізу GoogleMaps. Основним функціонал додатку є географічне зображення інфраструктурних об’єктів, туристичних пам’яток, кафе, ресторанів, кінотеатрів, готелів тощо. Не менш важливим є функціонал побудови маршрутів та сортування географічних міток за категоріями, що значно полегшує навігацію по місцевості, перегляд вулиць через Режим Перегляду Вулиць. Основним недоліком додатку є зображення великої кількості локацій від початку, користувач від початку відвертається на велику кількість інформації, що представляється мапою.

Наступним розглянемо додаток TripAdvisor, він являє собою кишенькового помічника, що може порадити куди сходити в п’ятницю ввечері. На відміну від попереднього додатку, він сконцентрований на допомозі користувачу обрати активність або, наприклад, готель. На головній сторінці він запитує користувача: “Куди?”, що дає користувачу розуміння, що робити далі. Після вибору кінцевого пункту, додаток пропонує користувачу інформацію за категоріями, інформацію про визначні пам’ятки, ресторани, готелі тощо та має поради мандрівникам.

Після короткого огляду основного функціонала додатків, створимо таблицю за зазначеними вище критеріям та переглянемо результати (Таблиця 1). Після таблиці написано коментар до кожного розглянутого пункту.

№	Критерій	GoogleMaps	TripAdvisor
1	Інформативність	3	4
2	Легкість користування	4	5
3	Можливість бронювати готелі	4	4
4	Побудова маршрутів	5	1
5	Наявність туристичних турів	1	5
6	Можливість обрати активність	1	5
7	Вибір ресторанів	4	5
	Загальний балл	**3,1**	**4,1**

Таблиця 1 - Порівняльні характеристики відомих рішень

Пройдемося по кожному пункту:

1. GoogleMaps має недостатню інформативність щодо туристичних локацій, меню ресторанів тощо, він має суттєво географічну та фактичну інформативність. Своєю чергою TripAdvisor має інформаційні відомості про майбутні події та останні новини, за туристичною тематикою. Проте він не отримав найвищий бал через відсутність візитки до міста, що переконала б користувача завітати до нього.
2. Мапи не отримали найвищий бал через вищезазначені проблеми: велика кількість геоміток, велика мапа, це все розсіює увагу користувача і не допомагає йому зосередитися на пошуку цікавих локацій. Туристичний радник напрочуд легкий в користуванні, всі пункти впорядковані та навіть головна сторінка напряму каже що користувач повинен робити.
3. Обидва застосунки отримали однаковий бал через однакову систему бронювання готелів через сторонній сервіс, це не є зайвою проблемою, оскільки він має сайт і не вимагає завантаження мобільного застосунку. Хоча і не можна за це поставити найвищий бал.
4. Тут все зрозуміло, основним функціоналом першого сервісу є побудова маршрутів, а другий сервіс лише користується можливостями першого.

5. Тут ситуацію прямо протилежна четвертому пункту, перший додаток має слабку інформаційну базу, а другий додаток надає це як основний функціонал.
6. Тут та сама ситуація, що і з п’ятим пунктом.
7. Тут ситуація цікава, оскільки обидва додатки надають майже однаковий функціонал, проте він виглядає по різному, але перевага була на боці TripAdvisor через можливість забронювати столик.

Виходячи з зазначених вище пунктів можна розробити можна описати додаток, що буде задовольняти усім критеріям. Застосунок повинен мати легкий для розуміння інтерфейс та надавати користувачу базовий функціонал в зручному вигляді, обов’язково повинен містити мапу, що зображає усі найближчі туристичні локації. Для зручності необхідно пропонувати користувачу можливі тури та активності, щоб звільнити його від необхідності пошуку на сторонніх ресурсах. Необхідно надати користувачу перелік готелів для можливого бронювання. Екран міста буде головною його візитівкою, він повинен бути першим, що бачить користувач після запуску додатку.

Для надання такої кількості функціоналу необхідне значне пропрацювання з боку дизайнерів, які зможуть красиво зобразити та зробити правильні акценти в додатку. Також необхідна робота маркетологів, що можуть правильно представити кожне місто, що бачитиме користувач. Для більшої повноти необхідна потужна підтримка з боку сервера і для розширення аудиторії Android версія застосунку.

Через значне обмеження інструментарію, в рамках даної випускної роботи, буде розроблено додаток зі значно урізаним функціоналом, з можливістю подальшої доробки.

1.2 Засоби розробки мобільних додатків iOS

Для розробки додатків на дану платформу потрібно використовувати застосунок XCode, для відрисовки мапи ми будемо використовувати вбудовану бібліотеку map kit, для самої розробки буде використано низка додаткових бібліотек. Нижче обрано необхідні для розробки бібліотеки.

Xcode - це середовище розробки створене компанією Apple в 2003 році. Завантажити його можно безкоштовно в AppStore. Цей застосунок має усю документацію до розробки мобільних застосунків iOS, а також вбудований редактор інтерфейсів InterfaceBuilder. Окрім цього він має вбудовані симулятори для тестування застосунків в разі відсутності фізичного девайсу. Для розробки на iOS платформу розробник повинен мати комп'ютер з операційною системою MacOS або Linux. У випадку з останньою розробник матиме складності з вигрузкою додатку до AppStore. Для користувачів Windows є варіант з орендою віртуальних комп'ютерів Mac.

MapKit - це бібліотека мови програмування swift, що дозволяє нам використовувати карту, а також маніпулювати з нею. Альтернативою даного фреймворку є google maps kit, проте при великій кількості користувачів з розробника може братися плата за користування, що не підходить на даному етапі розробки.

Важливою проблемою сучасної iOS розробки є не строга типізація файлів ресурсів, це означає, що для отримання картинки розробник повинен вказати її ім'я в форматі строки і при помилці введення назви картинки, вона не буде загружена. Така проблема зустрічається і у випадку з локалізацією, відео, аудіо, сторібордами тощо. Для вирішення цієї проблеми було створено фреймворк R.swift.

R.swift - це інструмент, що дозволяє нам строго типізувати ресурси додатку, такі як картинки, строкові ресурси, екрани, тощо.

Для розробки логіки зручно використовувати асинхронне програмування, для реалізації цієї парадигми програмування існує два популярні фреймворки Combine та RxSwift. Серед них і будемо обирати.

RxSwift - є імплементацією стандарту Reactive extension (RX) в мові програмування swift. Даний фреймворк полегшує роботу з асинхронним програмуванням, та дозволяє нам імплементувати його в інтерфейс не впливаючи на UX додатку. Даний фреймворк був розроблений у 2011 році компанією Microsoft та знайшов свою реалізацію в різних мовах програмування: C#, C++, Java Script, Python, Ruby, Swift тощо. Перша

платформа, для якої було розроблено даний фреймворк є .Net Framework. Даний фреймворк має підтримку починаючи з версії iOS 9.0

Combine надає декларативний API Swift для обробки значень з часом. Ці значення можуть представляти багато видів асинхронних подій. Combine оголошує, що publishers отримують значення, які можуть змінюватися з часом, а subscribers отримують ці значення від publisher. Вперше було представлено на WWDC 2019 року. Даний фреймворк є нативною імплементацією від Apple, проте в більшості своїй він був розроблений як доповнення до фреймворку SwiftUI, він має багато обгорток, що імплементовані для оновлення інтерфейсу додатку, написаного на SwiftUI. Даний фреймворк має підтримку починаючи з версії iOS 13.

Обидва фреймворки мають у своїй концепції схожі ідеї, та багато однакових елементів під різними назвами. Observable в rx зустрічається як Publisher в комбайні, Subscribers зустрічаються в обох фреймворках.

Головною перевагою RX перед Combine є кроссплатформеність, оскільки даний фреймворк має імплементацію в багатьох інших мовах програмування, легше знайти інформацію по помилках та концепції фреймворку. Виходячи з усього вищезазначеного в додатку будемо використовувати RxSwift.

Сучасний світ надає два основні способи розробки користувацького інтерфейсу, через storyboards з використанням модулю UIKit, та з використанням SwiftUI. Другий варіант є доволі молодим, також він обмежує потенційну аудиторію додатку. Тож найліпшим є перший варіант. Хоча він має свої переваги та недоліки, основною перевагою є система обмежень, що накладається на користувацькі інтерфейси й допомагає розробляти їх гнучко для різних екранів. Водночас сама система storyboards є доволі складною для обчислення комп’ютером і це значно сповільнює роботу. Компромісним варіантом є використанням стороннього інструменти для розробки інтерфейсу, що полегшує роботу з констрейнтами, таким інструментів є snap kit.

SnapKit - інструмент для полегшення розробки користувацького інтерфейсу. Даний фреймворк дозволяє нам розробляти інтерфейс за допомогою

коду. Він полегшує роботу з констрейнтами та дає можливість оновлювати розташування елементів відповідно до поточного стану екрану.

В сучасних додатках невід'ємною частиною розробки є серверна чистина додатку, що є буквально записником додатку. Наш випадок не є виключенням, більша частина інформації повинна зберігатися на сервері, в якості сервера будемо використовувати сервіс firebase, зокрема firebaseremote config, що є потужним інструментов для налаштування додатку без необхідності його оновлення, та firestore databse, що дозволяє зберігати необхідні дані.

FirebaseRemoteConfig - це хмарний сервіс, що дозволяє розробникам змінювати налаштування додатку не змушуючи користувачів оновлювати його, це значно полегшує UX додатку.

FirebaseFirestore - це гнучка, масштабована хмарна база даних від Firebase та Google Cloud Platform для веб, мобільних платформ, і серверних додатків. Як і Firebase Realtime Database, вона синхронізує ваші дані між клієнтськими додатками за допомогою слухачів реального часу і пропонує підтримку офлайн режиму для мобільних платформ і вебу.

Для відображення інформації в додатку часто використовуються картинки, що дозволяє користувачеві легше сприймати інформацію. Проте в розробника майже немає можливості зберігати картинки локально, оскільки вони можуть часто змінюватися, що приведе за собою постійні оновлення додатку. Для запобігання цьому картинки необхідно зберігати на сервері з можливістю їх постійного оновлення. Для завантаження таких картинок дуже зручним є фреймворк Kingfisher.

Kingfisher - це потужна бібліотека Swift для завантаження та кешування зображень з Інтернету. Це дає розробнику можливість використовувати "чистий" Swift для роботи з віддаленими зображеннями.

1.3 Постановка задач

Результатом проведеного аналізу існуючих додатків та інформаційного їх забезпечення є постановка задач для дослідження в бакалаврській роботі - створити програмне та інформаційне забезпечення додатку для туристичної

індустрії. Для реалізації поставленого завдання необхідно реалізувати наступні задачі:

1. Розглянути інструменти для імплементації залежностей до проекту, та обрати оптимальний
2. Встановити та налаштувати залежності проєкта
3. Розробити простий інтерфейс додатку
4. Розробка сторінок додатку, що не будуть обмежуватися наявним інструментарієм.

Результатом виконання даної роботи буде розглянуто сучасні методології розробки iOS додатку, також буде розроблений туристичний додаток, що буде представляти місто Суми та його найвідоміші пам’ятки. Ще одним результатом роботи буде розглянуто подальші преспективи проєкту та функціонал, що може бути розроблений.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Інструменти імплементування залежностей до проєкту

Мова програмування swift має декілька способів імплементування залежностей: cocoapods, carthage та swift package manager. Розглянемо кожен з них.

Cocoapods - це централізований менеджер залежностей для проєктів на мові програмування swift та objective c cocoa. Це є опен сорсним проєктом написаним мовою програмування Ruby. Цей менеджер є централізованим, оскільки всі фреймворки він містить в одній репозиторії з назвою Specs. Також він має зручний сайт з пошуком залужностей. Для користування необхідно встановити інструменти до терміналу, та користуватися командами, що надаються. Головними перевагами цього методу є багата база доступних фреймворків, зручна система команд та легкість користування. Основним недоліком є низька швидкість збору проєкта, через те, що кожну залежність компілятор буде будувати.

Carthage - це децентралізований менеджер залежностей для проєктів на мові програмування swift та objective c cocoa. Він є також опен сорсним, проте він побудований на мові програмування swift. На відміну від попереднього менеджеру, він не має одієї репозиторії, що знижує вірогідність збоїв в системі, проте це ускладнює пошук залежностей. Цей менеджер також використовує командну строку для додавання залежностей до проєкту. Головною перевагою є швидкість в порівнянні з попереднім менеджером. В той час як головним недоліком є складність у встановленні залежностей, зокрема велика кількість кроків для встановлення, через це можна легко пропустити один із кроків встановлення, також не кожен фреймворк підтримує цей менеджер.

Swift Package Manager (SPM) вперше з’явився у версії swift 3.0, як сказано в офіційній документації - це інструмент для керування розповсюдженням вихідного коду, спрямований на те, щоб спростити обмін вашим кодом та повторне використання коду інших. Інструмент безпосередньо вирішує проблеми компіляції та зв’язування пакетів Swift, керування залежностями, керування версіями та підтримку гнучких моделей розповсюдження та

співпраці [1]. Нещодавно додане графічне відображення цього інструменту до середовища розробки XCode (рисунок 2.1).

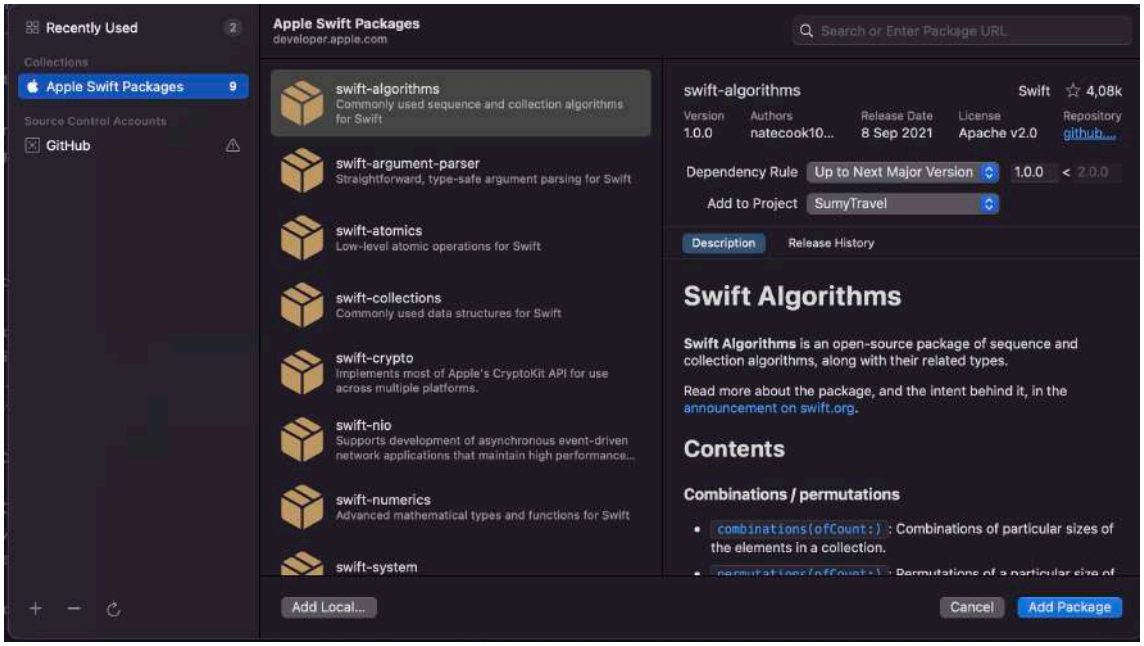

Рисунок 2.1 - Графічне SPM в XCode

Перевагою даного менеджеру в тому, що він є вбудованим у середовище розробки та працює на платформі Linux. Недоліками ж є відсутність підтримки деякої частини фреймворків.

Вважаючи все вище зазначене для роботи над даним проєктом буде використаний менеджер cocoapods, не зважаючи на низьку швидкість першого запуску проєкту він є більш зручним у використанні та має підтримку більшої кількості фреймворків.

2.2 Розробка інтерфейсу додатку

Для розробки інтерфейсу додатку використаємо застосунок Figma. Він дозволяє створювати користувацькі інтерфейси як для десктопу так і для мобільних додатків. Наш додаток буде складатися з трьох вікон:

- Мапи (основне вікно додатку)
- Новини
- Налаштування

Вікно мапи буде містити в собі мапу з відмітками туристичних локацій міста Суми, при натисканно на відмітку з’являється вікно з назвою, що переводить на сторінку з описом локації (рисунки 2.2 - 2.4).

Рисунок 2.2 - Мапа з локаціями

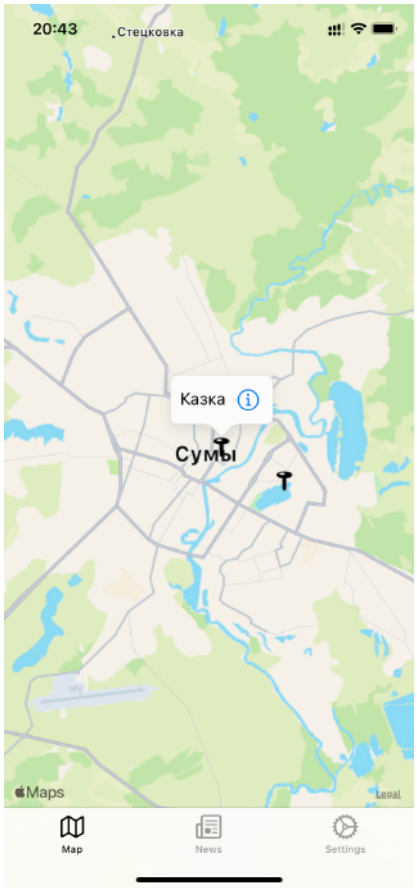

Рисунок 2.3 - Вікно з назвою локації

Рисунок 2.4 - Вікно опису локації

Сторінка новин буде містити список подій які заплановані на найближчий час в місті Суми при натисканні на подію відкриється сторінка з описом події (рисунки 2.5 - 2.6).

Рисунок 2.5 - Сторінка найближчих подій

Рисунок 2.6 - Сторінка опису подій

Сторінка налаштувань буде містити посилання на зміну локалізації додатку.

2.3 Налаштування залежностей проєкту

Далі необхідно до нашого проєкту додати залежності, вище ми обрали менеджер залежностей cocoapods. В терміналі переходимо до папки нашого проєкту та вводимо команду:

pod init

В папці нашого проєкту створено podfile куди ми заносимо наші залежності (рисунок 2.7). Залежності нашого проєкту були зазначені вище, це: R.swift, RxSwift, RxCocoa, SnapKit, FirebaseRemoteConfig, FirebaseFirestore, Kingfisher. Після цього необхідно буде налаштувати наш проєкт до firebase для використання можливостей firebase firestore.

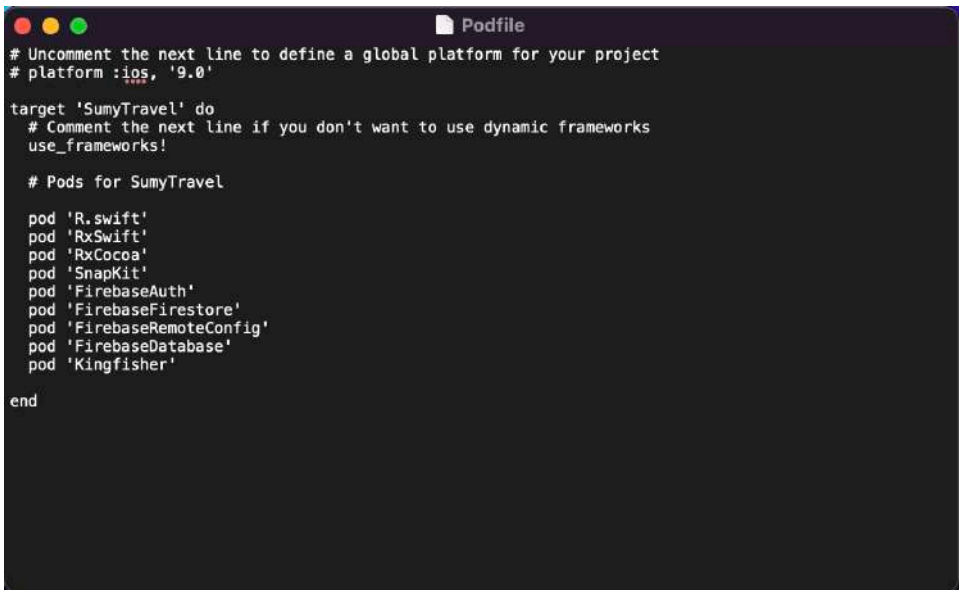

Рисунок 2.7 - Контент файлу podfile

Після додавання всіх залежностей виконуємо команду:

pod install

Вона встановлює залежності до нашого проєкту. Далі переходимо до консолі firebase, та натискаємо кнопку “Додати проєкт” (рисунок 2.8). Опинившись на сторінці створення проєкту першим кроком ми обираємо ім’я, що буде відображено в консолі (рисунок 2.9), оберемо ім’я SumyTravel, натискаємо “Далі”, на другій сторінці підтверджуємо політику безпеки (рисунок 2.10), на третій сторінці обираємо аккаунт, що буде управляти проєктом (рисунок 2.11). Наш поєкт створено, після чого ми опиняємося на сторінці “Project Overview” (рисунок 2.12).

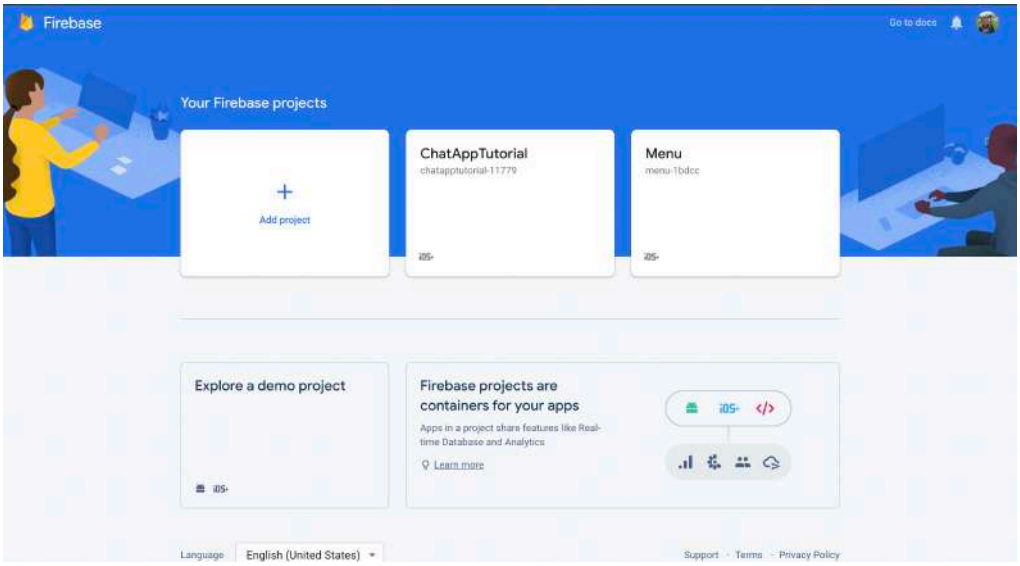

Рисунок 2.8 - Консоль проєктов firebase

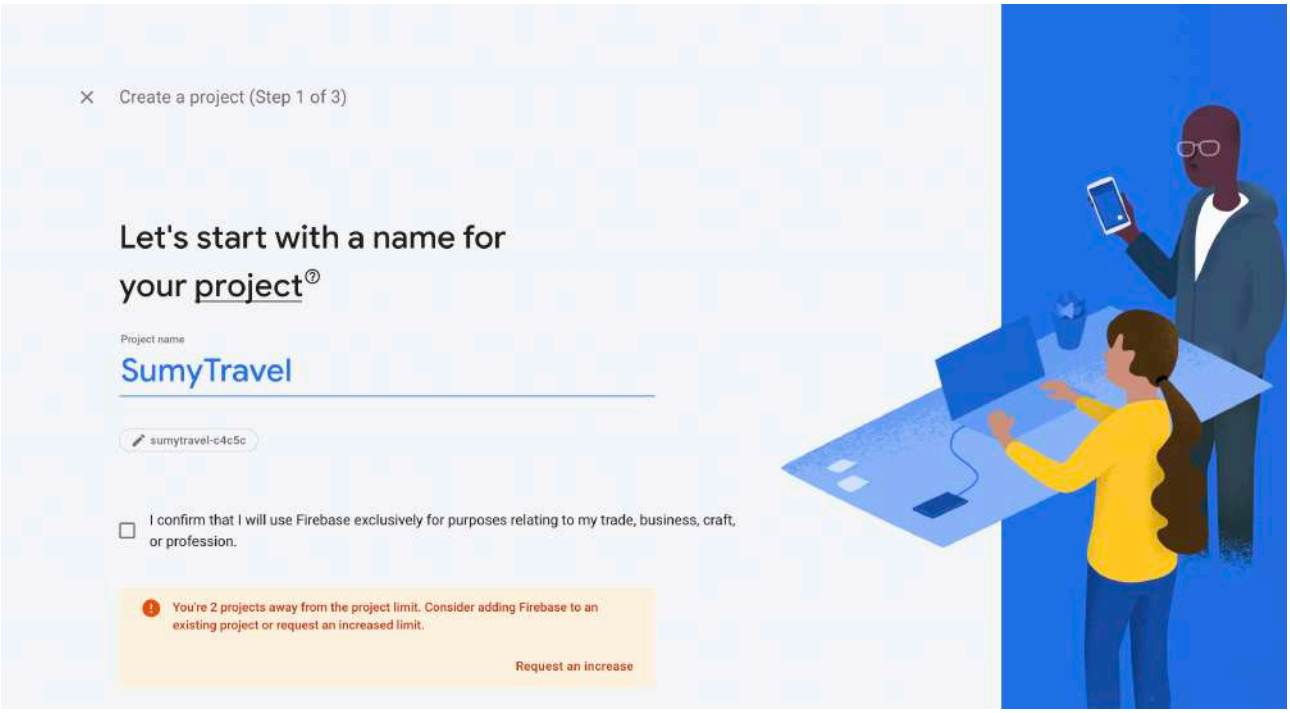

Рисунок 2.9 - Перша сторінка створення проєкту

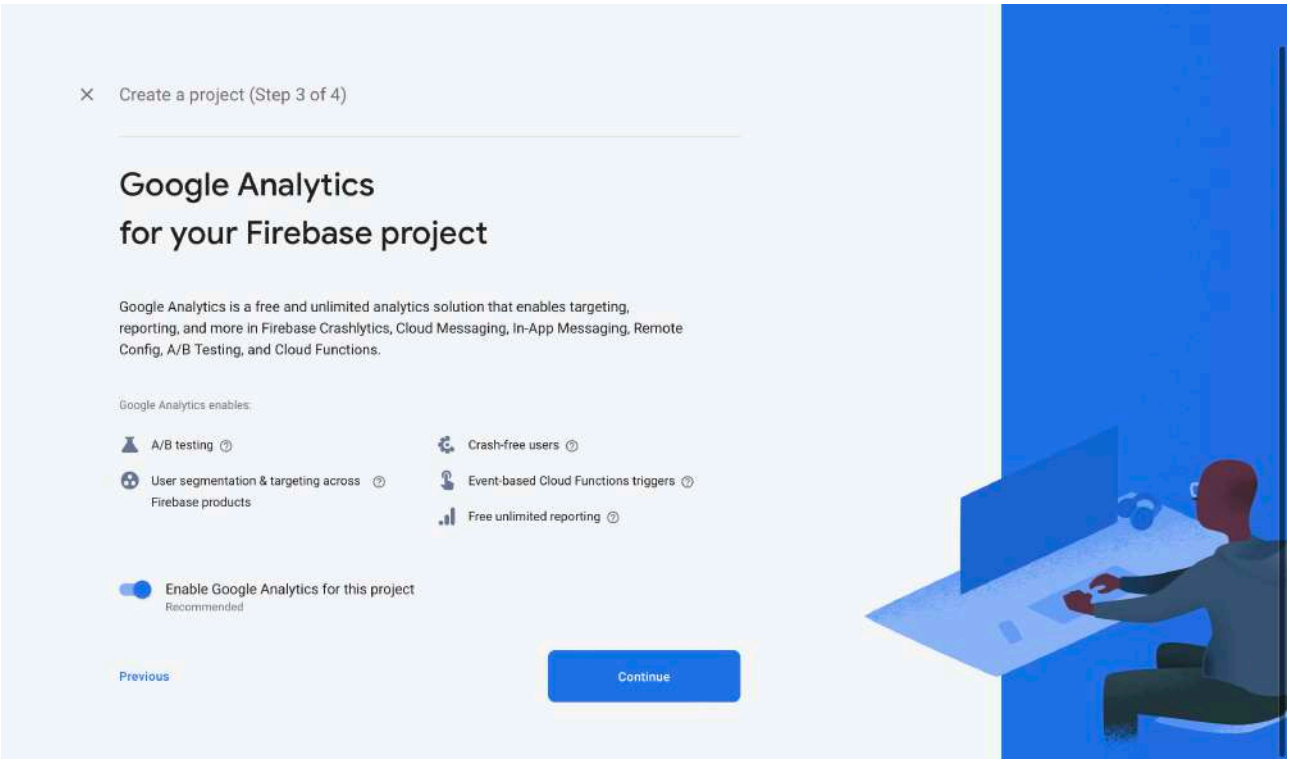

Рисунок 2.10 - Друга сторінка створення проєкту

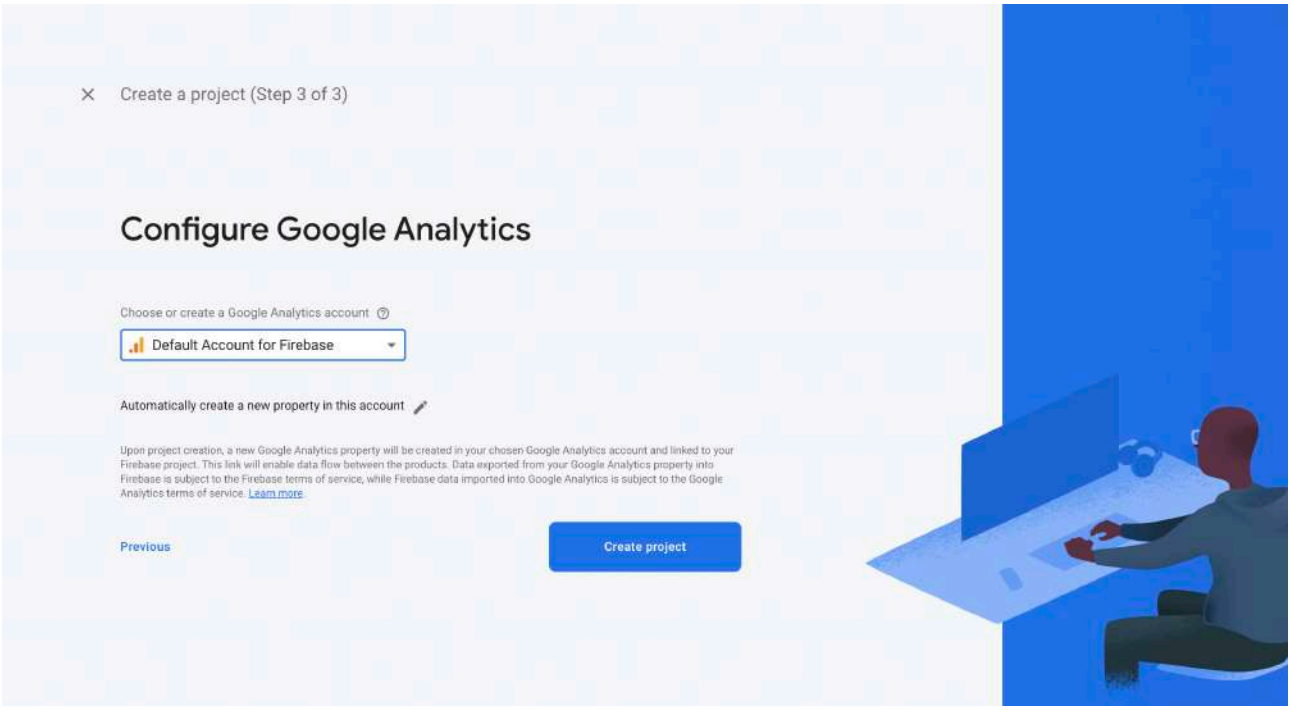

Рисунок 2.11 - Третя сторінка створення проєкту

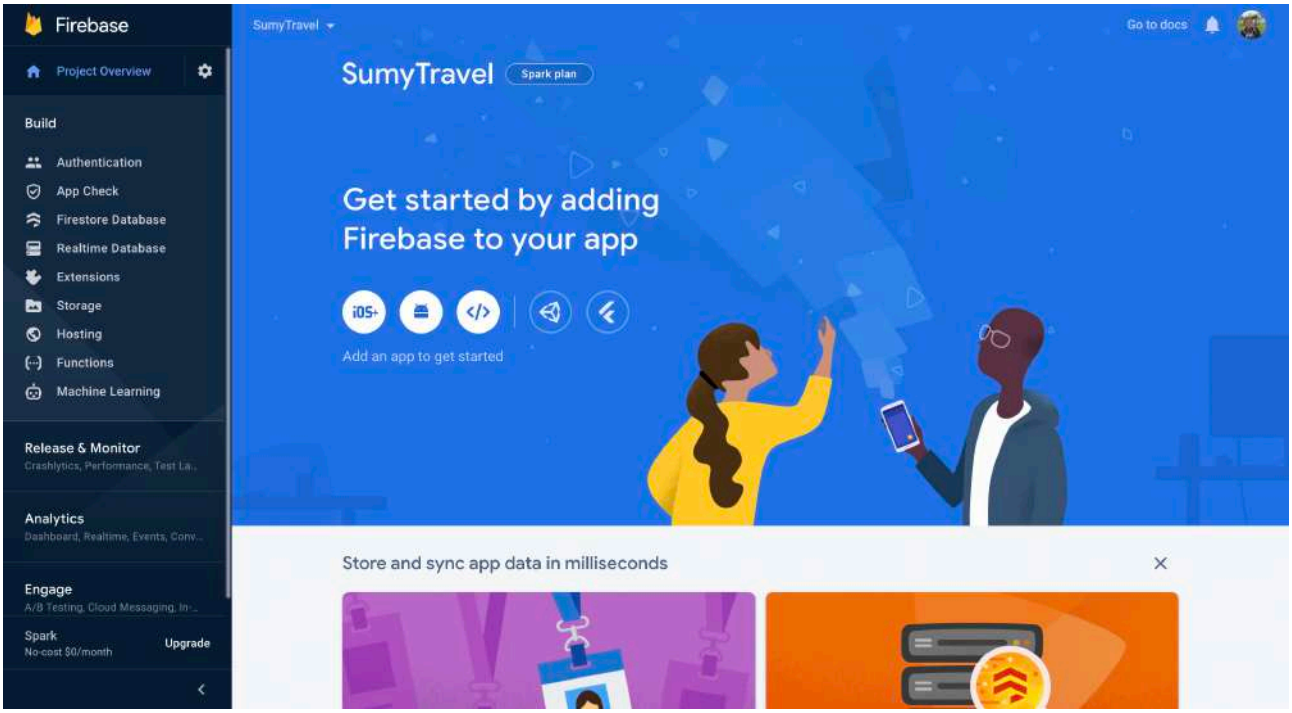

Рисунок 2.12 - Головна сторінка проєкту

Ми налаштували наш проєкт у firebase. Далі необхідно налаштувати firebase firestore де ми будемо збурігати наш контент. Для цього переходимо на сторінку firebase Firestore (рисунок 2.14). І натискаємо кнопку “Go to Cloud Console”. Ми опиняємося на вкладці створення баз даних (рисунок 2.15). Де ми

натискаємо кнопку “Create Entity” де ми створюємо нашу таблицю з новинами (рисунки 2.16 - 2.17). Після цього ми створили базу даних, що будемо використовувати в нашому проєкті.

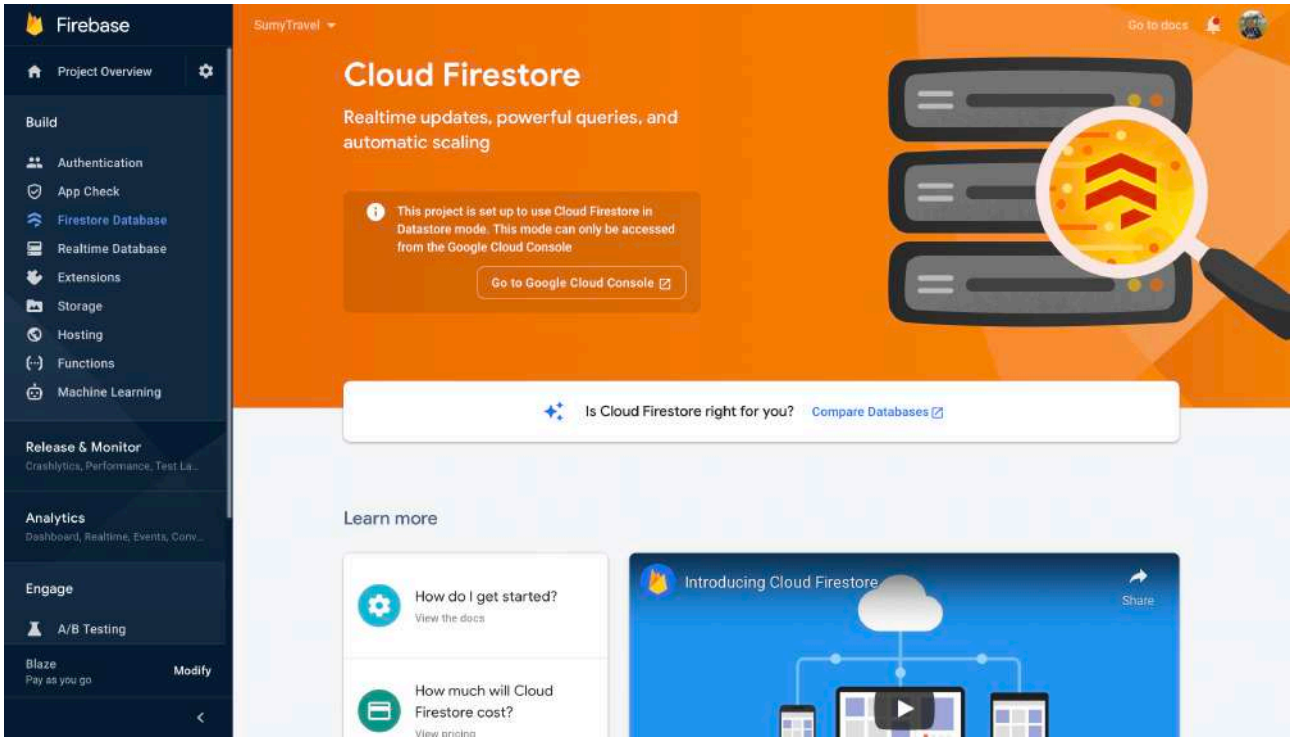

Рисунок 2.14 - Сторінка Firebase Firestore

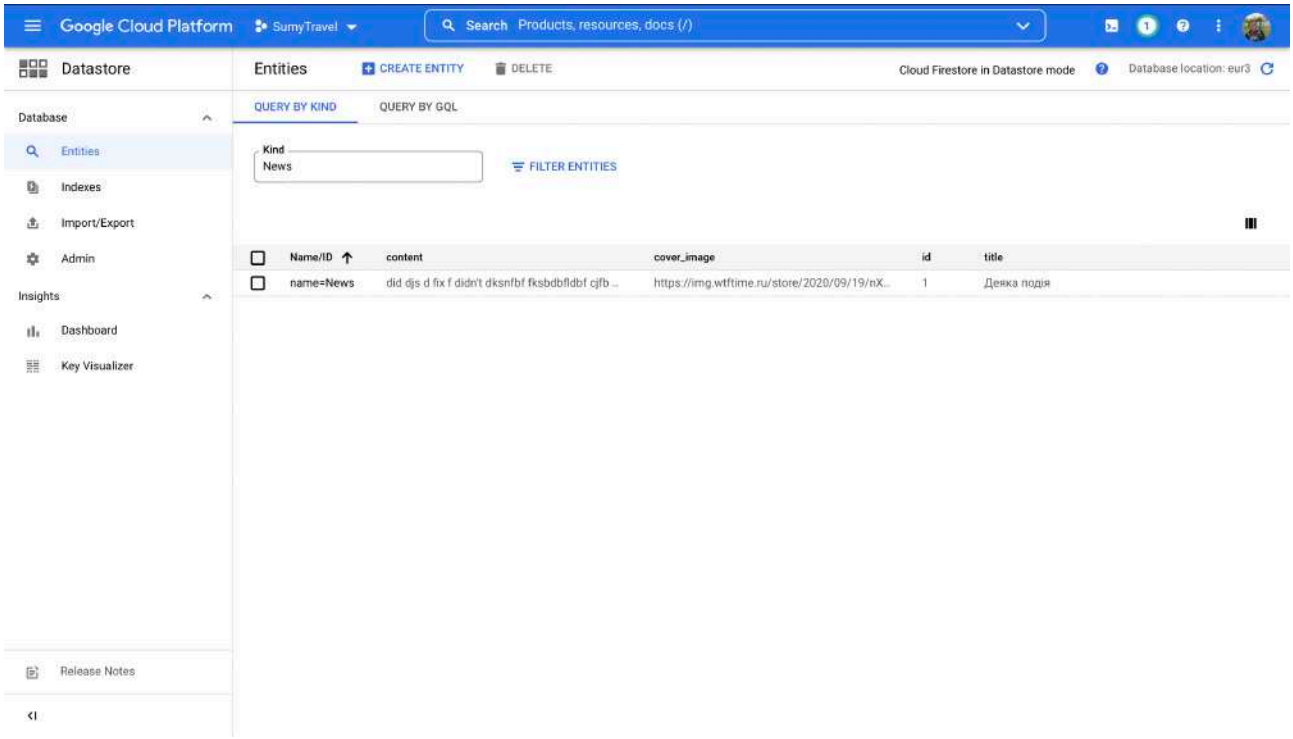

Рисунок 2.15 - Сторінка менеджеру баз даних

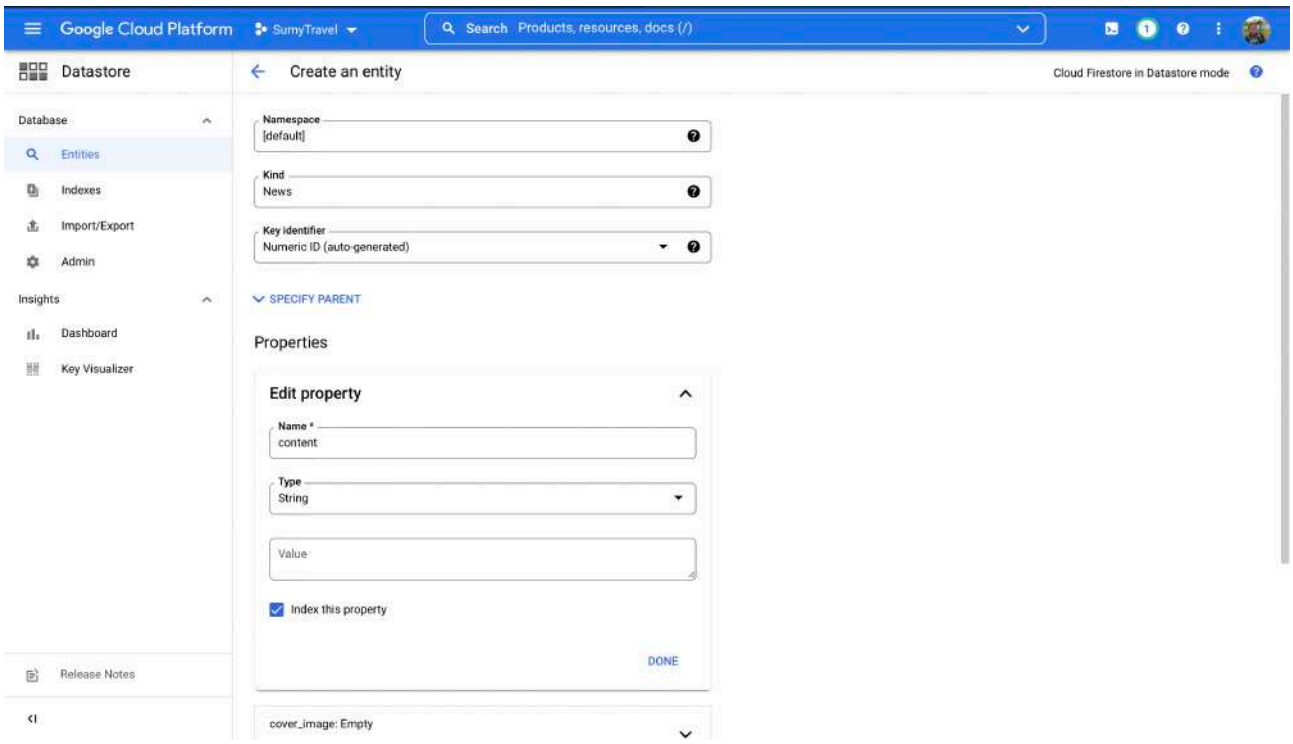

Рисунок 2.16 - Сторінка створення баз даних

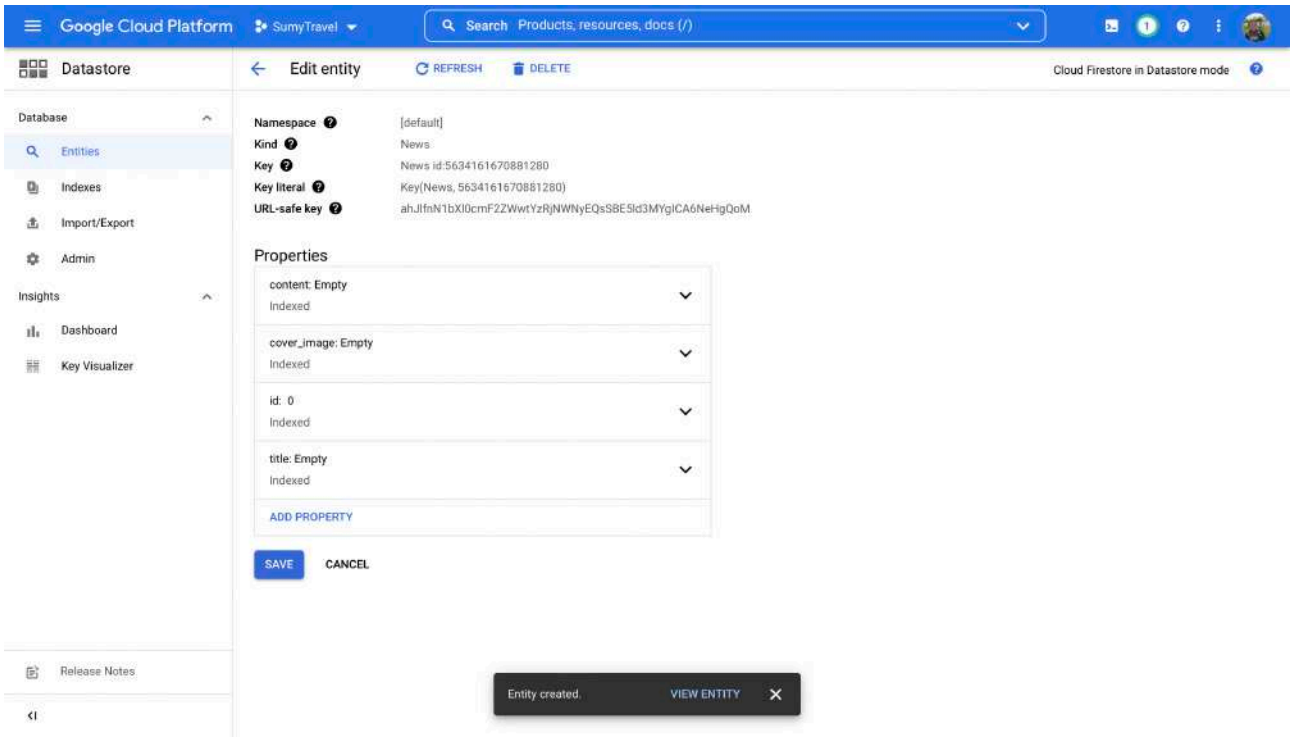

Рисунок 2.17 - Заповнене поле в базі даних

2.4 Методологія розробки програмного продукту

Для подальшої розробки необхідно визначитися з загальним розташуванням екранів в додатку, для цього розробимо схему розташування екранів (рисунок 2.18).

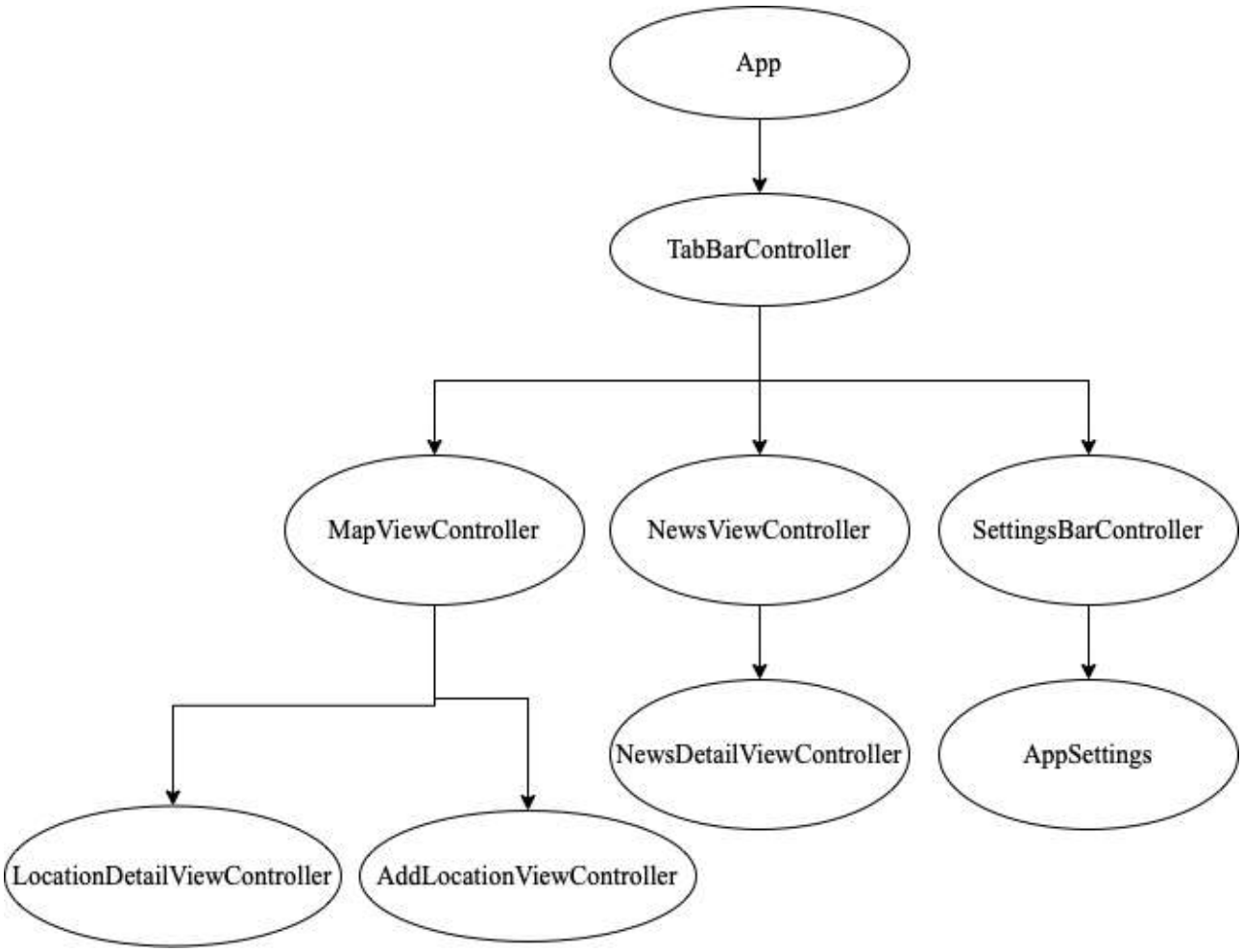

Рисунок 2.18 - Схема розташування екранів

Після запуску додатку користувач опиняється на TabBarController, що містить в собі екрани MapViewController, NewsViewController fn SettingsBarController. Після чого відбувається стандартна навігація.

В якості патерну розробки обрано MVVM (Model View ViewModel). MVVM - патерн, що дозволяє розділити логіку додатку від візуальної частини додатку, цей патерн є архітектурним. Головною перевагою даного патерну є відмежування даних дизайну, що полегшує читання коду та подальшу розробку додатку (рисунок 2.19)

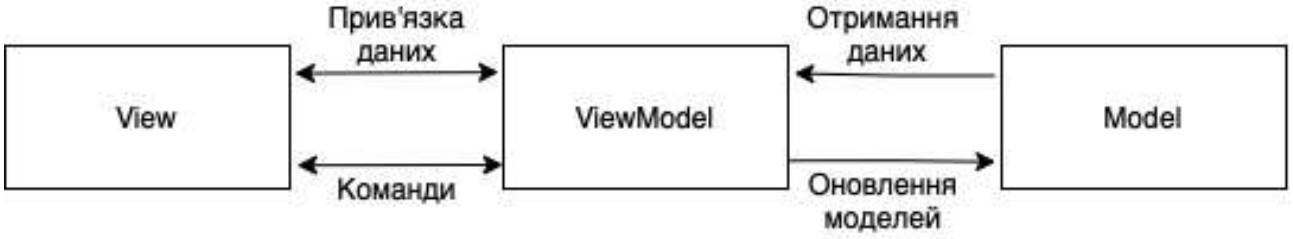

Рисунок 2.19 - Загальний вигляд архітектури MVVM

Наразі для розробки інтерфейсу iOS додатків можна використовувати дві технології: Storyboard та SwiftUI.

Storyboard — це функція, вперше представлена в iOS 5, яка економить час на створення інтерфейсів користувача для програм. Storyboards дозволяють створювати прототип і проектувати декілька екранів контролера представленні в одному файлі, а також дозволяють створювати переходи між контролерами екранів.

SwiftUI допомагає створювати структуру нашого єкрану виключно у коді. При чому Apple пропонує декларативний стиль написання цього коду (додаток 2), розробник просто прописує процес, що потрібно зробити для отрумання результати.

Для написання додатку обрано перший спосіб, проте UI буде написано за допомогою коду. Оскільки Storyboard хоч і є доволі зручними, сам файл Interface builder, що займається рендером екранів, виконує дуже багато роботи з розрахунку позицій ui елементів, що сповільняє роботу ПК та створює незручності при розробці. Для написання інтерфейсу використано фреймворком SnapKit.

SnapKit, легкий DSL (доменна мова), що дозволяє легко працювати з автоматичним макетом та обмеженнями. Auto Layout — це потужний інструмент для опису зв’язків та обмежень між різними уявленнями та складними ієрархіями представлень у програмі.

Для реалізації даного фреймворку до кожного контроллеру пишеться окремий файл view, в якому прописується інтерфейс екрану, після чого у методі loadView() він присвоюється змінній view. В методі viewDidLoad налаштовуємо усі дії для кнопок та викликаємо додаткові методи для наповнення нашого дизайну інформацією.

Розглянемо програмну реалізацію додатку по екранам. Розпочнемо з MapViewController. На цьому екрані користувач може переглянути мапу з локаціями на екрані, переглянути сторінку з деталями локації та додати локацію. Повну комунікацію екрану можна відобразити за допомогою Use Case Diagram (Рисунок 2.20).

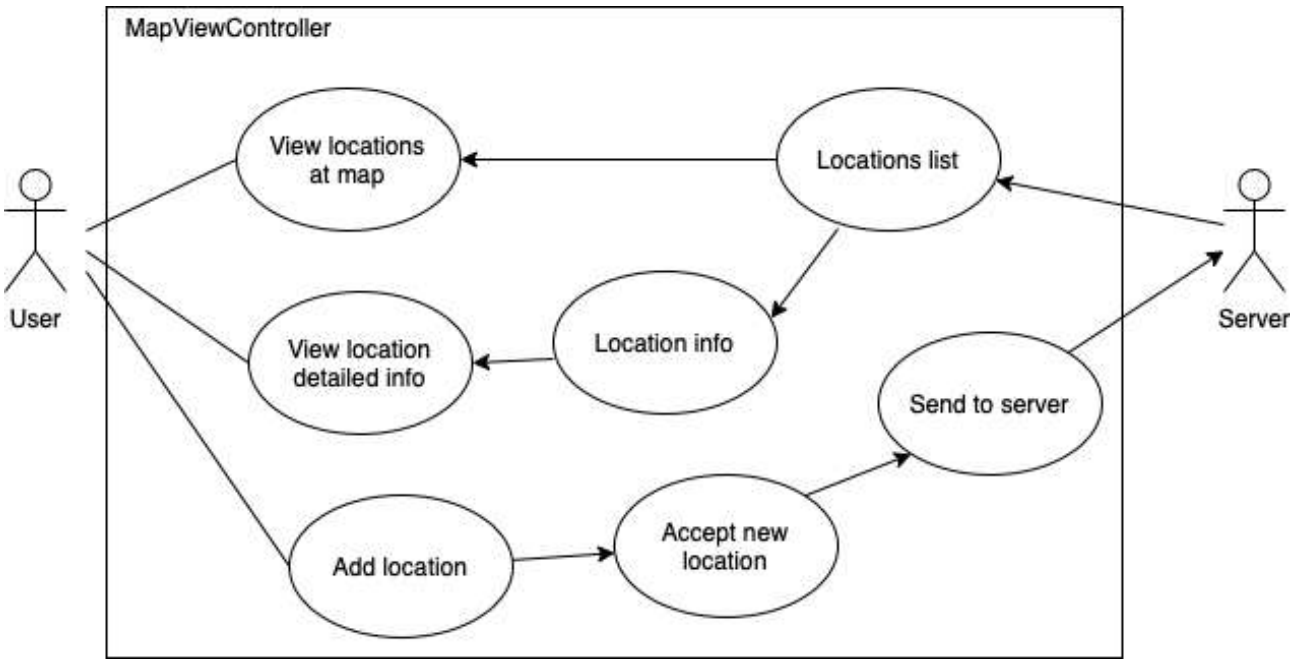

Рисунок 2.20 - MapViewController Use case diagram

Даний екран містить один UI елемент - мапу. Дана мапа надається нативним модулем apple - map kit. MapKit — це потужний API, доступний на пристроях iOS, який дозволяє легко відображати карти, позначати місця, доповнювати користувацькими даними і навіть малювати маршрути чи інші фігури зверху. У нашому view controller`і ми повинні підписати мапу до делегату, що дозволяє імплементувати методи мапи в рамках даного контроллеру.

Для даного екрану використано наступні методи мапи:

1. mapView(_ mapView: MKMapView, didUpdate userLocation: MKUserLocation)
2. mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView, calloutAccessoryControlTapped control: UIControl)
3. mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView?

Перший метод, з представлених, оновляє локацію користувача, та вирівнює нашу карту відносно його місцезнаходження.

Другий метод оброблює натискання на аннотацію на карті, після натискання з’являється callout з назвою локації, після натчскання на який користувач опиняється на сторінці з детальним описом локації.

Третій метод налаштовує наші локації, та визначає як вони будуть виглядати на карті.

Всі ці методи надає MKMapViewDelegate (Додаток 1).

Для завантаження локацій використано firestore database та rxswift для відображення.

Firestore database - це гнучка, масштабована база даних для мобільних, веб- та серверних розробок від Firebase і Google Cloud. Як і Firebase Realtime Database, він синхронізує ваші дані між клієнтськими додатками за допомогою прослуховувачів у реальному часі та пропонує автономну підтримку для мобільних пристроїв та Інтернету, щоб ви могли створювати адаптивні програми, які працюють незалежно від затримки в мережі або підключення до Інтернету. Cloud Firestore також пропонує плавну інтеграцію з іншими продуктами Firebase і Google Cloud, включаючи Cloud Functions.[5]

Після завантаження даних, зберігаємо їх змінній locations у MapViewModel, що має тип BehaviorRelay<[LocationModel]?>.

BehaviorRelay - це ретранслятор, який передає останній спостережений елемент і всі наступні спостережені елементи кожному спостерігачу, на який підписався.[6]

У нашому контроллері ми підписуємося до спостерігача і в методі onNext оновлюємо дані. В якості типу даних маємо опціональний масив локацій, для безпечного використання необхідно розвернути його за допомогою guard statement. Для невілювання загрози memory leak необхідно зробити посилання на контроллер за допомогою [weak self], тобто ми робимо слабке посилання на контроллер.

Взагалі реактивне програмування описує парадигму проектування, яка покладається на логіку асинхронного програмування для обробки оновлень у реальному часі статичного вмісту. Він забезпечує ефективний засіб — використання автоматизованих потоків даних — для обробки оновлень даних вмісту щоразу, коли користувач робить запит.

Потоки даних, що використовуються в реактивному програмуванні, є когерентними, зв’язаними колекціями цифрових сигналів, створених на постійній або майже безперервній основі. Ці потоки даних надсилаються з джерела - наприклад, датчика руху, датчика температури або бази даних інвентарю продукції - у відповідь на тригер.

У мобільному програмуванні, цю парадигму зручно використовувати для оновлення користувацького інтерфейсу без зупинки додатку, що позитивно впливає на UX додатку.

Для відправки локації використано UILongPressGestureRecognizer, для позначення на карті локації та визначення її координат, та MessageUI, для відправлення повідомлення через email.

Фреймворк Message UI надає спеціалізовані контролери перегляду для представлення стандартних інтерфейсів композицій для текстових повідомлень електронної пошти та SMS (Short Messaging Service). Цей інтерфейс використаний, щоб додати можливості доставки повідомлень, не вимагаючи від користувача залишати вашу програму.

Для імплементації необхідних методів використано MFMailCompose ViewControllerDelegate зокрема його метод:

func mailComposeController(_ controller: MFMailComposeViewController, didFinishWith result: MFMailComposeResult, error: Error?)

Він дозволяє обробляти події після відправки повідомлення. В нашому випадко видаляти тимчасово додану позначку локації та закривати екран з повідомленням.

Наступним екраном є NewsViewController. На ньому користувач може переглянути актуальні події Сумщини. Даний екран дозволяє користувачу переглянути стислу інформацію по події або розглянути більш детальну інформацію про подію (рисунок 2.21). Головним елементом цього екрану є UITableView. UITableView - це ui елемент, який представляє дані за допомогою рядків в одному стовпці. Для відображення даних необхідно прописати рядок, де розміщена скорочена інформація по події, при натисканні на рядок користувач опинеться на сторінці деталей новини.

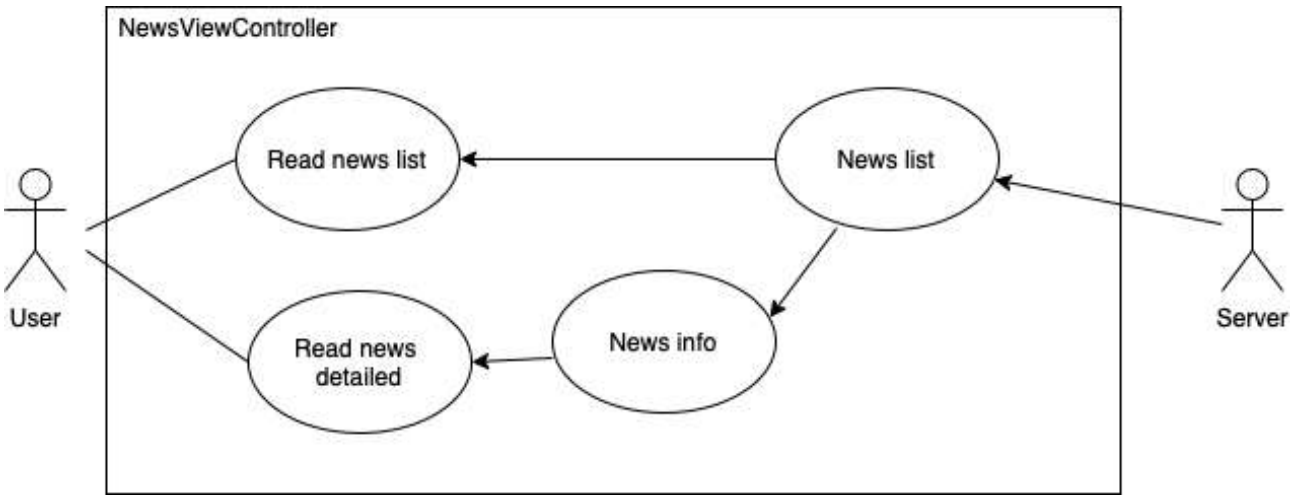

Рисунок 2.21 - NewsViewController Use case diagram

На екрані MapViewController було розглянуто використання rxSwift для завантаження даних, та оновлення інтерфейсу, проте він має ширші можливості, зокрема він допомагає при роботі з таблицями. RxSwift надає розширення до, майже, усіх елементів інтерфейсу, включаючи таблиці, так ми можемо через змінну rx звертатися до даних нашої таблиці і додавати до них значення. Слідкувати за зміною значень ми можемо за допомогою методу bind. Він підписується на спостережувану послідовність за допомогою спеціальної функції зв’язування та остаточного параметра, що передається функції сполучника після передачі self. Разом з параметром, ми також зазначаємо ячейку, яка буде передана до таблиці. Після отримання кожного значення, ми можему налаштовувати ячейку, та передавати туди дані.

Також rx допомагає нам обробляти натискання на ячейки, передаючи нам модель, що в ній зберігається.

Останнім є екран SettingsViewController, що містить в собі основні налаштування, наразі це зміна мови додатку. Цей екран використовує усі техніки з попередніх екранів, за виключенням завантаження даних, вони зберігаються локально за допомогою enum’у.

Наразі додаток має такий вигляд, проте він не є остаточним, в преспективі додаток можна значно покращити. Наприклад відмовитися від сторінки мапи, замість неї можна зробити сторінку візитки, де ми зможемо переглянути основні відомості за допомогою картинок, перейти на сторінку мапи.

Наступним кроком є відмова від концепту додатку про одне місто, в налаштуваннях можна додати можливість вибору міста або області після чого додаток змінює свою тематику. Також можна зробити адаптивність додатку під поточне місцезнаходження користувача з можливістю вимикання або вмикання даної функції. Можливе використання технологій доповненої реальності.

Основною перепоною в розробці такого додатку є мала команда розробників. Для подальшого розвитку такого додатку необхідна робота багатьох людей: дизайнерів, маркетологів та розробників. Також значною перепоною є відсутність веб розробника, що може спроектувати та розробити необхідну базу даних. Якщо дивитися в рамках проекту для одного міста, то достатньо засобів firebas, для реалізації більш маштабного проєкту необхідна роботи власного серверу.

Для більш повного розповсюдження проєкту є необхідним створення android додатку.

ВИСНОВКИ

В ході виконання випускної роботи проведені дослідження по створенню мобільного туристичного проекту для міста Суми, з метою популяризації та просування його в інформаційному просторі. Результатами роботи є наступне:

1. Проведений інформаційний огляд та обрано інструментарій, застосовний для даного проєкту. В якості інструменту імплементації залежностей до iOS порєкту обрано cocoapods. Він має легший в користуванні порівнянно з carthage та більшу базу фреймворків порівняно з swift package manager.
2. Інтерфейс додатку створено за допомогою додатку figma
3. Встановлено фреймворки проєкту з врахуванням обраного інструменту імплементації залежностей. Також розглянуте налаштування проєкту в сервісі firebase, яке використовується в якості серверу нашого додатку
4. Розроблено мобільний додаток для платформи iOS та детально розглянуто етапи розробки
5. Розглянуто подальші преспективи розробки та ідеї для поліпшення інтерфейсу та роботи додатку

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Репозиторія apple/swift-package-manager [Електронний ресурс] // Режим доступу до ресурсу: https://github.com/apple/swift-package-manager
2. Дакетт Д. HTML та CSS. Розробка та дизайн веб-сайтів. – Таллінн: Ексмо, 2020. – 302 с.
3. The swift programming language [Електронний ресурс] // Режим доступу до ресурсу: https://docs.swift.org/swift-book/
4. App Design Apprentice (First Edition) By Prateek Prasad: 2021. – 281 с.
5. Документація з використання інструментів firebase [електронний ресурс] // Режим доступу до ресурсу: https://firebase.google.com/docs/firestore
6. Документація з використання фреймворку ReactiveExtension [Електронний ресурс] // Режим доступу до ресурсу: https://reactivex.io/intro.html
7. Шевелюк, М.М. (2021). Цифровізація у сфері туризму: інноваційні тренди і пріоритетні напрями розвитку. *Питання культурології*, (38), 226-235. doi: https://doi.org/10.31866/2410- 1311.38.2021.245956.
8. Звіт Світової ради з подорожей та туризму про економічний впливу туризму на світову економіку [Електронний ресурс] // Режим доступу до ресурсу: https://wttc.org/Research/Economic-Impact/economic-research/economic-impact-analysis

ДОДАТКИ

1. Код застосунку

```
//  AppDelegate.swift

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

        func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
            FirebaseApp.configure()
            return true
        }

        // MARK: UISceneSession Lifecycle

         func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
            return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
        }
    }

    //  SceneDelegate.swift
    class SceneDelegate: UIResponder, UIWindowSceneDelegate {
        var window: UIWindow?
        var dependencies: RouterDependencies?

         func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
```

```
        guard let windowScene = (scene as? UIWindowScene) else { return }

        setupRouterDependencies()

        let window = UIWindow(windowScene: windowScene)
                    let viewController = TabBarViewController(dependencies:
self.dependencies!)
        window.rootViewController = viewController
        self.window = window
        window.makeKeyAndVisible()
    }

    func sceneDidDisconnect(_ scene: UIScene) {
        discarded (see `application:didDiscardSceneSessions` instead).
    }

    private func setupRouterDependencies() {
        let configService = FirebaseConfigService()
        let databaseService = STFirebaseDatabaseService()
        self.dependencies = RouterDependencies(configService: configService,
                                    databaseService: databaseService)
    }
}
//  TabBarViewController.swift

protocol TabBarControllerProtocol: UIViewController {
    func configure(dependencies: RouterDependencies)
}

enum TabBarDataSource: Int, CaseIterable {
    case map
```

```
case news
case settings

var icon: UIImage? {
    switch self {
    case .map:
        return UIImage(systemName: "map")
    case .news:
        return UIImage(systemName: "newspaper")
    case .settings:
        return UIImage(systemName: "gear")
    }
}

var itemName: String {
    switch self {
    case .map:
        return "Map"
    case .news:
        return "News"
    case .settings:
        return "Settings"
    }
}

var scene: TabBarControllerProtocol.Type {
    switch self {
    case .map:
        return MapViewController.self
    case .news:
        return NewsViewController.self
```

```
            case .settings:
                return SettingsViewController.self
            }
        }

            func navigationController(with viewController: UIViewController) -> UINavigationController {
            let controller: UINavigationController
            switch self {
            default:
                                controller = UINavigationController(rootViewController: viewController)
            }
            return controller
        }

        static let items: [TabBarDataSource] = [.map, .news, .settings]
    }

    class TabBarViewController: UITabBarController {
        private let dependencies: RouterDependencies

        init(dependencies: RouterDependencies) {
            self.dependencies = dependencies
            super.init(nibName: nil, bundle: nil)
            setupViewControllers()
        }

        required init?(coder: NSCoder) {
            return nil
        }
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        setupView()
    }

    func setupViewControllers() {
        let controllers = TabBarDataSource.items.map { item -> UIViewController
in
            let viewController = item.scene.init()
            viewController.configure(dependencies: dependencies)
            let navItem = item.navigationController(with: viewController)
            navItem.tabBarItem.title = item.itemName
            navItem.tabBarItem.image = item.icon
            return navItem
        }
        viewControllers = controllers
    }

    func setupView() {
        view.backgroundColor = .systemBackground
        tabBar.backgroundColor = .white.withAlphaComponent(0.45)
        tabBar.tintColor = .label
    }
}

//  MapViewController.swift

class MapViewController: UIViewController, TabBarControllerProtocol {
    private let contentView = MapView()
    private var viewModel: MapViewModel!
```

```
    private var locations: [LocationModel] = []
    private var userLocation: CLLocationCoordinate2D?
    private let disposeBag = DisposeBag()

    func configure(dependencies: RouterDependencies) {
        self.viewModel = STMapViewModel(dependencies: dependencies)
    }

    override func loadView() {
        super.loadView()
        self.view = contentView
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setupMap()

            let longGesture = UILongPressGestureRecognizer(target: self, action:
#selector(createNewAnnotation(_:)))
        contentView.mapView.addGestureRecognizer(longGesture)

        viewModel.locations.subscribe(onNext: { [weak self] locations in
            guard let self = self, let locations = locations else { return }
            DispatchQueue.main.async {
                self.loadData(locations)
            }
        }).disposed(by: disposeBag)
    }

    func showLocationDetail(location: LocationModel) {
        let vc = LocationDetailViewController()
```

```
            vc.configure(location: location)
            self.navigationController?.pushViewController(vc, animated: true)
        }

        @objc
        func createNewAnnotation(_ sender: UIGestureRecognizer) {
            let touchPoint = sender.location(in: self.contentView.mapView)
                        let coordinates = contentView.mapView.convert(touchPoint,
toCoordinateFrom: self.contentView.mapView)
            let heldPoint = MKPointAnnotation()
            heldPoint.coordinate = coordinates
            if (sender.state == .began) {
                heldPoint.title = "Set Point"
                  heldPoint.subtitle = String(format: "%.4f", coordinates.latitude) + "," +
String(format: "%.4f", coordinates.longitude)
                contentView.mapView.addAnnotation(heldPoint)
            }
            createAlert(coordinates: coordinates)
            sender.state = .cancelled
        }

        func createAlert(coordinates: CLLocationCoordinate2D) {
              let alert = UIAlertController(title: "Do you really want to suggest new
location?", message: nil, preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Yes", style: .default, handler: { _ in
                self.sendEmail(coordinates: coordinates)
            }))
            alert.addAction(UIAlertAction(title: "No", style: .cancel, handler: { _ in
                self.removeAnnotation()
            }))
            present(alert, animated: true)
```

```
        }

        func sendEmail(coordinates: CLLocationCoordinate2D) {
            if MFMailComposeViewController.canSendMail() {
                let mail = MFMailComposeViewController()
                mail.mailComposeDelegate = self
                mail.setToRecipients(["andrea.awesome.ua@gmail.com"])
                 mail.setMessageBody("<p>Here is coordinates for new location!</p></
br><p>latitude: \(coordinates.latitude) <br> longitude: \(coordinates.longitude)</p>",
isHTML: true)

                present(mail, animated: true)
            }
        }

        func removeAnnotation() {
            contentView.mapView.annotations.forEach {
                if ($0.title ?? "") == "Set Point" {
                    self.contentView.mapView.removeAnnotation($0)
                }
            }
        }
    }

    extension MapViewController: MFMailComposeViewControllerDelegate {
        func mailComposeController(_ controller: MFMailComposeViewController,
didFinishWith result: MFMailComposeResult, error: Error?) {
            removeAnnotation()
            controller.dismiss(animated: true)
        }
    }
```

```
extension MapViewController: MKMapViewDelegate {
    func setupMap() {
        self.contentView.mapView.delegate = self
        self.contentView.mapView.showsUserLocation = true
                    let centerCoordinate = CLLocationCoordinate2D(latitude:
50.91458471412202, longitude: 34.803296391480565)
        let coordinate = MKCoordinateRegion(center: centerCoordinate,
                                latitudinalMeters: CLLocationDistance(10000),
                                longitudinalMeters: CLLocationDistance(10000))
        self.contentView.mapView.setRegion(coordinate, animated: false)
    }

    //MARK: load locations data
    func loadData(_ locations: [LocationModel]) {
        self.locations = locations
        self.showAnnotations()
        self.contentView.mapView.reloadInputViews()
    }

    private func showAnnotations() {
        locations.forEach { item in
            let annotation = MKPointAnnotation()
            annotation.coordinate = CLLocationCoordinate2D(latitude: item.latitude,
longitude: item.longitude)
            annotation.title = item.title
            self.contentView.mapView.addAnnotation(annotation)
        }
    }
```

```
func mapView(_ mapView: MKMapView, didUpdate userLocation: MKUserLocation) {
    guard self.userLocation == nil else {
        return
    }
    self.userLocation = userLocation.coordinate
    mapView.setCenter(userLocation.coordinate, animated: true)
}

func mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView, calloutAccessoryControlTapped control: UIControl) {
    let loc = locations.first { (item) -> Bool in
        return item.title == view.annotation?.title
    }
    guard let location = loc else { return }
    showLocationDetail(location: location)
}

func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {
    let title = annotation.title!!
    var annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: title)
    _ = locations.first { (item) -> Bool in
        return item.title == title
    }
    if annotationView == nil {
        annotationView = MKAnnotationView(annotation: annotation, reuseIdentifier: title)
        annotationView?.canShowCallout = true
        annotationView?.image = UIImage(systemName: "mappin")
```

```
            annotationView?.scalesLargeContentImage = true
                                        annotationView?.rightCalloutAccessoryView =
UIButton(type: .detailDisclosure)
        } else {
            annotationView?.annotation = annotation
        }

        return annotationView
    }
}

//  MapView.swift

final class MapView: UIView {
    let mapView = MKMapView()

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupLayout()
    }

    required init?(coder: NSCoder) {
        return nil
    }

    private func setupLayout() {
        addSubview(mapView)
        mapView.snp.makeConstraints {
            $0.edges.equalToSuperview()
        }
    }
```

```
    }

    //  MapViewModel.swift

    protocol MapViewModel {
        var locations: BehaviorRelay<[LocationModel]?> { get }
    }

    class STMapViewModel: MapViewModel {
        private let configService: ConfigService
        private let databaseService: FirebaseDatabaseService
                var locations: BehaviorRelay<[LocationModel]?> =
BehaviorRelay<[LocationModel]?>.init(value: nil)

        init(dependencies: RouterDependencies) {
            self.configService = dependencies.configService
            self.databaseService = dependencies.databaseService

            self.databaseService.getLocations { locations in
                self.locations.accept(locations)
            }
        }
    }
    //  LocationDetailViewController.swift

    class LocationDetailViewController: UIViewController {
        private let contentView = LocationDetailView()

        func configure(location: LocationModel) {
            contentView.setup(location: location)
        }
```

```
    func configure(news: NewsModel) {
        contentView.setup(news: news)
    }

    override func loadView() {
        super.loadView()
        self.view = contentView
    }
}

//  LocationDetailView.swift

final class LocationDetailView: UIView {
    private let imageView = UIImageView()
    private let title = UILabel()
    private let subtitle = UILabel()

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupLayout()
    }

    required init?(coder: NSCoder) {
        return nil
    }

    private func setupLayout() {
        backgroundColor = .white

        let scrollView = UIScrollView()
```

```
addSubview(scrollView)
scrollView.snp.makeConstraints {
    $0.edges.equalTo(safeAreaLayoutGuide)
}

let contentView = UIView()
scrollView.addSubview(contentView)
contentView.snp.makeConstraints {
    $0.top.bottom.equalToSuperview()
    $0.leading.trailing.equalTo(self)
}

imageView.contentMode = .scaleToFill
contentView.addSubview(imageView)
imageView.snp.makeConstraints {
    $0.top.leading.trailing.equalToSuperview()
    $0.height.lessThanOrEqualTo(self).multipliedBy(0.35)
}

title.numberOfLines = 0
title.font = .boldSystemFont(ofSize: 22)
title.textColor = .black
contentView.addSubview(title)
title.snp.makeConstraints {
    $0.top.equalTo(imageView.snp.bottom).offset(16)
    $0.leading.trailing.equalToSuperview().inset(24)
}

subtitle.numberOfLines = 0
subtitle.font = .systemFont(ofSize: 18)
subtitle.textColor = .black
```

```
        contentView.addSubview(subtitle)
        subtitle.snp.makeConstraints {
            $0.top.equalTo(title.snp.bottom).offset(16)
            $0.leading.trailing.equalToSuperview().inset(24)
            $0.bottom.equalToSuperview().inset(24)
        }
    }

    func setup(location: LocationModel) {
        let imageUrl = URL(string: location.image)
        imageView.kf.setImage(with: imageUrl)
        title.text = location.title
        subtitle.text = location.content
    }

    func setup(news: NewsModel) {
        let imageUrl = URL(string: news.image)
        imageView.kf.setImage(with: imageUrl)
        title.text = news.title
        subtitle.text = news.content
    }
}

//  NewsViewController.swift

class NewsViewController: UIViewController, TabBarControllerProtocol {
    private var viewModel: NewsViewModel!
    private let contentView = NewsView()
    private let disposeBag = DisposeBag()

    func configure(dependencies: RouterDependencies) {
```

```
    viewModel = STNewsViewModel(dependencies: dependencies)
}

override func loadView() {
    super.loadView()
    self.view = contentView
}

override func viewDidLoad() {
    super.viewDidLoad()
    navigationItem.title = "Breaking news"

    viewModel.newsItems
        .bind(to: contentView.tableView.rx
            .items(cellIdentifier: NewsTableCell.identifier,
                   cellType: NewsTableCell.self)) { index, news, cell in
                cell.setup(news: news)
        }.disposed(by: contentView.disposeBag)

    contentView.tableView.rx
        .modelSelected(NewsModel.self)
        .subscribe(onNext: { [weak self] model in
            guard let self = self else { return }
            self.showNewsDetail(news: model)
    }).disposed(by: disposeBag)
}

func showNewsDetail(news: NewsModel) {
    let vc = LocationDetailViewController()
    vc.configure(news: news)
    self.navigationController?.pushViewController(vc, animated: true)
```

```
    }
}

// NewsView.swift

final class NewsView: UIView {
    let tableView = UITableView()
    let disposeBag = DisposeBag()

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupLayout()
    }

    required init?(coder: NSCoder) {
        return nil
    }

    private func setupLayout() {
        backgroundColor = .white

                    tableView.register(NewsTableCell.self, forCellReuseIdentifier:
NewsTableCell.identifier)
        addSubview(tableView)
        tableView.snp.makeConstraints {
            $0.edges.equalTo(safeAreaLayoutGuide)
        }
    }
}

// NewsTableCell.swift
```

```
final class NewsTableCell: UITableViewCell {
    static let identifier = String(describing: NewsTableCell.self)

    private let newsImageView = UIImageView()
    private let title = UILabel()
    private let subtitle = UILabel()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        setupLayout()
    }

    required init?(coder: NSCoder) {
        return nil
    }

    private func setupLayout() {
        selectionStyle = .none

        newsImageView.contentMode = .scaleAspectFill
        newsImageView.clipsToBounds = true
        contentView.addSubview(newsImageView)
        newsImageView.snp.makeConstraints {
            $0.leading.top.bottom.equalToSuperview()
            $0.width.equalToSuperview().multipliedBy(0.3)
            $0.height.equalTo(120)
        }

        title.numberOfLines = 2
        title.font = .boldSystemFont(ofSize: 16)
```

```
        title.textColor = .black
        contentView.addSubview(title)
        title.snp.makeConstraints {
            $0.top.equalToSuperview().inset(15)
            $0.leading.equalTo(newsImageView.snp.trailing).offset(12)
            $0.trailing.equalToSuperview().inset(24)
        }

        subtitle.font = .systemFont(ofSize: 14)
        subtitle.numberOfLines = 3
        subtitle.textColor = .black
        contentView.addSubview(subtitle)
        subtitle.snp.makeConstraints {
            $0.top.equalTo(title.snp.bottom).offset(15)
            $0.leading.equalTo(newsImageView.snp.trailing).offset(12)
            $0.trailing.equalToSuperview().inset(24)
            $0.bottom.lessThanOrEqualToSuperview().inset(15)
        }
    }

    func setup(news: NewsModel) {
        let url = URL(string: news.image)
        newsImageView.kf.setImage(with: url)
        title.text = news.title
        subtitle.text = news.content
    }
}

//  NewsViewModel.swift

protocol NewsViewModel {
```

```
    var newsItems: BehaviorRelay<[NewsModel]> { get }
}

class STNewsViewModel: NewsViewModel {
    private let configService: ConfigService
    private let databaseService: FirebaseDatabaseService
    let newsItems = BehaviorRelay<[NewsModel]>.init(value: [])

    init(dependencies: RouterDependencies) {
        self.configService = dependencies.configService
        self.databaseService = dependencies.databaseService

        self.databaseService.getNews { news in
            self.newsItems.accept(news)
        }
    }
}

//  SettingsViewController.swift

enum SettingsDestination {
    case language

    var title: String {
        switch self {
        case .language:
            return "Change language"
        }
    }

    static var dataSource: [SettingsDestination] {
```

```
            return [.language]
        }
    }

    class SettingsViewController: UIViewController, TabBarControllerProtocol {
        private let dataSource = BehaviorRelay<[SettingsDestination]>.init(value:
SettingsDestination.dataSource)

        private let contentView = SettingsView()

        override func loadView() {
            super.loadView()
            self.view = contentView
        }

        func configure(dependencies: RouterDependencies) {

        }

        override func viewDidLoad() {
            super.viewDidLoad()

            navigationItem.title = "Settings"

            dataSource
                .bind(to: contentView.tableView.rx
                    .items(cellIdentifier: EmptyCell.identifier,
                           cellType: EmptyCell.self)) { index, setting, cell in
                    cell.title.text = setting.title
                }.disposed(by: contentView.disposeBag)
```

```
            contentView.tableView.rx
                .modelSelected(SettingsDestination.self)
                .subscribe(onNext: { [weak self] model in
                    guard let self = self else { return }
                    switch model {
                    case .language:
                                                    UIApplication.shared.open(URL(string:
UIApplication.openSettingsURLString)!, options: [:], completionHandler: nil)
                    }
                }).disposed(by: contentView.disposeBag)
        }
    }

    //  SettingsView.swift

    final class SettingsView: UIView {
        let tableView = UITableView()
        let disposeBag = DisposeBag()

        override init(frame: CGRect) {
            super.init(frame: frame)
            setupLayout()
        }

        required init?(coder: NSCoder) {
            return nil
        }

        private func setupLayout() {
                            tableView.register(EmptyCell.self,   forCellReuseIdentifier:
EmptyCell.identifier)
```

```
        addSubview(tableView)
        tableView.snp.makeConstraints {
            $0.edges.equalTo(safeAreaLayoutGuide)
        }
    }
}

//  EmptyCell.swift

class EmptyCell: UITableViewCell {
    static let identifier = String(describing: EmptyCell.self)

    let title = UILabel()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        setupLayout()
    }

    required init?(coder: NSCoder) {
        return nil
    }

    private func setupLayout() {
        selectionStyle = .none

        title.font = .systemFont(ofSize: 21)
        addSubview(title)
        title.snp.makeConstraints {
            $0.top.bottom.equalToSuperview().inset(10)
            $0.leading.equalToSuperview().inset(20)
```

```
        }
    }
}

//  RouterDependencies.swift

struct RouterDependencies {
   let databaseService: FirebaseDatabaseService
}

//  FirebaseDatabaseService.swift

protocol FirebaseDatabaseService {
    func getNews(completion: @escaping([NewsModel]) -> ())
    func getLocations(completion: @escaping([LocationModel]) -> ())
}

class STFirebaseDatabaseService: FirebaseDatabaseService {
    let db = Firestore.firestore()

    func getNews(completion: @escaping([NewsModel]) -> ()) {
        db.collection("News").getDocuments { query, error in
            guard let query = query else {
                return
            }
            let dataArray = query.documents.map { document -> [NewsModel] in
                let data = document.data()["news"] as? Array<[String: Any]>
                var array: [NewsModel] = []
                data?.forEach { news in
                    let id = (news["id"] as? Int) ?? 0
                    let title = (news["title"] as? String) ?? ""
```

```
                let content = (news["content"] as? String) ?? ""
                let image = (news["image"] as? String) ?? ""
                    let newsItem = NewsModel(id: id, title: title, content: content,
image: image)
                array.append(newsItem)
            }
            return array
        }
        let item = dataArray[0]
        completion(item)
    }
}

    func getLocations(completion: @escaping([LocationModel]) -> ()) {
        db.collection("Locations").getDocuments { query, error in
            guard let query = query else {
                return
            }
            let dataArray = query.documents.map { document -> [LocationModel] in
                let data = document.data()["locations"] as? Array<[String: Any]>
                var array: [LocationModel] = []
                data?.forEach { location in
                    let id = (location["id"] as? Int) ?? 0
                    let title = (location["title"] as? String) ?? ""
                    let content = (location["content"] as? String) ?? ""
                    let image = (location["image"] as? String) ?? ""
                    let latitude = (location["latitude"] as? Double) ?? 0
                    let longitude = (location["longitude"] as? Double) ?? 0
                        let newsItem = LocationModel(id: id, title: title, content: content,
latitude: latitude, longitude: longitude, image: image)
                    array.append(newsItem)
```

```
                }
                return array
            }
            let item = dataArray[0]
            completion(item)
        }
    }
}

// CLLocationCoordinate2DExtensions.swift

extension CLLocationCoordinate2D {
    func distance(from: CLLocationCoordinate2D) -> CLLocationDistance {
                                                                    let
destination=CLLocation(latitude:from.latitude,longitude:from.longitude)
        return CLLocation(latitude: latitude, longitude: longitude).distance(from:
destination)
    }
}
```

2. Приклад коду SwiftUI

```
struct ContentView: View {
    var text1 = "some text"
    var text2 = "some more text"
    var body: some View {
        VStack{
            Text(text1)
                .padding()
                .frame(width: 100, height: 50)
            Text(text2)
                .background(Color.gray)
```

```
                .border(Color.green)
        }
    }
}
```