МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: " Мобільний застосунок для організації бізнес заходів "

Виконавець: Грушецький Назарій Ігорович

Керівник: к.т.н., доцент Моденов Юрій Борисович

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____" _____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Грушецького Назарія Ігоровича
(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Мобільний застосунок для організації бізнес заходів» затверджена наказом

. ректора від 29.09.2023 за № 1976/ст.

2. **Термін виконання роботи:** з 02.10.2023 по 31.12.2023.

3. **Вихідні дані до роботи:** теоретичні та практичні відомості та основи створення сучасного мобільного додатку на базі iOS.

4. **Зміст пояснювальної записки:** вступ, опис процесу розробки мобільного застосунку, розгляд інструментів та фреймворків, опис та порівняння стандартної архітектури запропонованної Apple та MVP, демонстрація структури та функціоналу додатка.

5. **Перелік обов'язкового ілюстративного матеріалу:** слайди, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз літературних джерел	02.10.2023 20.10.2023	
2.	Створення плану дипломного проекту	21.10.2023 01.11.2023	
3.	Консультація з науковим керівником	02.11.2023 14.11.2023	
4.	Розробка розділу «Основні етапи розробки мобільного додатку»	15.11.2023 25.11.2023	
5.	Розробка розділу «Інструментарій»	26.11.2023 01.12.2023	
6.	Розробка розділу «Демонстрація мобільного додатку»	02.12.2023 10.12.2023	
7.	Оформлення дипломного проекту	11.12.2023 31.12.2023	

7. Дата видачі завдання: 02.10.2023р.

Керівник дипломної роботи Юрій МОДЕНОВ
(підпис керівника)

Завдання прийняв до виконання Назарій ГРУШЕЦЬКИЙ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Мобільний застосунок для організації бізнес заходів ” складається із вступу, чотирьох розділів, загальних висновків, списку використаних джерел і містить 63 сторінки тексту та 33 рисунка. Список використаних джерел містить 11 найменувань

Метою дипломної роботи є теоретичий та практичний розгляд розробки мобільного застосунку для системи IOS на основі мови програмування – Swift, основною ціллю додатку є спрощення організації бізнес заходів за допомогою доступного та зрозумілого інтерфейсу.

Об’єктом дослідження організація бізнес заходів за допомогою застосунку.

Предметом дослідження є розробка мобільного застосунку для організації бізнес заходів, що охоплює собою не лише визначення архітектури додатку, розробку серверної частини, API, розробку інтерфейсу, тестування, запуск та підтримку, але і підготовчу частину, яка відповідає за аналіз ідеї додатку та визначення умов забезпечення безпеки користування та підписання угоди нерозголошення.

Ключові слова: SWIFT, МОБІЛЬНИЙ ЗАСТОСУНОК, ІНТЕРФЕЙС, ФРЕЙМОВОРК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UI – *User Interface* – Інтерфейс користувача

VC – *View Contoller* – Контролер виду.

UX – *User Experience* – Досвід користувача.

TV – *Table View* – Вид таблиці.

NC – *Navigation Controller* – Контролер навігації.

ВСТУП

Сучасний темп життя вимагає, щоб ми планували кожен момент свого існування. Коли йдеться про організацію не лише власного часу, а й планування майбутніх подій і конференцій, це стає викликом через потребу ураховувати не лише свій графік, але й плани інших людей.

Ефективним рішенням цієї задачі є використання спеціалізованого додатку для планування, який об'єднує всі необхідні функції, такі як визначення місця, дати, часу та кількості учасників події. Такий застосунок повинен враховувати не лише індивідуальні вподобання, але й думку всієї групи. Розуміється, що це завдання пов'язане із певними труднощами, оскільки воно обов'язково враховує як технічні, так і теоретичні та соціальні аспекти.

Створення такого додатку потребує детального планування, яке охоплює як технічні, так і теоретичні аспекти, починаючи від етапу ідеї і закінчуючи втіленням в життя. У процесі розробки мобільного додатку для iOS виникає ідея та мотивація, які переходять до визначення типу додатку та його вимог. Робота над серверною частиною і дизайном інтерфейсу поступово переходить до створення прототипу та тестування. Заключним етапом є публікація готового додатку.

РОЗДІЛ 1.
ОСНОВНІ ЕТАПИ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

Згідно з даними Statista, річний дохід від додатків досяг 111 мільярдів доларів, що свідчить про значне зростання кількості програм, доступних у Play Store та App Store.

Значно раніше, аніж це стало стандартом, поняття, які раніше з'являлися у науково-фантастичних фільмах чи книгах, стали повсякденністю в різних аспектах нашого життя.

Чи йдеться про розширену реальність (AR), віртуальну реальність (VR), машинне навчання чи Інтернет речей, ці технології революціонізують кожну сферу.

1.1. Визначення мобільного застосунку

Мобільний додаток є конкретним видом програмного забезпечення, призначеного для використання на портативних електронних пристроях, таких як смартфони чи планшети. Зазвичай ці додатки характеризуються обмеженим обсягом та функціональністю. Поширення концепції використання мобільних додатків спочатку стало можливим завдяки ініціативі компанії Apple та запуску її магазину додатків, який пропонує розмаїття застосунків.

Початковий мобільний додаток надавав загальну інформацію та інформаційні послуги через глобальну мережу. Серед його можливостей були електронна пошта, календар, фондовий ринок і інформація про погоду. Проте, зі зростанням попиту користувачів мобільних пристроїв, можливості розробки мобільних додатків розширилися на різні категорії, такі як мобільні ігри, автоматизація виробництва, GPS та інші.

Зі збільшенням числа та різноманітності застосувань виникло велике розмаїття різних сфер. Багато сервісів тепер використовують технології мобільних додатків, такі як визначення місця розташування, інтернет-банкінг, відстеження, покупки квитків і навіть мобільні медичні послуги.

Починаючи як просте перенесення програм з ПК на мобільний пристрій, розвиток мобільних додатків тепер базується на сучасних стратегіях. Цей розвиток включає специфічний підхід до мобільного середовища, враховуючи його обмеження та переваги. Наприклад, додатки, спеціально розроблені для мобільних пристроїв, ураховують особливості їхньої роботи, враховуючи, що користувач може знаходитися в різних місцях, не обмежений конкретною локацією, як це може бути в разі використання ПК.

1.2. Основні типи мобільних застосунків

Нативний мобільний додаток - це програма, яка спеціально розробляється та оптимізується для конкретної платформи пристроїв, такої як Android або iOS (рис. 1.1). Цей тип додатка використовує спеціалізовану мову програмування, яка має доступ до функцій та інтерфейсів конкретної платформи.

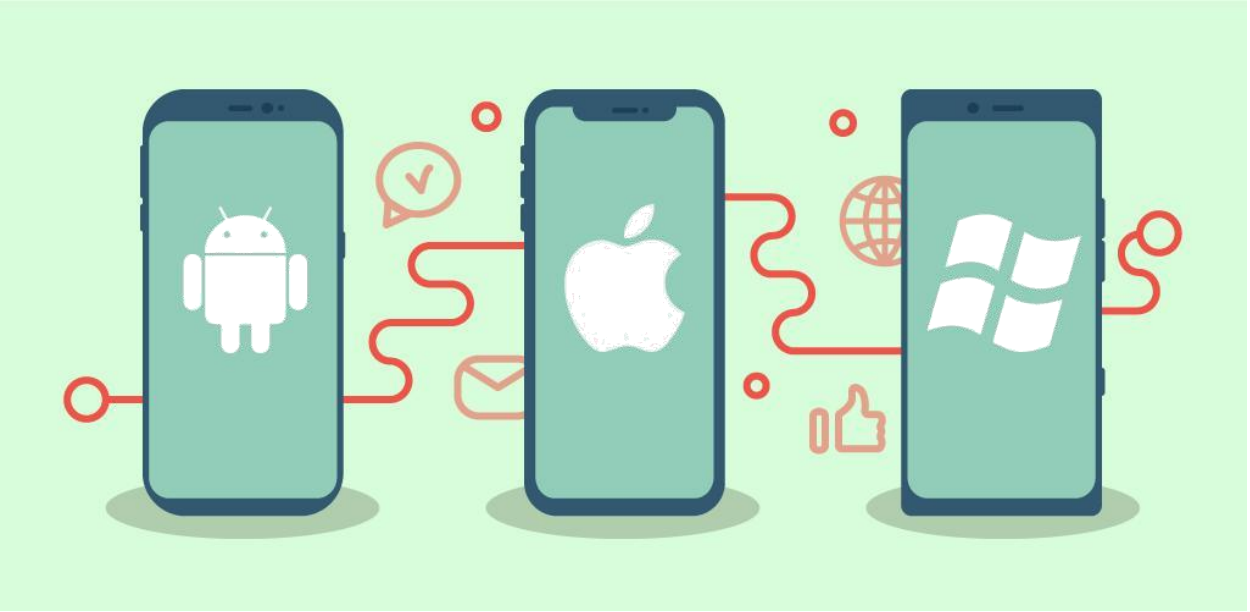

Рис. 1.1. Нативний додаток

Нативні мобільні додатки мають ряд значущих переваг. Однією з ключових переваг є високоякісний користувацький інтерфейс. Дизайнери, які працюють над такими додатками, використовують свої власні пристрої для створення інтерфейсу користувача, що гарантує його якість та оптимальність.

Крім того, нативні додатки мають доступ до різноманітних API-інтерфейсів, що прискорює процес розробки та розширює можливості додатків. Благодаря своєму високоякісному інтерфейсу та функціональності, нативні додатки забезпечують користувачам оптимальний досвід використання.

Навпаки, веб-додатки є програмами, які взаємодіють з власними мобільними додатками та функціонують на мобільних пристроях. Вони, як правило, використовують браузери та побудовані на CSS, HTML або JavaScript. Ці додатки перенаправляють користувача на URL-адресу та пропонують вибрати відповідний додаток. З цієї причини вони вимагають менше обсягу пам'яті та дозволяють користувачам зберігати їх у вигляді закладок для подальшого використання.

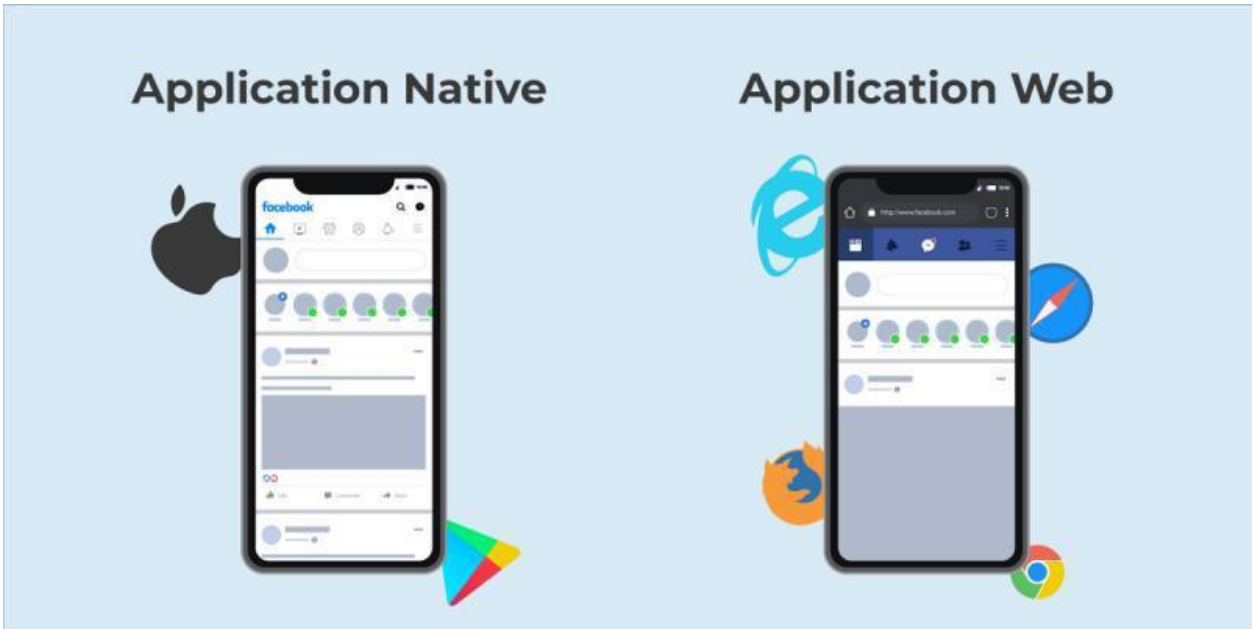

Рис. 1.2. Нативний та веб додатки

Веб-додатки використовують подібний метод організації, що і нативні додатки, проте до них можна отримати доступ через браузер веб-сайту на мобільному пристрої (див. рис. 1.2).

Основна відмінність полягає в тому, що веб-додатки не є незалежними програмами, які встановлюються на пристрій через завантаження коду. Фактично, веб-додатки представляють собою адаптивні веб-сайти, які модифікують свій інтерфейс користувача відповідно до конкретного пристрою клієнта.

1.3. Визначення розробки мобільних додатків

З технічної точки зору, розробка мобільного додатка – це процес або процедура створення програмного забезпечення, призначеного для роботи на смартфонах. Після цього додаток завантажується та встановлюється користувачем через магазини додатків.

Для успішної розробки мобільного додатка необхідно враховувати різноманітні фактори, такі як розмір екрану, функціональність, платформа розробки додатків, дизайн та інші. На даному етапі часто виникає необхідність залучення

фахівців з розробки мобільних додатків, які складають команду для вирішення поставлених завдань (див. рис. 1.4).

Для створення мобільного додатка необхідно враховувати безліч факторів, таких як розмір екрану, функції, платформа розробки додатків, дизайн і так далі. Відтепер необхідно найняти фірму з розробки мобільних додатків, у якої є ціла команда для роботи над вимогами (рис. 1.4.).

Рис. 1.4. Процес розробки мобільного додатку

1.4. Ідея застосунку

Кожен етап розробки програми розпочинається з концепції. Для чіткого розуміння застосування програми необхідно провести інтенсивний аналіз та наукові дослідження. Якщо ідея програми зрозуміла, можна перейти до наступного етапу.

У випадку сумнівів слід відповісти на п'ять ключових питань:

- Яка головна мета додатка?
- Які проблеми вирішує додаток?
- Хто є цільовою аудиторією?
- Чим буде відрізнятися програма від інших?
- Чому користувачам слід продовжувати використовувати додаток?

Тільки після відповідей на ці запитання варто розробляти стратегію створення додатка та проводити дослідження ринку.

Необхідно отримати інформацію про поточні ринкові тенденції та проблеми, з якими стикається цільова аудиторія. Лише після цього можна приступити до вирішення проблем користувачів через розробку мобільного додатка, вивчивши його переваги і знаючи його унікальність. Коротше кажучи, важливо детально вивчити конкурентне середовище та знайти своє вигідне місце на ринку.

Процес створення стратегії для майбутнього мобільного додатка можна порівняти з плануванням відпустки. Важливо ретельно вивчити ринок у відповідній галузі, вибрати оптимальні варіанти для програми, розрахувати бюджет і встановити терміни виконання.

1.5. Визначення вимог проекту

Насамперед треба вибрати платформо розробки мобільного додатку, це досягається шляхом визначення цільової аудиторії. Зокрема допоможе короткий огляд відсотка користувачів мобільних пристроїв Android та iOS у всьому світі (рис. 1.5.).

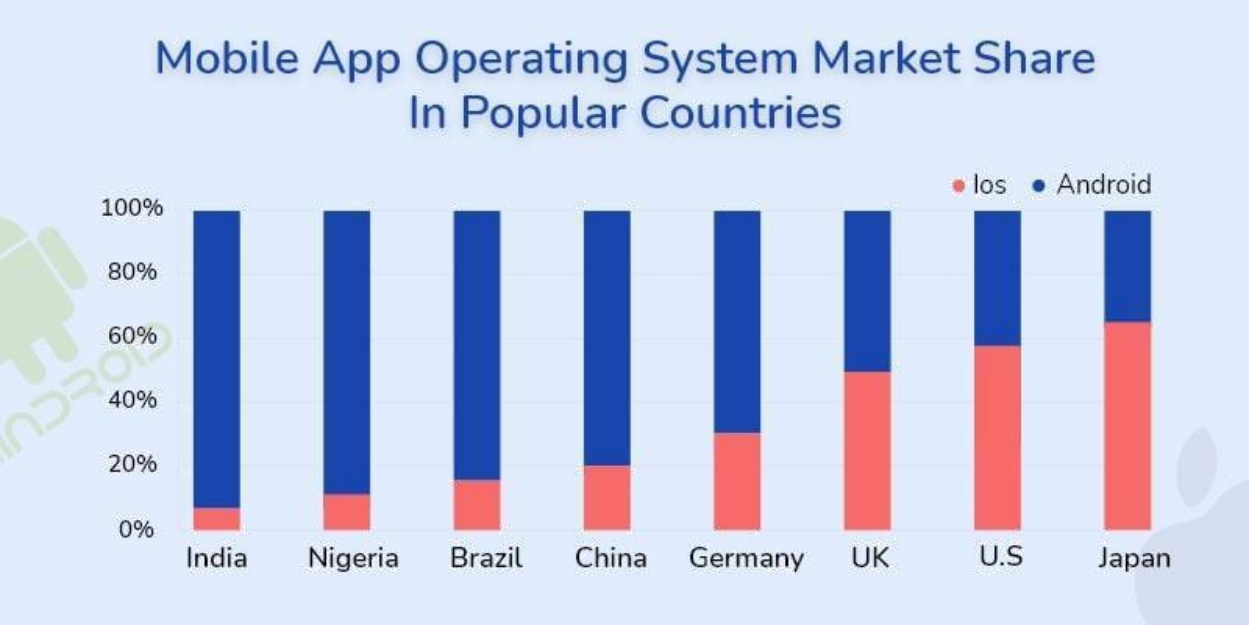

Рис. 1.5. Огляд користувачів мобільних пристроїв Android та iOS у всьому світі

Технологічний стек, також відомий як стек рішень або технологічна інфраструктура, представляє собою перелік всіх технологічних сервісів, які використовуються для створення і запуску програми. Вибір популярного та надійного технологічного стеку є ключовим компонентом для забезпечення безперебійної роботи додатка.

Наступним етапом є розробка функціональних можливостей, функцій та бізнес-моделі для додатка. Різні варіанти бізнес-моделей включають підписку, двосторонній ринок, freemium та покупки в програмі, серед інших. Під час аналізу галузі рекомендується зосередитися на ключових функціях, які додаток надасть користувачам. Співпраця з фахівцями з розробки мобільних додатків дозволить вам визначити технологічний стек.

1.6. Основні етапи розробки програми

Вигляд мобільного додатку є важливим аспектом для користувачів, особливо при висококласній розробці додатків або створенні продукту MVP. Особливо вагомим стає дизайн інтерфейсу (UI/UX). Користувальницький досвід визначається як один з найсуттєвіших аспектів розробки додатка, оскільки безпосередньо впливає на його успішність.

Сфера дизайну мобільних додатків поділяється на кілька відділень, аналогічно процесу розробки додатків. Подібно до розробки, дизайн UI/UX має власний процес та підрозділи.

Перш, ніж переходити до деталей проектування додатків, важливо зрозуміти два ключові терміни дизайну: UI (інтерфейс користувача) і UX (користувальницький досвід).

1.6.1. Каркасний дизайн програми

Каркас направляє весь процес розробки програми і може бути названий схематичною версією майбутнього продукту, каркасною структурою програми чи кресленням програми.

Створення каркасу допомагає команді розробників представити своє бачення клієнтам та прийняти рішення щодо остаточних технічних та дизайнерських рішень.

На відміну від створення ескізу або прототипу програми, каркас продукту є пошуком потенційних дизайнерських та інженерних рішень, а також перевіркою того, чи сприймають клієнти та агентства з розробки майбутній продукт таким чином.

Рис. 1.7. Приклад каркасу

1.6.2. Посібник зі стилю

Посібник із стилю визначається як цілісний набір стандартів дизайну інтерфейсу користувача для елементів інтерфейсу користувача та їх взаємодії з різними продуктами додатків. Це гарантує, що узгодженість підтримується між різними проектними групами та компаніями.

Важливі елементи, включені в посібник із стилю інтерфейсу користувача, включають шрифти, кольори, макети, шаблони дизайну, бібліотеки компонентів, графіку, діалоги, зміни, кнопки та інше.

Посібники із стилю - це більше, ніж просто документація. Вони є основою для прийняття рішень з проектування та реалізації, які впливають на мобільний додаток не лише в інтерфейсі користувача, але і в користувацькому досвіді. Керівництво по стилю повинно бути:

- Докладним і інформаційним: містити точні специфікації та інструкції для кожного елемента.
- Організованим: розділи, компоненти та деталі повинні бути чіткими та груповаными.
- Адаптованим до майбутніх сценаріїв: враховувати майбутні ситуації в керівництві по стилю, такі як можливі розширення макетів або варіанти значків.
- Специфічним для платформи: містити графіку і компоненти в інформаційних стандартах платформи.
- Простим у використанні і розумінні: бути доступним для всіх учасників, включаючи власників продуктів, дизайнерів, розробників та учасників SQA.

1.6.3. Макет застосунку

Макети додатків і веб-сайтів є важливим інструментом, оскільки вони дозволяють створити практично остаточний дизайн мобільного додатка чи веб-сайту, не займаючись складною роботою з розробки функціоналу. Завдяки їх візуальному впливу, макети стають відмінним ресурсом для обміну з клієнтом, щоб вони могли оцінити прийняті дизайнерські рішення та їх вплив на користувальницький інтерфейс перед тим, як розпочати внутрішню роботу над продуктом.

В області дизайну веб-сайтів та мобільних додатків термін "макет" вказує на високоякісне моделювання зовнішнього вигляду веб-сайту чи мобільного додатка. Макети об'єднують в собі структуру та логіку каркасу з високоякісною графікою та елементами інтерфейсу користувача. Таким чином, макети є проміжним етапом у розробці продукту, що має значно більшу візуальну деталізацію, ніж каркас, але ще не має повної функціональності прототипу.

Макети мобільних додатків і веб-сайтів також можуть бути важливими на початковому етапі тестування користувачів. Навіть якщо користувачі не можуть взаємодіяти з макетом, вони можуть висловлювати свої думки щодо дизайну на відносно ранній стадії розробки продукту. Отримання негативних відгуків користувачів чи клієнтів на цьому етапі є гораздо більш ефективним з точки зору економії, ніж отримання їх пізніше.

1.6.4. Прототип застосунку

Прототипи - це цінні інструменти перевірки та тестування, які дозволяють перевірити доцільність концепції програми.

Прототип - це інтерактивний макет мобільного додатка. Він містить ключові користувальницькі інтерфейси, екрани та імітовані функції без будь -якого робочого коду або остаточних елементів дизайну. У порівнянні зі статичним каркасом або макетом, може бути використаний так само, як і будь-який інший додаток.

Прототипування може відповісти на різноманітні питання, допомагаючи визначити ефективність та зручність дизайну та функціоналу програми. Деякі з питань, на які може відповісти прототипування, включають:

Чи працює колірна схема?

- Прототип може відобразити використання кольорів у користувальницькому інтерфейсі та оцінити їх вплив на сприйняття та настрій користувача.

Чи можуть користувачі інтуїтивно орієнтуватися в додатку?

- Прототип може відобразити структуру додатку та його навігацію, перевіряючи, чи можуть користувачі легко знаходити потрібні функції.

Чи краще використовувати два екрани для певної функції, ніж використовувати тільки один?

- Прототип може представити різні варіанти розміщення елементів та функцій, дозволяючи визначити оптимальний дизайн.

Найбільш значущою перевагою прототипування є зменшення ризику невдач та витрат часу і грошей шляхом виявлення можливих проблем та виправлення їх на ранніх етапах проекту.

Прототипування також служить ефективним інструментом для порівняння різних варіантів інтерфейсу та функціоналу, дозволяючи здійснити вибір оптимального рішення. Крім того, цей процес може бути використаний для залучення команди розробників, постачальників та клієнтів, а також для збору пропозицій та вдосконалення концепцій на ранніх етапах розробки.

1.7. Архітектура мобільного застосунку

Мобільна архітектура включає в себе збірку правил, методів і шаблонів, які визначають структуру та організацію мобільного додатка. Головна мета - створення логічних і добре структурованих програм, які відповідають вимогам клієнтів і відповідають галузевим стандартам.

Під час розробки мобільного чи веб-додатка важливо, щоб кожен компонент виконував свою функцію. Навіть невелика проблема на етапі створення архітектури мобільного додатка може мати далекосяжні наслідки для життєздатності кінцевого продукту.

При обговоренні архітектури розробки мобільних додатків архітектори розглядають найкращі галузеві практики та стандарти. Розробники повинні враховувати різні типи бездротових продуктів в цьому аналізі, щоб забезпечити легку роботу додатка як зі смартфонами, так і з планшетами.

З простої точки зору розробки додатки розбиваються на три різних рівні (див. рис. 1.8). Рівень даних включає всі утиліти обробки даних, сервісні агенти та компоненти доступу до даних. Переходячи від даних, бізнес-рівень включає всі різні робочі процеси і бізнес-об'єкти, а також самі бізнес-компоненти. Нарешті, рівень представлення - це місце, де розташовані процеси та компоненти інтерфейсу користувача (UI).

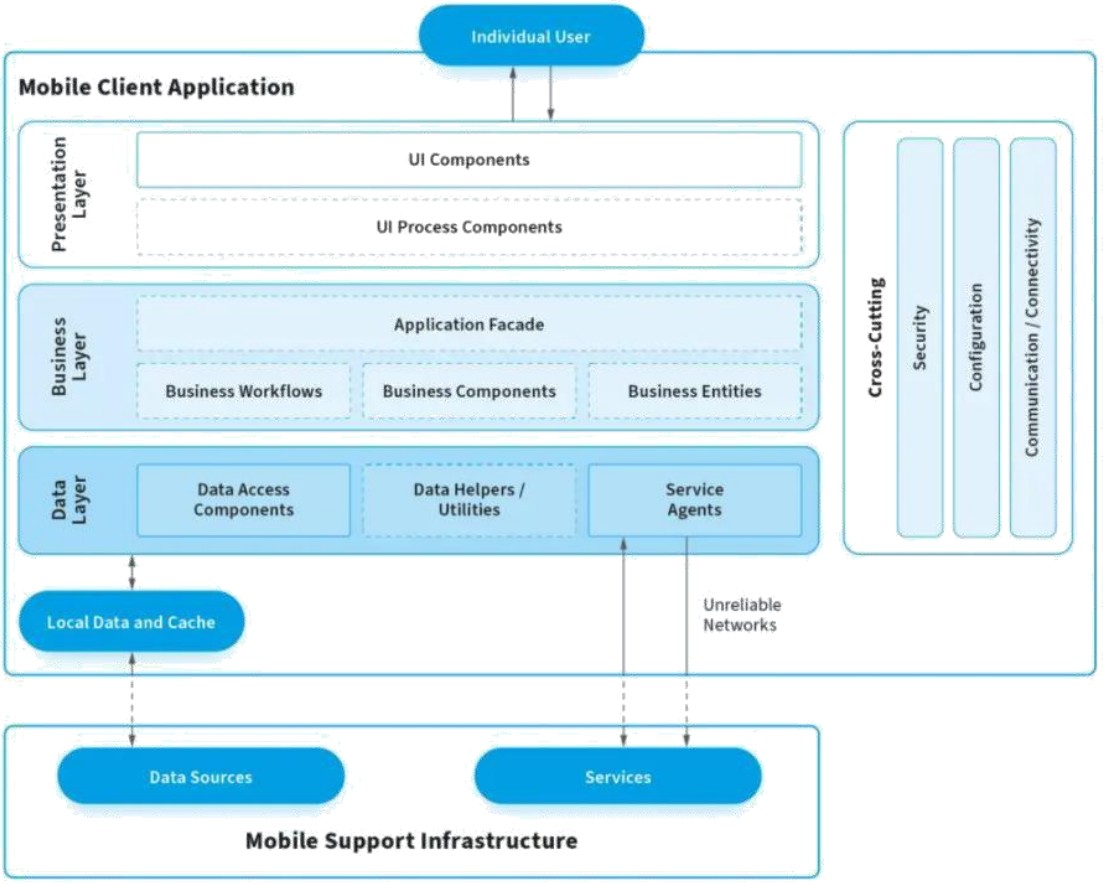

Рис 1.8. Архітектурні рівні

1.7.1. Особливості якісної архітектури

Визначимо деякі ключові особливості якісної архітектури:

- **Збалансований розподіл між ролями:** Для зниження зв'язаності, обов'язки різних об'єктів повинні бути розділені між різними компонентами. Кожен компонент повинен мати єдину відповідальність, що сприяє збалансованому розподілу завдань.
- **Тестованість:** Тестованість включає ефективну стратегію тестування. Архітектура повинна бути побудована так, щоб було можливо ефективно тестувати окремі компоненти та їх взаємодію. Наявність автоматизованих тестів сприяє швидкому виявленню і виправленню помилок.
- **Простота використання і експлуатаційна надійність:** Чим менше коду, тим менше ймовірність помилок. Простота коду полегшує розуміння та обслуговування. Мінімізація складності сприяє підтримці продукту та швидкому впровадженню нових розробників у проект. Збільшення простоти сприяє надійності та стабільності.

Ці особливості допомагають створити архітектуру, яка є не лише ефективною та функціональною, але й легкою у розумінні, тестуванні та підтримці протягом усього циклу розробки.

1.7.2. Огляд Apple MVC

Apple використовує архітектурний шаблон MVC (Model-View-Controller) для розробки користувацького інтерфейсу в їхніх мобільних додатках, зокрема в UIKit, який використовується для розробки iOS-додатків.

Основні принципи цього шаблону реалізовані у власному фреймворку Apple.

1. Модель (Model):

- Опис:

Представляє собою шар, де знаходяться дані додатка.

Містить бізнес-логіку та методи обробки даних.

- Особливості:

Забезпечує управління та доступ до даних.

Часто використовує шаблони проектування, такі як Observable або Delegate, для повідомлення про зміни даних.

2. Представлення (View):

- Опис:

Візуально представляє інтерфейс користувача.

Відповідає за відображення даних та взаємодію з користувачем.

- Особливості:

Не містить бізнес-логіки.

Може бути пере-використаним у різних частинах додатка.

3. Контролер (Controller):

- Опис:

Взаємодіє як з моделлю, так і з представленням.

Керує потоком даних і взаємодії між моделлю та представленням.

- Особливості:

Має деяку бізнес-логіку, але не повинен виконувати завдань, що не відносяться до управління потоком даних.

4. Інші аспекти:

- Повідомлення:

Компоненти можуть взаємодіяти за допомогою системи повідомлень або делегування.

- Оновлення інтерфейсу:

Про зміни даних повідомляє модель, а контролер керує оновленням представлення.

5. Переваги та недоліки:

- Переваги:

Розділення відповідальностей дозволяє легше управляти та змінювати код.

Підтримує багаторазове використання компонентів.

- Недоліки:

Для складних додатків може виникнути проблема перевантаження контролера.

Може бути складним для розробки деяких аспектів великих додатків.

Шаблон MVC дозволяє розробникам вести структурований код та легше управляти взаємодією між компонентами у мобільних додатках Apple.

1.8. Розробка мобільного додатку

Розробка програми складається з трьох основних етапів. Перед тим, як розглядати деталі, ми ретельно аналізуємо процес розробки програми.

Мобільний додаток конструюється на основі трьох ключових компонентів: серверних технологій (Backend), API (інтерфейсів прикладного програмування) та розробки інтерфейсу (Frontend).

Ті елементи, які відображаються користувачам і з якими вони взаємодіють, становлять інтерфейс програми. Розробка, яка відбувається на стороні сервера, відома як бекенд, а набір функцій, що дозволяє додаткам отримувати доступ до даних і взаємодіяти з зовнішніми компонентами програмного забезпечення, операційними системами або мікросервісами, називається API.

1.8.1. Розробка серверної частини

Розробка серверної частини відноситься до серверної частини програми і всього, що взаємодіє між базою даних і додатком.

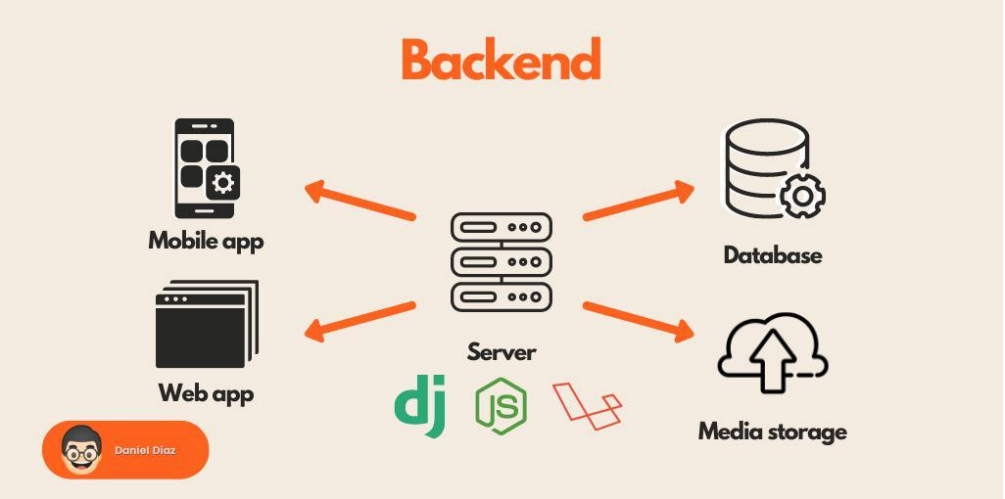

Рис. 1.11. Серверна частина

Серверний розробник - це фахівець, який спеціалізується на створенні, налагодженні та управлінні серверною частиною програмного забезпечення(див. рис. 1.11.). Основні завдання серверного розробника включають розробку серверних застосунків, роботу з базами даних, обробку запитів від клієнтських додатків і забезпечення ефективного обміну даними між клієнтами та серверами.

Основні обов'язки серверного розробника можуть включати:

- Створення серверних додатків: Розробка програмного забезпечення, яке працює на стороні сервера та відповідає за обробку запитів від клієнтів.
- Взаємодія з базами даних: Оптимізація та керування взаємодією серверу з базами даних, включаючи створення та оптимізацію запитів.
- Безпека серверної частини: Забезпечення безпеки серверних додатків, включаючи заходи з обмеження доступу та захист від атак.
- Оптимізація продуктивності: Вдосконалення продуктивності серверних додатків, виявлення та усунення можливих перешкод.
- Розробка та реалізація API: Створення та підтримка API (інтерфейсів прикладного програмування) для взаємодії з клієнтськими додатками.

- Оптимізація мережевої взаємодії: Забезпечення ефективної взаємодії між сервером і клієнтами через мережу.
- Управління конфігурацією та деплоїмент: Керування конфігураціями серверних додатків і проведення їх деплоїменту.

Серверні розробники можуть працювати з різними технологіями та мовами програмування, такими як Node.js, Python, Ruby, Java, а також фреймворками та платформами для створення серверних додатків.

1.8.2. Інтерфейс прикладного програмування

API (інтерфейс прикладного програмування) - це набір визначень та протоколів, які дозволяють різним програмним компонентам взаємодіяти один з одним.

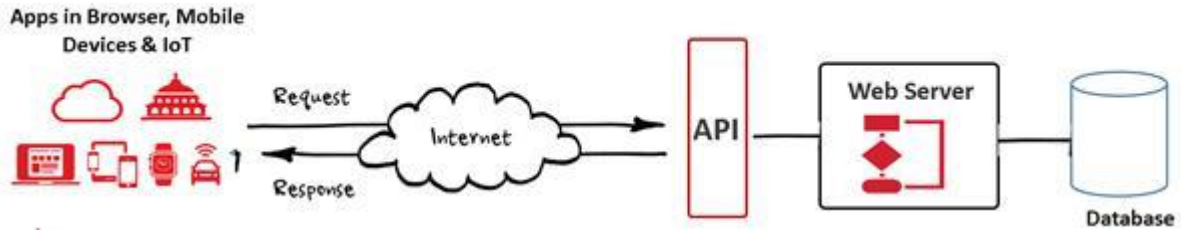

Рис. 1.12. Інтерфейс прикладного програмування

API визначає набір правил та конвенцій для створення програм та визначення методів, які можуть бути використані для обміну даними між програмами.

Основні характеристики API:

- Методи: Визначення доступних дій або операцій, які можна виконати за допомогою API. Це може включати отримання, відправлення, оновлення або видалення даних.
- Параметри: Інформація, яка передається разом з запитом або відповіддю, щоб забезпечити необхідні дані для виконання дій.
- Формати даних: Способи представлення даних в запитах та відповідях. Зазвичай це JSON або XML формати.
- Аутентифікація та безпека: Механізми, які забезпечують безпеку ідентифікації та авторизації, щоб контролювати доступ до ресурсів через API.

1.8.3. Фронт-енд

Розробка інтерфейсу (фронтенд) відноситься до галузі веб-розробки, яка зосереджена на створенні того, що користувачі бачать зі свого боку (рис. 1.13.). Це включає в себе перетворення коду, створеного серверними розробниками, в графічний інтерфейс, що забезпечує представлення даних в зручному для читання і розуміння форматі.

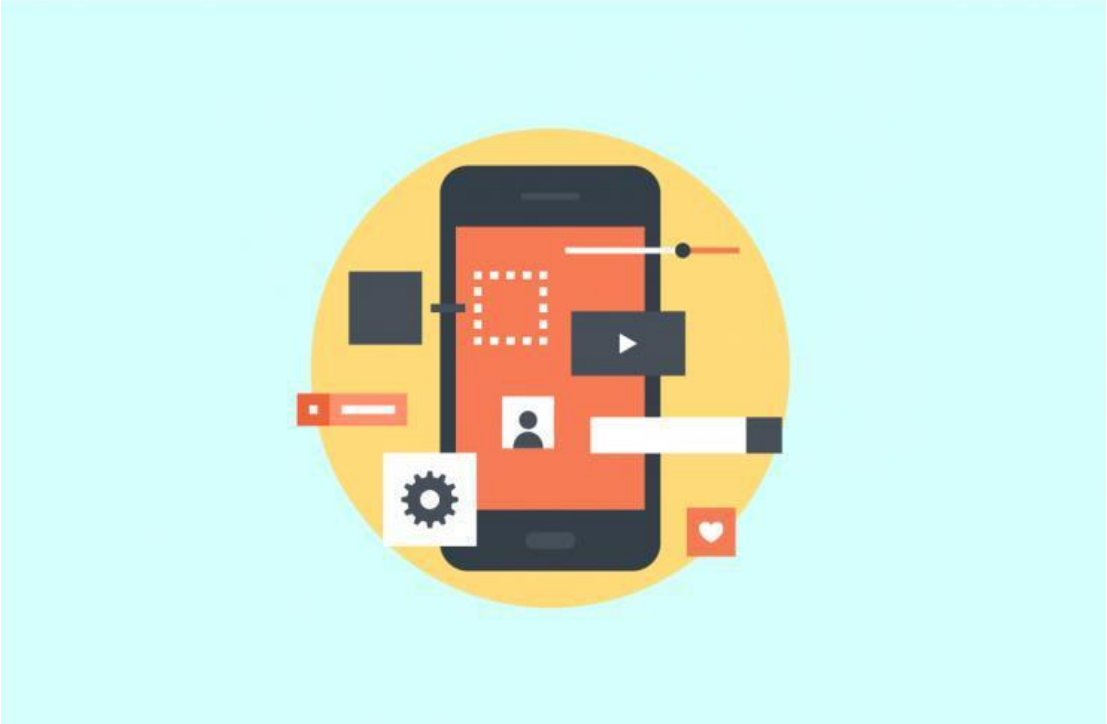

Рис. 1.13. Фронт-енд

Фронтенд-розробники займаються аналізом, проектуванням та вдосконаленням програм, а також забезпечують безперебійну роботу для користувачів.

Розробник інтерфейсу зосереджується на передній частині програми, відповідальної за те, що користувачі бачать і з чим вони взаємодіють. Ці фахівці беруть участь у перетворенні дизайну користувацького інтерфейсу (UI/UX) програми в код, щоб ефективно його втілити та продемонструвати на екрані.

Це все відбувається завдяки розробці інтерфейсу. Технології розробника інтерфейсу для розробки нативних додатків включають Objective C або Swift для

розробки додатків iOS і Java або Kotlin для Android App. Для розробки мобільних додатків сьогодні є ще один варіант — розробка гібридних додатків.

Наприклад, при відкритті додатка YouTube розробник інтерфейсу відповідає за те, щоб забезпечити правильне відображення попередніх переглядів та відео на екрані, а також за налагодження відчуттів користувача під час взаємодії з додатком, таких як візуальні та аудіо ефекти при натисканні на відео.

Це все відбувається завдяки розробці інтерфейсу. Технології розробника інтерфейсу для розробки нативних додатків включають Objective C або Swift для розробки додатків iOS і Java або Kotlin для Android App. Для розробки мобільних додатків сьогодні є ще один варіант — розробка гібридних додатків.

1.9. Тестування мобільних пристроїв

Тестування мобільних пристроїв — це важливий етап в розробці мобільних додатків, оскільки воно спрямоване на перевірку функціональності, продуктивності, безпеки та інших аспектів програмного забезпечення на мобільних пристроях.

Причини чому мобільне тестування є важливим:

- Різноманітність пристроїв і платформ: На ринку існує велика кількість різних мобільних пристроїв, які працюють на різних операційних системах (iOS, Android і т. д.). Тестування дозволяє переконатися, що додаток працює на різних платформах та пристроях без суттєвих проблем.
- Різні версії операційних систем: Виділення та коригування проблем, пов'язаних з різними версіями операційних систем, важливе для того, щоб забезпечити сумісність додатку з новими та старими версіями операційних систем.
- Різні розширення та розміри екрану: Різні мобільні пристрої мають різні розширення та розміри екранів. Тестування дозволяє переконатися, що додаток правильно відображається та працює на різних типах екранів.

- Взаємодія зі смартфонами та планшетами: Тестування дозволяє визначити, як добре додаток взаємодіє з різними форм-факторами, такими як смартфони та планшети.
- Перевірка функціональності: Тестування допомагає виявляти та виправляти помилки в роботі додатка, а також переконується, що всі функції працюють правильно та задовольняють вимоги користувача.
- Безпека даних: Мобільні додатки можуть містити конфіденційні дані користувачів. Тестування важливо для виявлення та виправлення можливих порушень безпеки та забезпечення захисту особистої інформації користувачів.
- Визначення продуктивності: Тестування мобільних пристроїв дозволяє визначити, як швидко та ефективно працює додаток, а також як він впливає на ресурси пристрою, такі як батарея і пам'ять.

Існує кілька видів мобільного тестування, призначених для визначення різних аспектів якості мобільних додатків. Основні види мобільного тестування включають:

Функціональне тестування:

- Одиничні тести (Unit Testing): Тестування окремих частин коду, які можна перевірити ізольовано від інших частин додатку.
- Інтеграційне тестування (Integration Testing): Визначення, як різні компоненти додатку працюють разом під час взаємодії.
- Системне тестування (System Testing): Перевірка роботи всього додатку в цілому для визначення, чи відповідає він вимогам та очікуванням.
- Інтерфейсне тестування:
- UI (User Interface) Testing: Перевірка коректності відображення інтерфейсу користувача на різних пристроях та розширеннях екрану.

- UX (User Experience) Testing: Оцінка загального враження користувача від взаємодії з додатком.

Тестування продуктивності:

- Тестування навантаження (Load Testing): Визначення та перевірка того, як додаток працює під час великого обсягу користувацьких запитань.
- Тестування швидкодії (Performance Testing): Вимірювання часу реакції додатка та його продуктивності при різних умовах.
- Тестування сумісності:
- Тестування на різних платформах (Platform Compatibility Testing): Визначення сумісності додатка з різними операційними системами (iOS, Android, і т.д.).
- Тестування на різних пристроях (Device Compatibility Testing): Перевірка роботи додатка на різних мобільних пристроях.

Тестування безпеки:

- Тестування на проникнення (Penetration Testing): Визначення та виправлення можливих вразливостей у системі для попередження можливого вторгнення.
- Тестування автоматизації:
- Автоматизовані тести (Automated Testing): Використання засобів автоматизації для виконання тестів, зокрема, для прискорення та автоматизації рутинних перевірок.

Висновок до розділу 1

Детально проведено аналіз різних етапів, які варіюються від вибору ідеї і пошуку продукту до доставки і обслуговування додатків. Кожний створений мобільний додаток проходить один і той же життєвий цикл розробки, незважаючи на відмінності такі як платформа використання, будь то – IOS чи Android.

Було проведено аналіз процесу розробки мобільного додатку. Додатки є результатом детального і складного процесу розробки. Розробка додатків - це безперервний процес, який триватиме навіть після початкового випуску в міру збору даних користувачів і додавання нових функцій.

.

РОЗДІЛ 2
ІНСТРУМЕНТАРІЙ

2.1. Figma

Figma - це веб-сервіс для дизайну та прототипування, який дозволяє командам працювати над дизайном інтерфейсу користувача (UI) разом в режимі реального часу.

Порівняно з іншими додатками для дизайну, Figma вирізняється своєю онлайн-колаборацією, легкістю використання та можливістю працювати на різних платформах без необхідності установки додатків. Figma часто використовується для створення макетів веб-сайтів, мобільних додатків, а також для роботи над загальними дизайн-проектами у реальному часі, що робить його популярним інструментом серед дизайнерів та розробників.

2.2. Середовище розробки Xcode

Xcode - це інтегроване середовище розробки (IDE) від Apple, призначене для створення програм для платформ iOS, iPadOS, macOS, watchOS і tvOS.

Це основний інструмент для розробки програмного забезпечення для продуктів Apple

Ця програма дозволяє створювати додатки, які подалі можуть бути розміщені у магазині додатків Apple. Її можуть використовувати як початківці, так і досвідчені розробники.

Цей інструментарій містить набір засобів, які сприяють швидкому просуванню в процесі розробки, дозволяючи досвідченим розробникам швидко створювати додатки, а початківцям подолати менше труднощів і перешкод при створенні високоякісного додатка.

Xcode містить всі необхідні інструменти для розробки програми в єдиному програмному пакеті, такі як текстовий редактор, компілятор і система збірки. За допомогою Xcode можна писати, компілювати і налагоджувати додаток, а по завершенні процесу розробки його можна відправити в Apple App Store.

Xcode призначений для того, щоб забезпечити розробникові єдине вікно для роботи. У ньому реалізовані функції перевірки вихідного коду і автозаповнення, що значно полегшує процес написання коду. При створенні нового проекту розробник може обрати один з доступних шаблонів, які надають базову основу для подальшого розширення.

Xcode, як редактор коду, підтримує широкий спектр мов програмування, включаючи C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit і Swift. Він використовує моделі програмування Cocoa, Carbon і Java.

Ці функції корисні для новачків, бо надають опору, на яку можна спиратися під час навчання. Досвідчені розробники використовують ці можливості для оптимізації свого робочого процесу та значного прискорення розробки додатків.

Xcode є обов'язковим інструментом для розробки додатків на платформі Apple, і це єдина офіційно підтримувана компанією Apple опція для створення програмного забезпечення для їхніх пристроїв. Хоча існують сторонні рішення, які не вимагають використання Xcode, вони не підтримуються Apple, і це може призвести до проблем при подальшій реалізації коду та оновленнях.

Xcode в світі програмування визначається високоякісними інструментами для налагодження, які дозволяють розробникам ефективно вирішувати проблеми у своєму додатку. Крім того, він допомагає при управлінні ресурсами, такими як зображення і файли коду, завдяки вбудованим інструментам(див. рис. 2.2.)

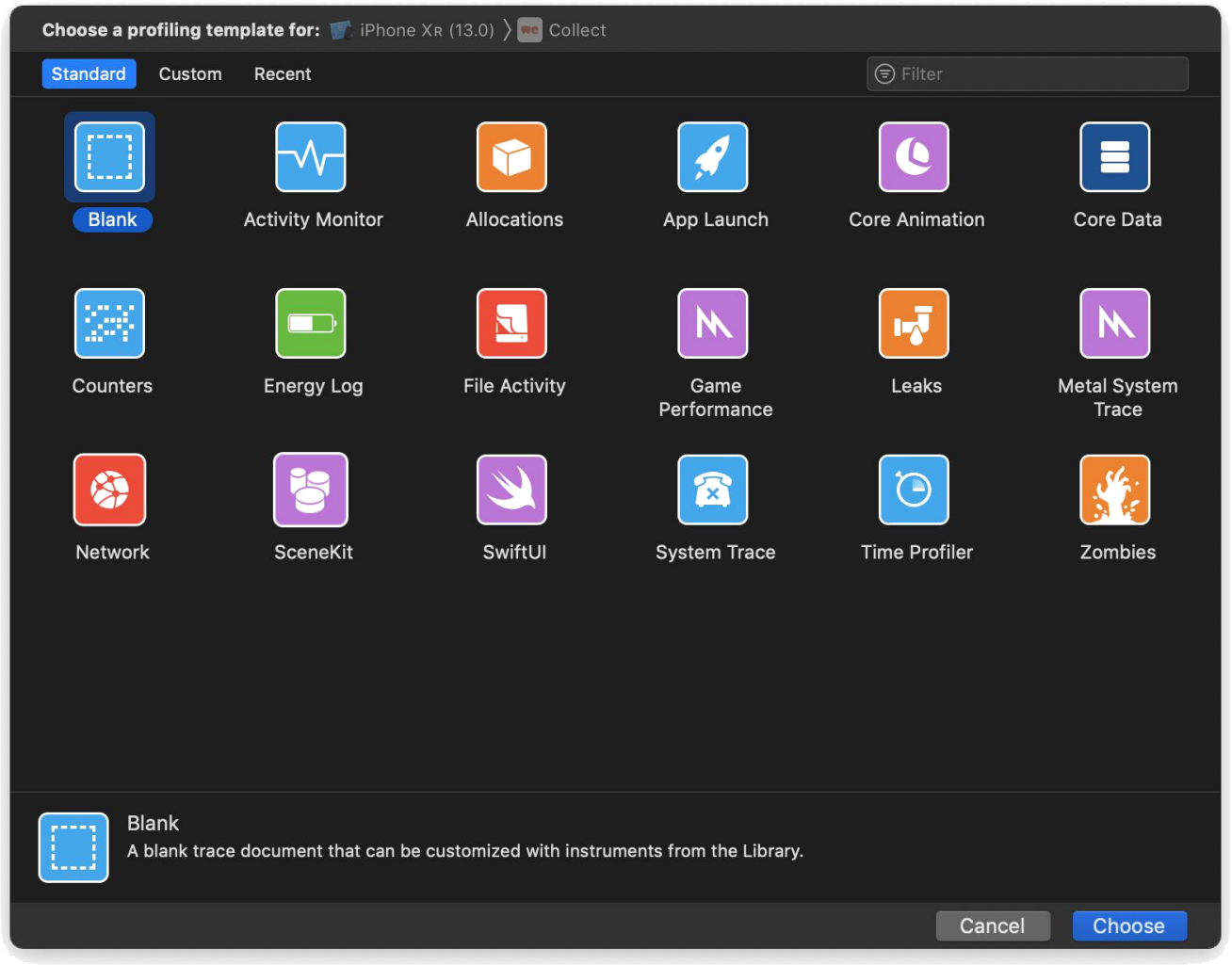

Рис. 2.2. Панель інструментів

Хоча існують сторонні інтегровані середовища розробки (IDE), які дозволяють створювати додатки для iOS за межами macOS, вони часто не можуть забезпечити повноцінне тестування та налагодження. Важливим аспектом розробки ефективного функціонального додатку для iOS є налагодження та тестування, і в цьому Xcode виявляється вельми корисним.

2.3. Swift

Swift - це мова програмування, розроблена компанією Apple для створення додатків під iOS, macOS, watchOS та tvOS. Вона була представлена вперше в 2014 році на конференції WWDC (Worldwide Developers Conference) і швидко здобула популярність серед розробників.

Рис. 2.3. Логотип Swift

Swift був розроблений (рис. 2.3.), щоб бути більш сучасною, безпечною та швидкою альтернативою до Objective-C. Незважаючи на те, що Swift взяв певні концепції з Objective-C, він є повністю незалежною мовою програмування. Однак важливо відзначити, що Swift побудований на основі технічних досягнень Objective-C і C.

Основні риси Swift включають:

- Безпека типів: Swift використовує строгу систему типів, що дозволяє виявляти та усувати помилки програмування на ранніх етапах розробки.
- Інтерактивність та Playgrounds: Інтерактивна оболонка Swift (Playgrounds) дозволяє розробникам експериментувати, тестувати і вивчати код без необхідності компіляції великих програм.
- Сучасний синтаксис: Swift має простий і чистий синтаксис, що полегшує розуміння та написання коду.

- Усунення багатьох проблем Objective-C: Swift вирішує деякі недоліки Objective-C, такі як використання заголовків і подвійного підрахунку коду.
- Високий рівень безпеки: Захист від багатьох типових помилок, таких як вказівники на нуль, завдяки опціональним типам.

Swift і Objective-C можуть співіснувати в одному проекті, що дозволяє розробникам поступово переходити від одного до іншого, не переписуючи весь код одразу. Це робить Swift привабливим вибором для розробників, які хочуть оновити свої додатки, використовуючи нові можливості та переваги мови Swift.

2.4. Поняття фреймворк

Фреймворк (framework) - це комплексна структура або складний набір інструментів та бібліотек, які надають основу для розробки програмного продукту. Фреймворки роблять розробку програм більш простою і ефективною, надаючи структурований підхід до проектування та реалізації програмних рішень.

2.5. Базові Фреймворки

UIKit є важливою складовою для розробки iOS-додатків і надає зручний інтерфейс для взаємодії зі стандартними компонентами операційної системи. Розробники використовують UIKit для створення інтерфейсів, які дотримуються стандартів дизайну Apple та забезпечують зручність взаємодії для користувачів.

Foundation - це один із основних фреймворків, які постачаються з платформою Apple, включаючи iOS, macOS, watchOS та tvOS. Цей фреймворк містить основні класи та інструменти, необхідні для створення додатків для цих платформ.

Обидва фреймворки мають спільну основну мету - запезпечння обміну даними та коду між різними бібліотеками та фреймворками. Крім цього, Core Foundation також вбудовує підтримку інтернаціоналізації. Ця підтримка реалізується

за допомогою непрозорого типу CFString, який ефективно керує масивом символів Unicode.

MapKit - це фреймворк, розроблений Apple для інтеграції карт та функцій мапу в додатки, створені для операційних систем iOS та macOS. Він надає розробникам інтерфейс та інструменти для відображення та взаємодії з картами, включаючи можливості маркування місць, отримання вказівок для маршрутів та інші картографічні функції. MapKit(рис. 2.4) спрощує інтеграцію карт у додатки, що дозволяє створювати додатки з розширеними географічними можливостями.

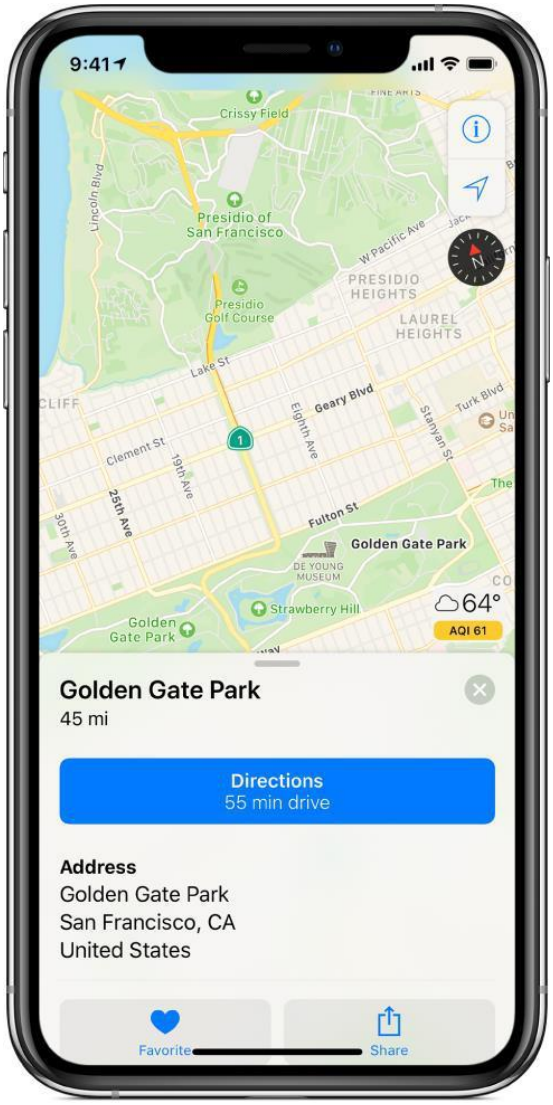

Рис. 2.4. Вигляд MapKit

При створенні об'єкта MKMapView можна задати початковий регіон для відображення, визначивши властивість "регіон карти". Цей регіон визначається центральною точкою та відстанню по горизонталі та вертикалі, що називається проміжком.

Один із прикладів використання фреймворку MapKit у додатках для iOS та macOS — це клас MKMapView, який використовується для відображення та управління інформацією на карті. Цей клас дозволяє центрувати карту за вказаними координатами, вказувати розмір відображуваної області та забезпечувати користувальницькою інформацією.

Клас MKMapView також дозволяє додавати анотації на карту для відображення специфічної інформації (рис. 2.5)

Оскільки карта може містити велику кількість анотацій, для їх відображення у режимі перегляду використовуються різні типи об'єктів анотацій. Анотації служать для позначення конкретних міток на карті і можуть містити різні дані для відображення. Кожен об'єкт анотації містить інформацію про розташування на карті, а також описову інформацію, яку можна відобразити.

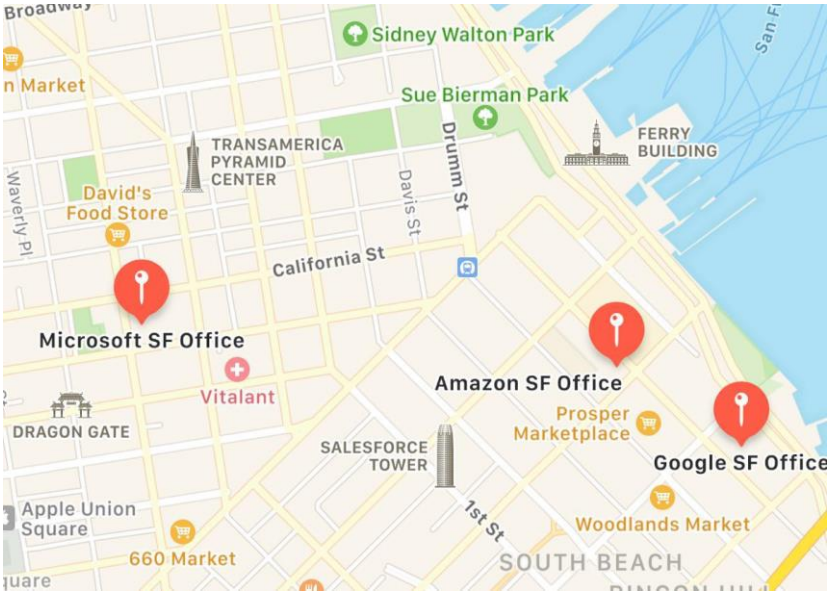

Рис. 2.5. Аннотації

У додатку карти будуть використані для вибору місця проведення заходів та для можливості стежити за тим як швидко її учасники добираються до місця зустрічі.

Push-повідомлення (push-сповіщення) - це повідомлення, яке відправляється з сервера до пристрою або додатку користувача без прямого запиту користувача. Ці повідомлення використовуються для того, щоб звертати увагу користувачів на нові події, оновлення або інші важливі інформаційні повідомлення. (рис. 2.6.).

Рис. 2.6. Push-повідомлення

Push-повідомлення можна використовувати для:

- Відображення короткого текстового повідомлення (попередження):

Push-повідомлення може включати короткий текст, який відображається на екрані пристрою. Це дозволяє привертати увагу користувача до нових подій чи оновлень у додатку.

- Відтворення звуку повідомлення:

Крім тексту, push-повідомлення може включати звуковий елемент, який відтворюється при отриманні повідомлення. Це може бути корисно для нагадування або вказівки на важливі події.

- Встановлення номеру на значку програми (Badge):

Значок програми може відображати номер, що вказує на кількість нових або непрочитаних елементів у додатку. Це сприяє зручності користувача відразу ж бачити, що сталося нового.

- Вказання дій, які користувач може зробити:

Push-повідомлення може містити додаткові дії, які користувач може виконати без відкривання додатку. Це може включати кнопки або швидкі дії прямо з повідомлення.

- Показ вкладень мультимедіа:

Push-повідомлення може містити вкладення, такі як зображення, відео чи інші мультимедійні елементи, які користувач може переглядати безпосередньо з повідомлення.

- Виконання завдань у фоновому режимі:

Додатки можуть використовувати push-повідомлення для виконання певних завдань або оновлення даних, навіть коли додаток не активний.

- Групування повідомлень в потоки:

Повідомлення можуть групуватися за певними категоріями або потоками для зручності організації та сприяння легшому взаємодії з користувачем.

- Редагування або видалення доставлених повідомлень:

Деякі системи дозволяють користувачам редагувати або видаляти вже доставлені push-повідомлення, що може бути корисно в ряді випадків.

Ці можливості дозволяють створювати ефективні та взаємодійливі сповіщення для користувачів.

2.6. Системи Контролю версій

Системи контролю версій (СКВ) - це програмні засоби для відстеження змін у файлах та координації роботи над спільними проектами. Вони дозволяють вам фіксувати стани файлів та відновлювати їх до певних моментів часу, а також спільно працювати з іншими розробниками, об'єднуючи їхні зміни.

Кожен розробник працює у власній гілці, що представляє собою ізольовану версію коду. Це означає, що внесені ними зміни не впливають на основну гілку розробки. Після завершення та готовності змін у гілці, їх можна об'єднати (злити) з основною гілкою. Цей підхід сприяє організації вихідного коду та підвищує ефективність розробки, роблячи процес більш гнучким.

Системи контролю версій (СКВ) надають численні переваги, які роблять їх важливим інструментом для розробників та команд розробки програмного забезпечення. Декілька основних переваг включають:

- Історія змін:

СКВ зберігає повну історію змін файлів та коду. Кожен коміт фіксує конкретний стан проекту на певний момент, що дозволяє вам відновлювати, переглядати та розуміти, як і чому відбулися зміни.

- Гілки:

Можливість створювати гілки дозволяє розробникам незалежно працювати над різними функціями чи виправленнями помилок. Гілки дозволяють робити експерименти, не впливаючи на основний код.

- Об'єднання та відгалуження:

Вміння об'єднувати (merge) та створювати відгалуження (branch) спрощує розробку. Це дозволяє здійснювати розподілену роботу над різними функціями, а потім злити зміни в один спільний код.

- Відновлення та відкат змін:

Завдяки історії змін можна легко відновити проект до попереднього стану або відкотити неправильні зміни. Це важливо для виправлення помилок чи відновлення працездатності проекту.

- Співробітництво:

СКВ сприяють співробітництву в реальному часі. Кілька розробників може працювати над одним і тим же проектом, і їхні зміни можуть бути легко об'єднані.

- Відстеження авторства:

СКВ зберігають інформацію про авторів кожного коміту, дозволяючи відстежувати, хто і коли вніс зміни до коду.

- Робота з віддаленими репозиторіями:

Багато СКВ підтримують віддалені репозиторії, що дозволяє розробникам спільно працювати над проектами, навіть якщо вони розташовані в різних місцях.

- Зручність виявлення конфліктів:

Конфлікти в коді можуть бути виявлені та вирішені шляхом розгляду змін між гілками або версіями файлів.

- Безпека та резервне копіювання:

Зберігання історії змін дозволяє вам забезпечити резервне копіювання коду та відновлення до попередніх станів проекту у випадку втрати даних.

У якості системи контролю версій було обрано Sourcetree. Sourcetree - це графічний клієнт для роботи з системами контролю версій Git та Mercurial. Він надає інтуїтивний інтерфейс для взаємодії з репозиторіями, управління гілками, перегляду та відстеження змін у коді, а також для здійснення комітів та об'єднання гілок.

Sourcetree розроблений компанією Atlassian і є популярним інструментом серед розробників для спрощення роботи з системами контролю версій на різних операційних системах, таких як Windows та macOS.(рис. 2.19.).

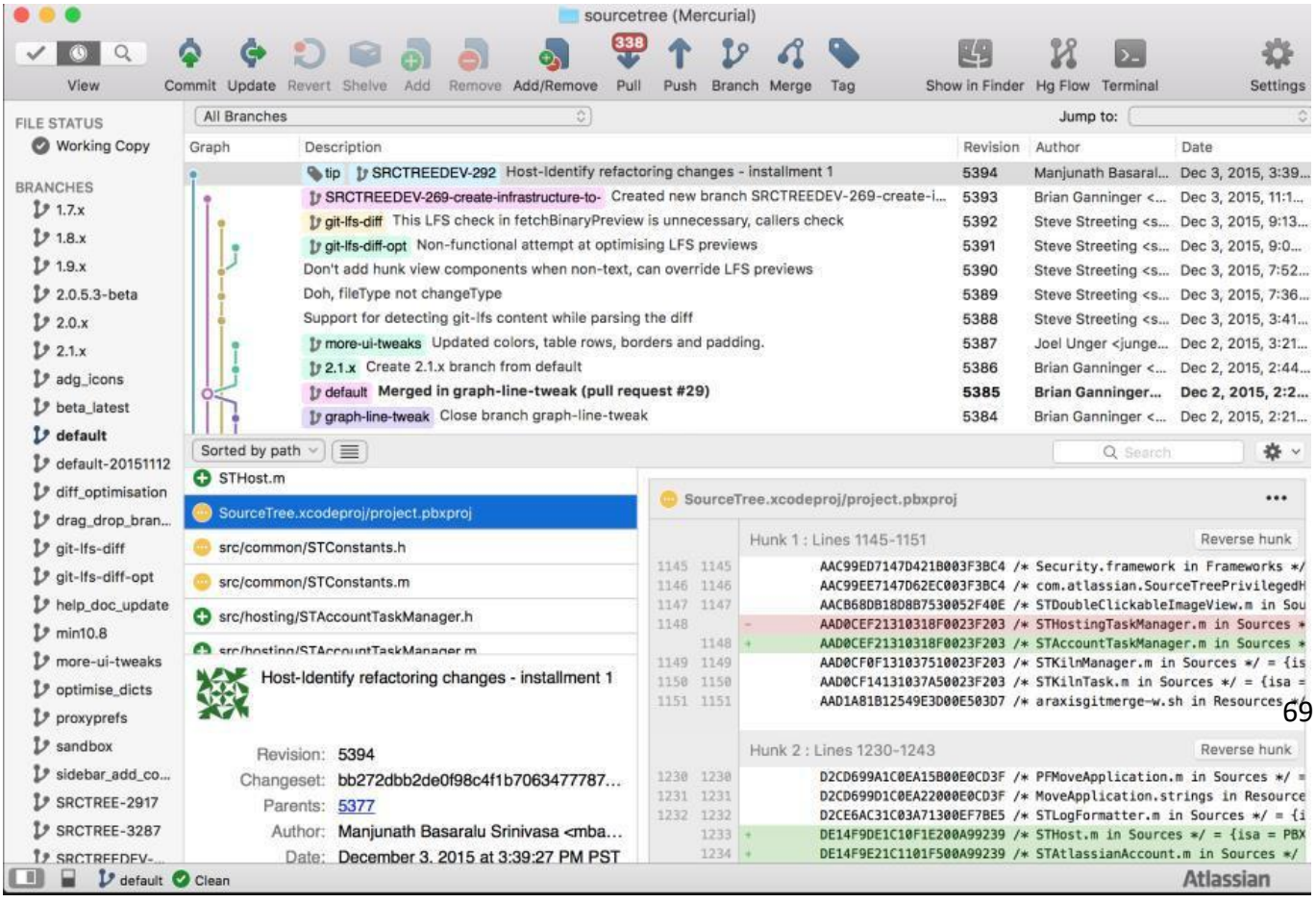

Рис 2.19. Вигляд Sourcetree

Висновок до розділу 2

Було розглянуто середовище розробки Xcode та корисні інструменти, які використовуються під час розробки. Також була приділена увага сучасній мові програмування Swift, яка застосовується для створення додатків для iOS/MacOS систем.

Основні інструменти, які надає середовище розробки Xcode, а також додаткові фреймворки, додані до проекту за допомогою Cocoapods, були детально проаналізовані. Серед найпопулярніших фреймворків, використовуваних розробниками, виділяються Moya та Kingfisher. Використання цих фреймворків значно спрощує та прискорює процес розробки мобільного додатка.

РОЗДІЛ 3.
ДЕМОНСТРАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Структура застосунку

На рис. 4.1. представлено Navigation Area проекту, в якому всі файли складають xcodeproj. Файли цього типу використовуються для збереження інформації про проект Xcode та результатів певного етапу розробки.

AppDelegate - це клас, що відповідає за обробку певних станів додатку, наприклад, коли додаток відправляє дані на сервер або коли користувач переходить у режим офлайн.

В Resources розміщені зображення та шаблони кольорів. SwiftGen бібліотека дозволяє зручно використовувати ці ресурси у коді.

У Components розміщені користувацькі елементи, які відображаються однаково у всьому додатку, наприклад, поле введення.

У Services розміщені файли з різними сервісами. Сервіс - це клас, що відповідає за певну ділянку функціональності програми. Наприклад, NetworkService відповідає за взаємодію з REST-запитами, а DataSource - за роботу з базою даних.

Extension - це окрема папка з розширенням функціоналу певних класів. Наприклад, розширення функціоналу класу UILabel може включати підкреслення тексту.

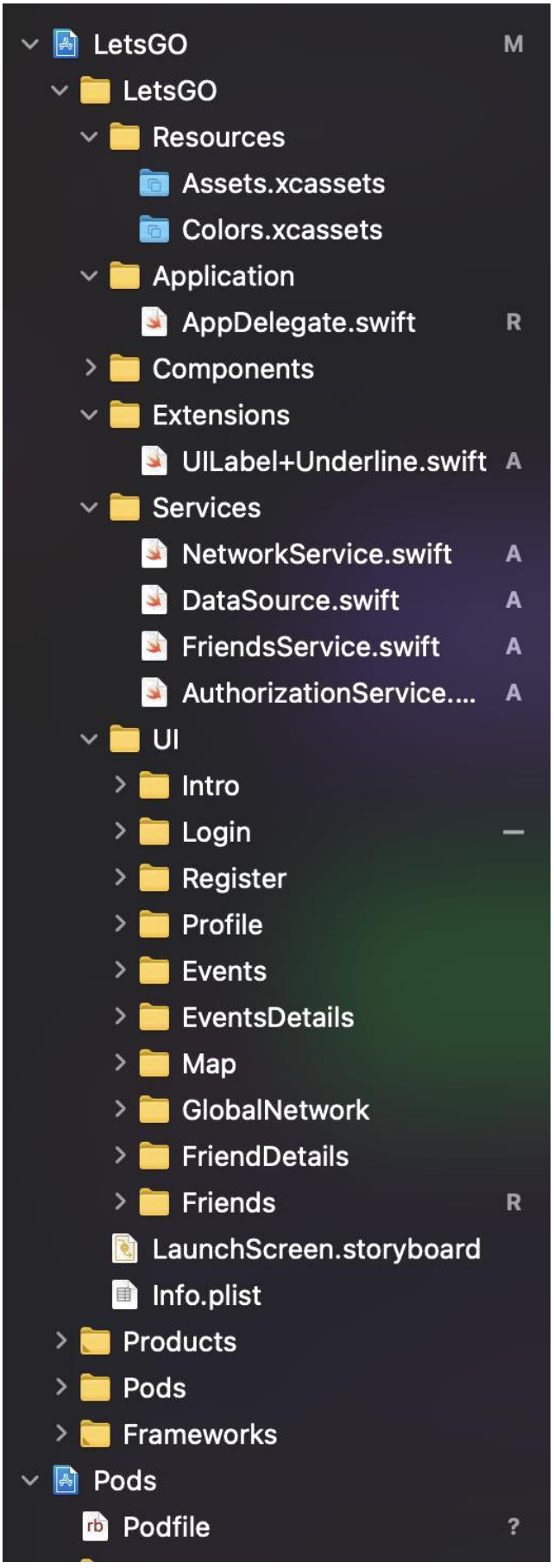

Рис 4.1. Navigation Area

Також у Figma є збережений власний шаблон зі стилю (рис. 4.2.), що має візуальні елементи додатка.

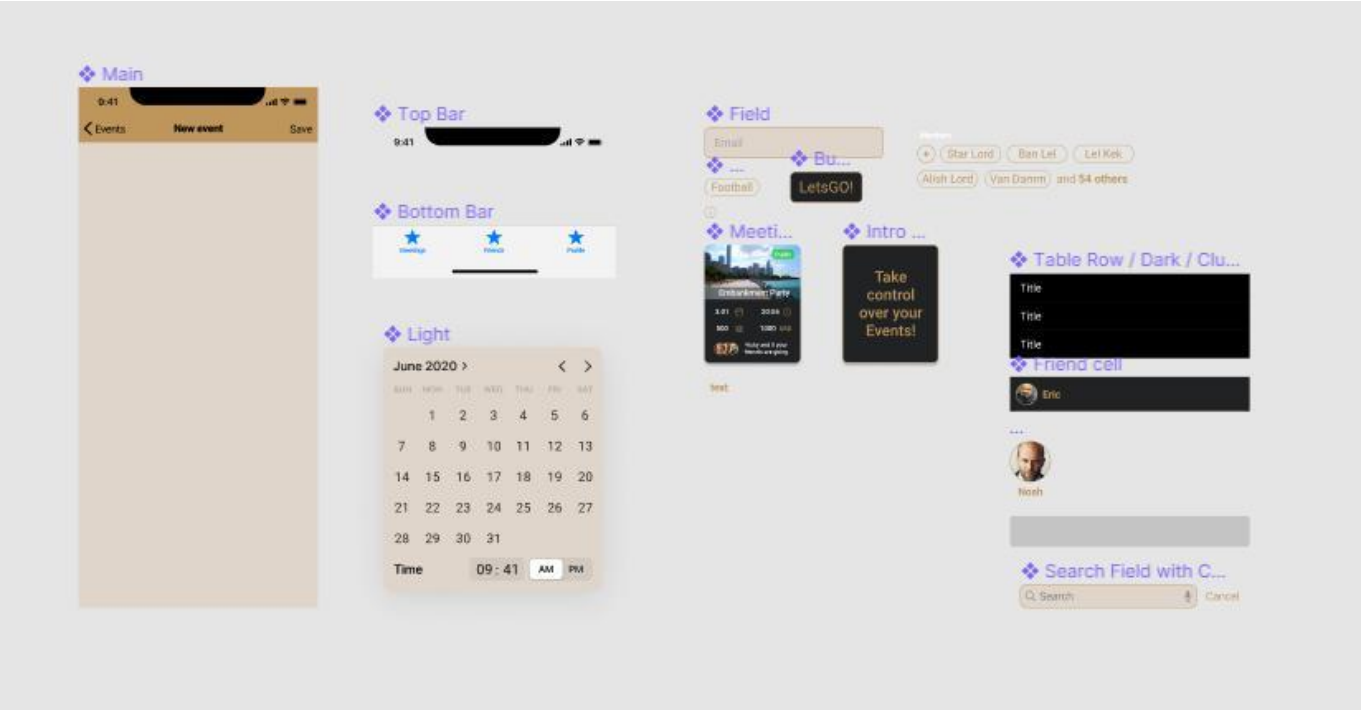

Рис 4.2. Посібник зі стилю

3.2. Огляд функціоналу застосунку

Кожен екран у додатку використовує свій власний View Controller. Навігацію між екранами забезпечує Navigation Controller, який є контейнером для одного або кількох дочірніх View Controller у вигляді інтерфейсу навігації.

Tab Bar Controller показує вкладки у нижній частині вікна для вибору різних режимів та відображення відповідних екранів для кожного режиму.

При відкритті додатка виконується перевірка булевого значення "isFirstEnter" в User Defaults. User Defaults - це plist (property list) файл у кореневій папці додатка, який використовується для зберігання простих даних.

У такому інтерфейсі може бути відображений лише один дочірній View Controller одночасно. Вибір елемента в контролері виду призводить до виведення нового контролера виду на екран з анімацією, тим самим приховуючи попередній контролер виду.

Структура plist файлу подібна до словника (колекції ключ-значення). Якщо значення "isFirstEnter" є false, то відображається екран вступу (рис. 4.3.).

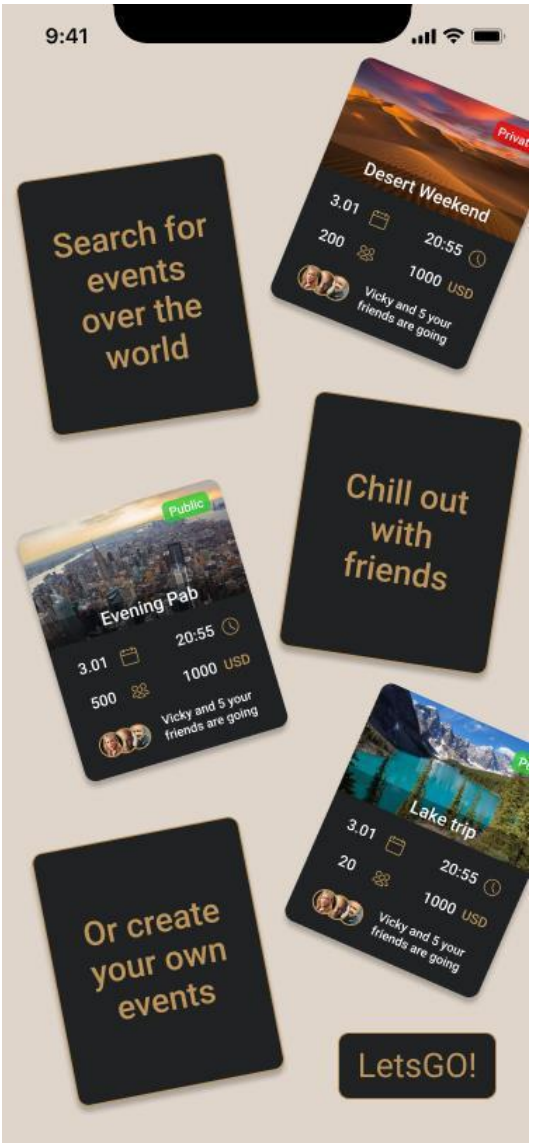

Рис 4.3. Екран вступу

Усі картки на екрані вступу реалізовані за допомогою комбінації наступних об’єктів з фреймворка UIKit:

- UIImageView - об'єкт, що відображає одне зображення або послідовність анімованих зображень.
- UILabel - представлення, що відображає одну або кілька рядків інформаційного тексту.
- UIView - об'єкт, який керує вмістом прямокутної області на екрані.

Далі користувач може натискати кнопку "LetsGO", розташовану в правому нижньому куті. Ця кнопка реалізована за допомогою UIButton, елементу управління, який виконує визначений код у відповідь на дії користувача. Після натискання Navigation Controller здійснює перехід на наступний екран – Login (рис. 4.4.).

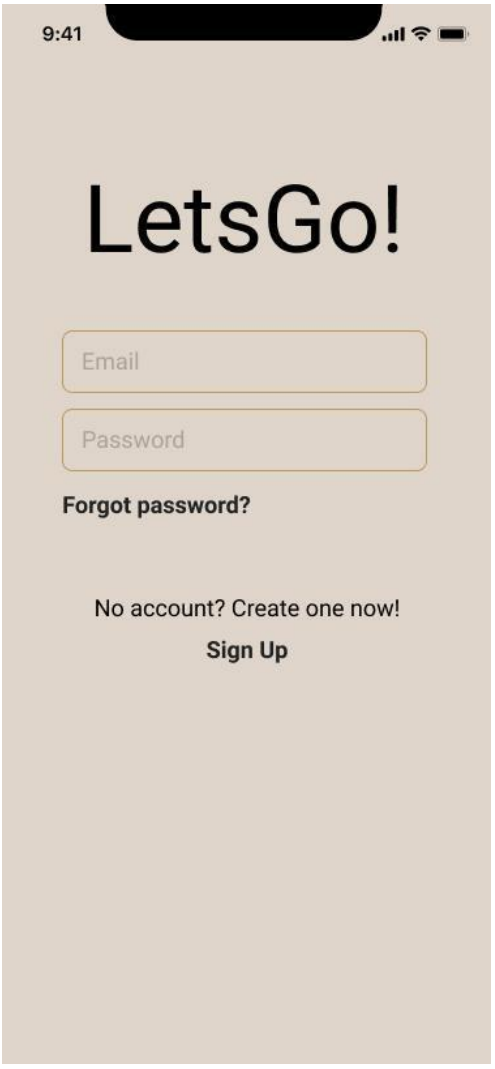

Рис 4.4. Логін

Будь-які зміни в кольорах або функціоналі можна легко внести, змінивши значення в масиві налаштувань. Крім того, у разі приєднання нового розробника до проекту, цей підхід спрощує розуміння і модифікацію коду.

Поля для введення на цьому екрані представлені елементами, створеними зі стандартних компонентів UIKit для подальшого використання. Таким чином, всі поля для введення в додатку будуть мати однаковий вигляд і не будуть потребувати великої кількості коду для створення та використання.

Функціонал додатку включає в себе випадок, коли користувач забуває свій пароль. Натискання кнопки "Forgot password" спричинить надсилання користувачеві детальної інструкції з відновлення паролю на електронну пошту.

На екрані для введення даних використовуються елементи UIView та UILabel для створення полів для вводу. Однаковий вигляд досягається завдяки спільним налаштуванням вигляду.

У випадку, якщо користувач не має облікового запису, він може зареєструватися, натискавши кнопку "SignUp". Navigation Controller перейде до екрану реєстрації (рис. 4.5.).

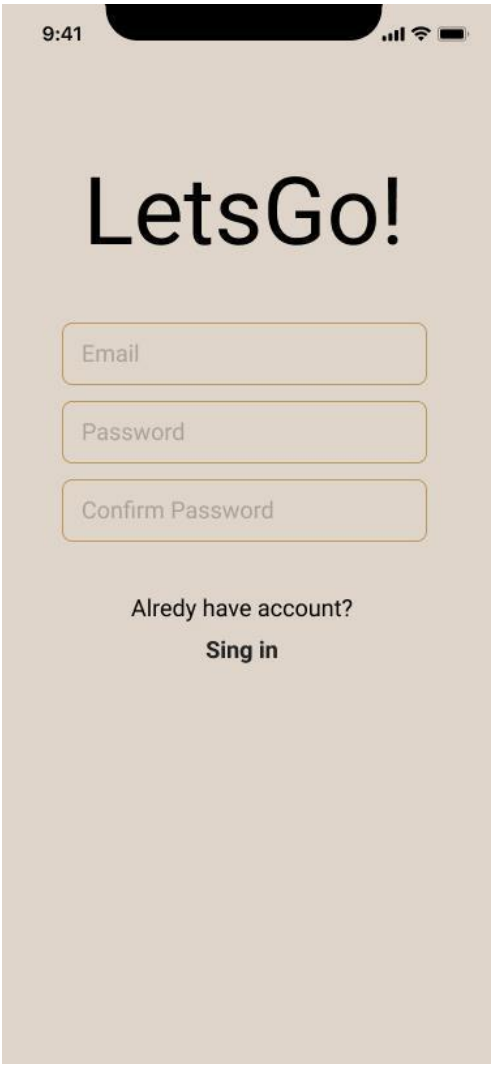

Рис. 4.5. Екран реєстрації

Після введення даних Navigation Controller знову відобразить екран входу з попередньо заповненою електронною адресою. Користувачу потрібно підтвердити свою електронну адресу, після чого він зможе увійти у свій обліковий запис.

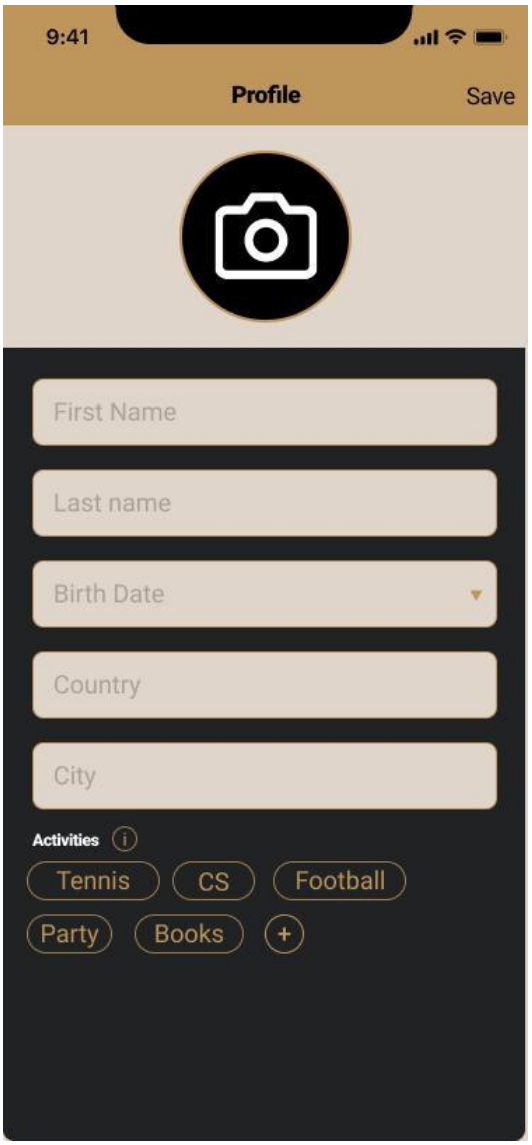

Рис. 4.6. Екран заповнення профілю

Щоб вибрати фотографію профілю, користувач повинен натиснути кнопку з іконкою фотокамери, що призведе до відкриття UIImagePickerController (рис. 4.7.). Це є контролером, який управляє системними інтерфейсами для фотографування, запису відео та вибору елементів із медіа-бібліотеки користувача.

Далі користувачу потрібно заповнити свій профіль (рис. 4.6.). Кожне поле буде проаналізовано та використано для підбору найкращої події.

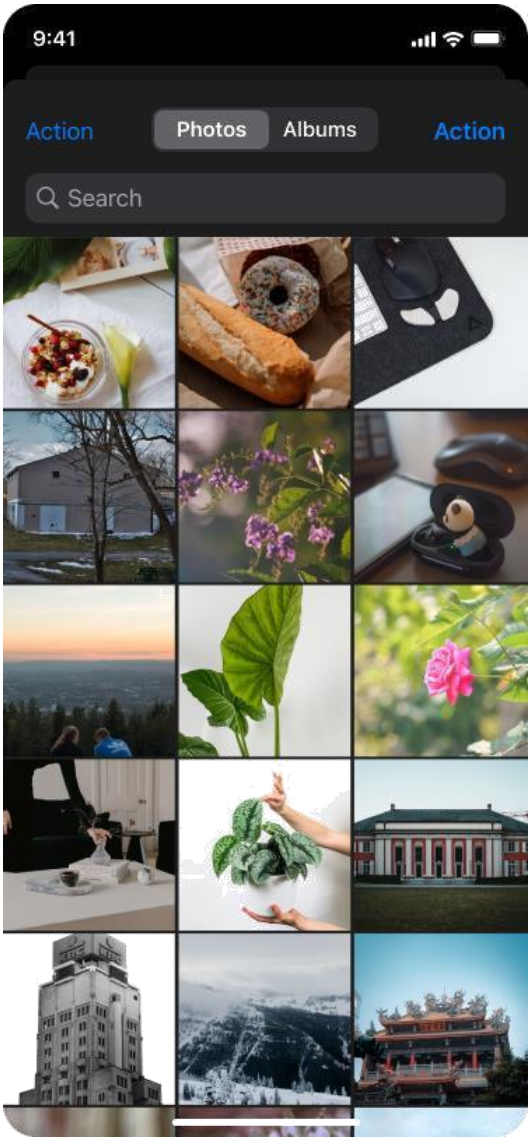

Рис. 4.7. Вигляд UIImagePickerController

При натисканні на кнопку «Info» з'явиться сповіщення (рис. 4.8.), в якому буде описано, для чого призначена ця функція.

На екрані заповнення профілю є функція додавання так званих «Activities». Завдяки їм буде значно легше підібрати подію, яка буде відповідати запитам користувача.

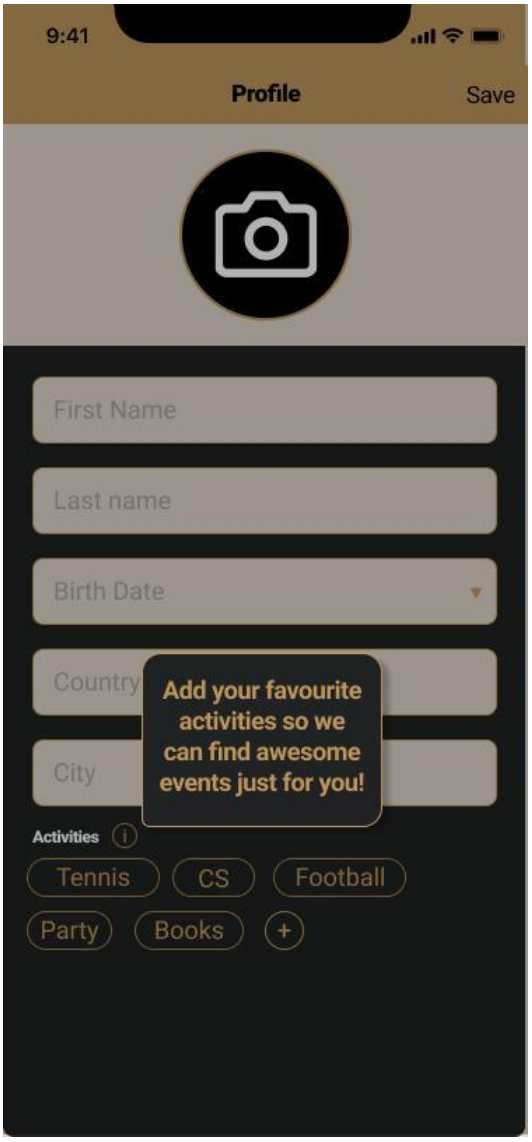

Рис 4.8. Приклад сповіщення

Кожний вид активності відображений за допомогою UIButton з певними налаштуваннями. Натиснувши кнопку «+», відкриється клавіатура з допоміжною панеллю (рис. 4.9.). На ній при вводі користувачем символів будуть показані варіанти діяльності що будуть максимально схожі на введені.

Панель з відповідями реалізована за допомогою UICollectionView - об’єкту, який керує впорядкованою колекцією елементів даних і представляє їх за допомогою настроюваних макетів. Відслідкування розміру та положення клавітури допоможе правильно розташувати панель.

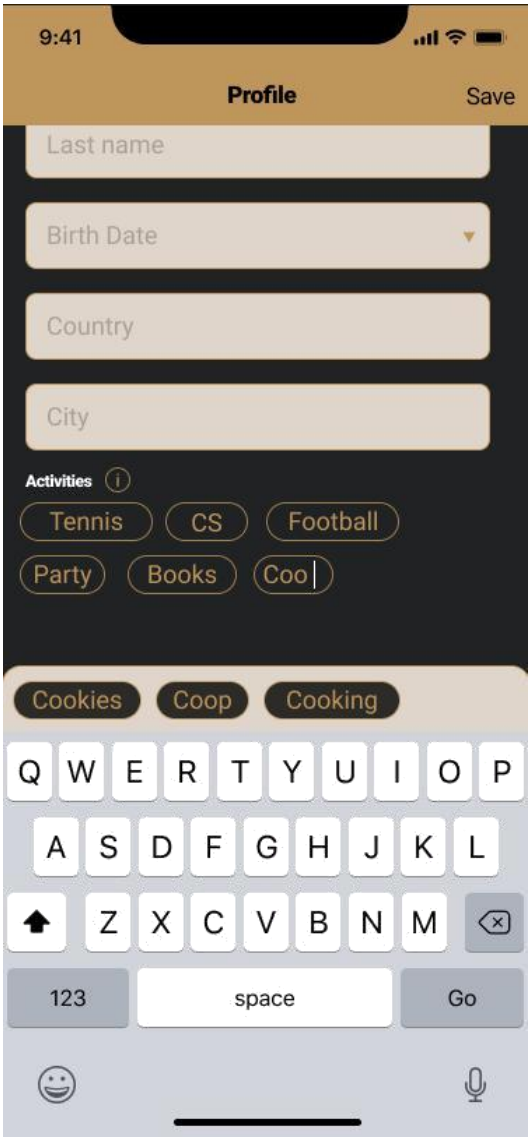

Рис. 4.9. Додавання видів діяльності

Після цього користувач опиниться на екрані із зустрічами (рис. 4.10.). Зустрічі можна відсортувати (за датою, активностями, місцем, тощо), натиснувши відповідну кнопку на навігаційній панелі справа.

Завершується заповнення профілю натисканням на кнопку «Save» у правому верхньому кутку.

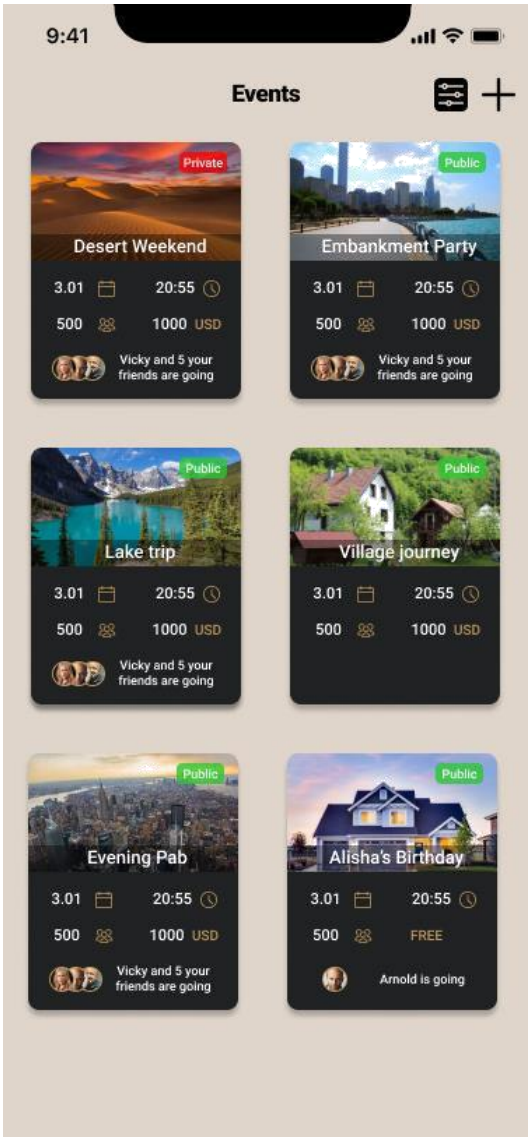

Рис. 4.10. Екран із зустрічами

Кнопка «+» на навігаційній панелі переведе користувача на сторінку створення зустрічі (рис. 4.11).

Список заходів буде персонально підібраний для кожного користувача. Будуть відображені як глобальні зустрічі (будь то фан-зустріч з улюбленим виноавцем), так і локальні (святкування дня нарождження, пікнік).

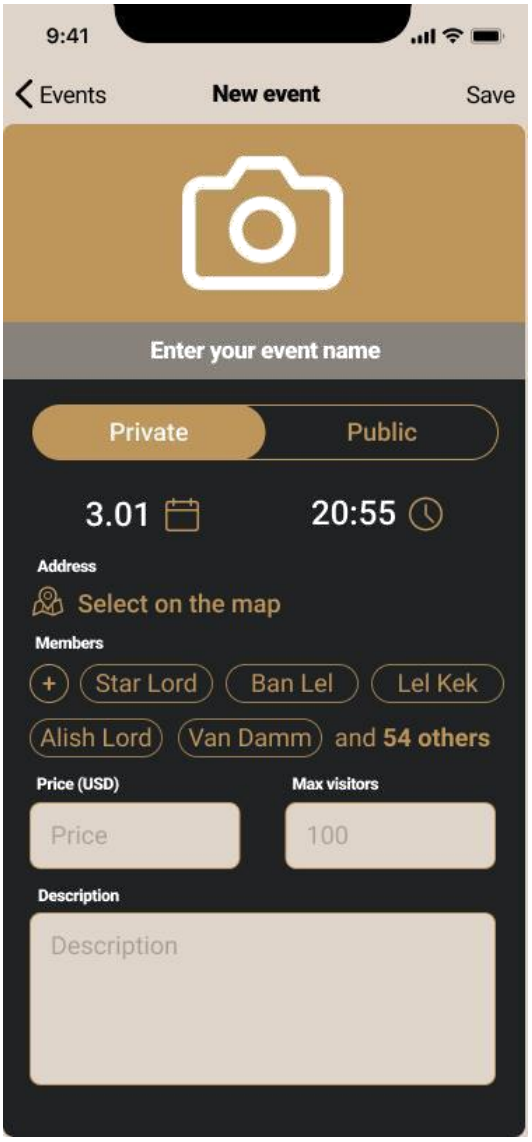

Рис. 4.11. Створення зустрічі

Дата та час зустрічі вибираються за допомогою DatePicker (рис. 4.12.) – елементу керування для вибору абсолютної дати.

Зустрічі можуть бути як приватні, так і публічні. До публічних можна приєднатися у будь-який момент, якщо ще не набралася максимальна кількість учасників.

Фотографія зустрічі вибирається за тим же принципом, що й вибір фотографії профілю.

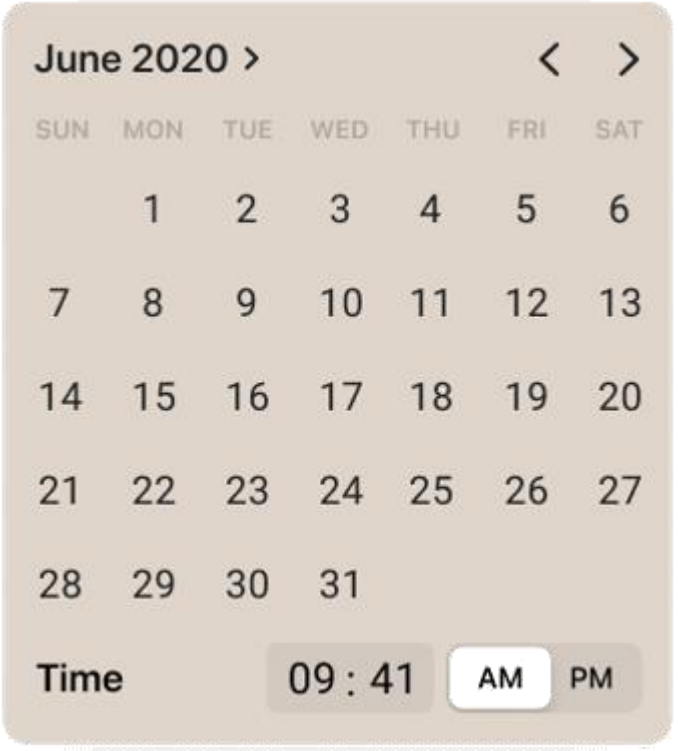

Рис 4.12. DatePicker

Щоб вибрати місце проведення зустрічі, потрібно натиснути на кнопку «Select on the map», після чого буде показаний екран вибору місця на карті (рис. 4.13.).

В описі можна залишити будь-яку додаткову інформацію: дрес-код, сторінка у соціальній мережі, правила поведінки, тощо.

Є можливість одразу запросити друзів за принципом схожим до Activities. Вказується ціна відвідування, або залишається пустою, якщо захід є безкоштовним. Кількість запрошених також вказується лише за необхідності.

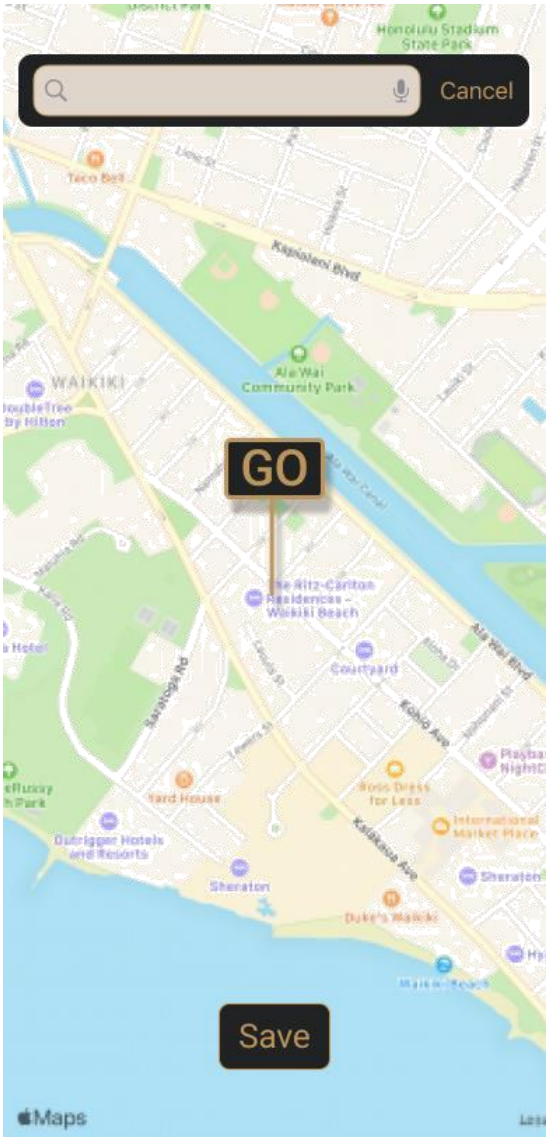

Рис. 4.13. Вибір місця на карті

Наступним етапом буде відкриття екрану друзів (рис. 4.14.). З друзями у майбутньому можна буде створювати та ділитися зустрічами.

Таблиця із надписом «GO» є анотацією, координати якої будуть переведені у реальну адресу та збережені на сервері. Також є можливість знайти місце за допомогою пошукової панелі зверху. Кнопка «Save» поверне користувача назад до створення зустрічі.

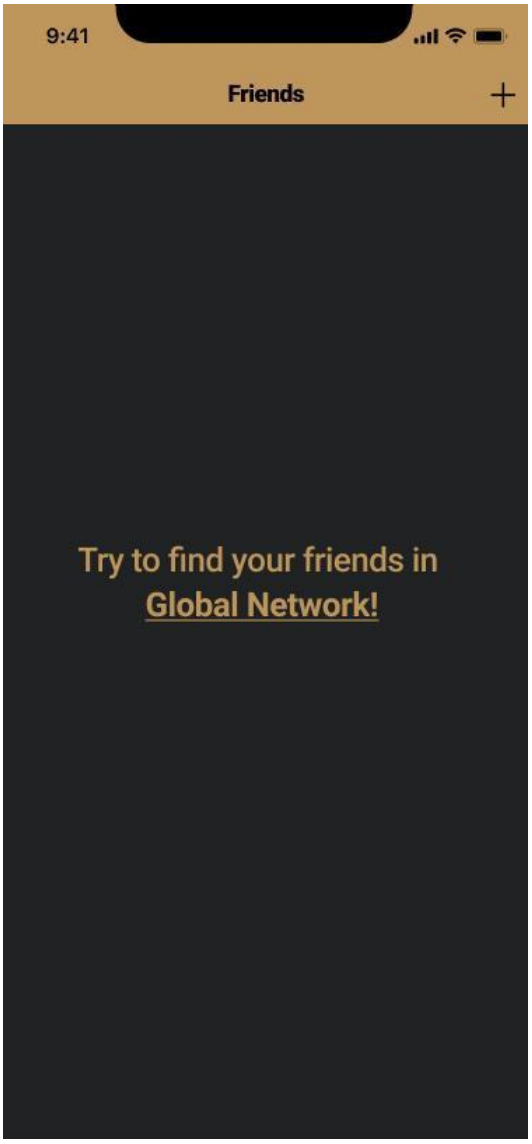

Рис 4.14. Екран друзів

При натисканні на цей підкреслений напис відкриється екран глобальної мережі з усіма користувачами (рис. 4.15.).

Якщо у користувача немає жодних друзів у межах додатку, відображатиметься напис "Try to find your friends in the Global Network!".

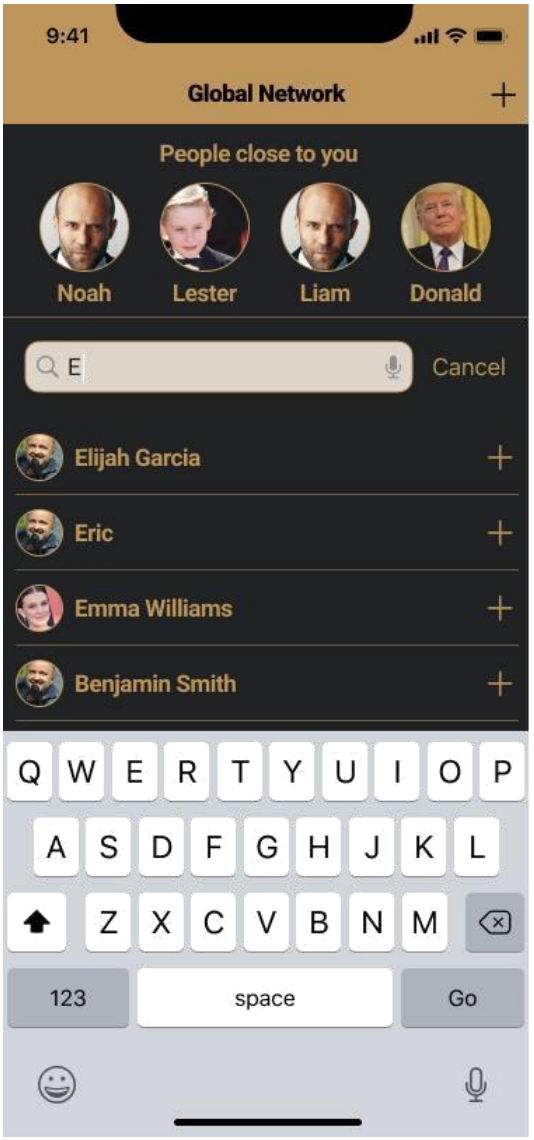

Рис 4.15. Екран глобальної мережі

Це типовий приклад використання UITableView. Однією з ключових функцій є можливість додавати у друзі людей, які знаходяться поблизу. Панель буде прихована, якщо користувач відмовить у доступі до свого місцезнаходження.

На цьому екрані ви зможете знайти як своїх друзів, так і інших користувачів з усього світу. Основою цього екрану є UITableView - представлення, яке відображає дані у вигляді рядків у одному стовпці.

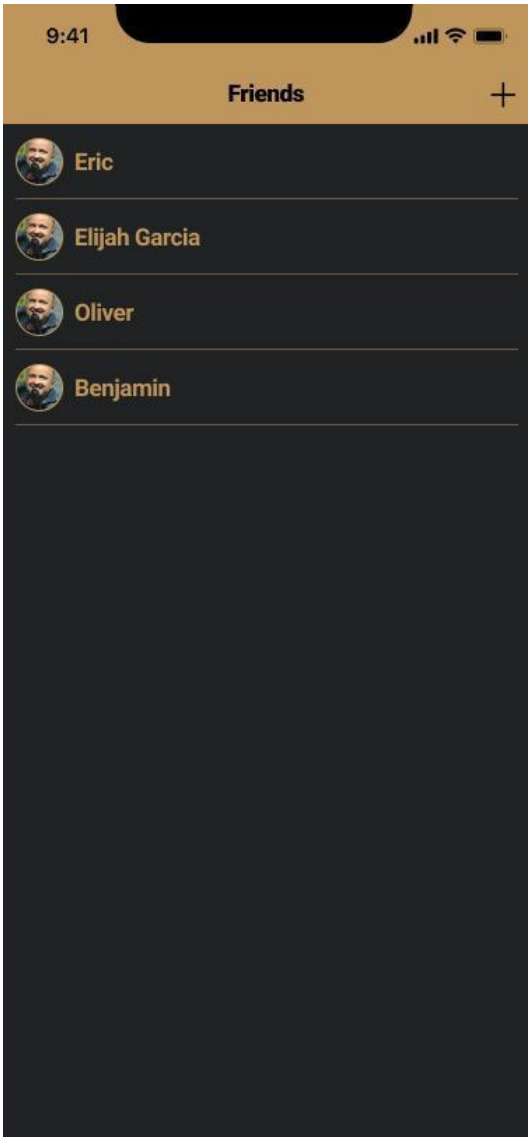

Рис. 4.16. Заповнений екран з друзями

Для кожної зустрічі буде створена окрема кімната чату (рис. 4.17.), реалізована за допомогою фреймворку з набором для чату Chatto та використанням бази даних Realm. Chatto має готові набори повідомлень, що значно полегшує створення чату. Безумовно, додаток має оптимізовану архітектуру, спроектовану для забезпечення ефективної роботи чату, завдяки чому він працює плавно та безперебійно.

Заповнений екран з друзями (рис. 4.16.) також використовує UITableView як основу.

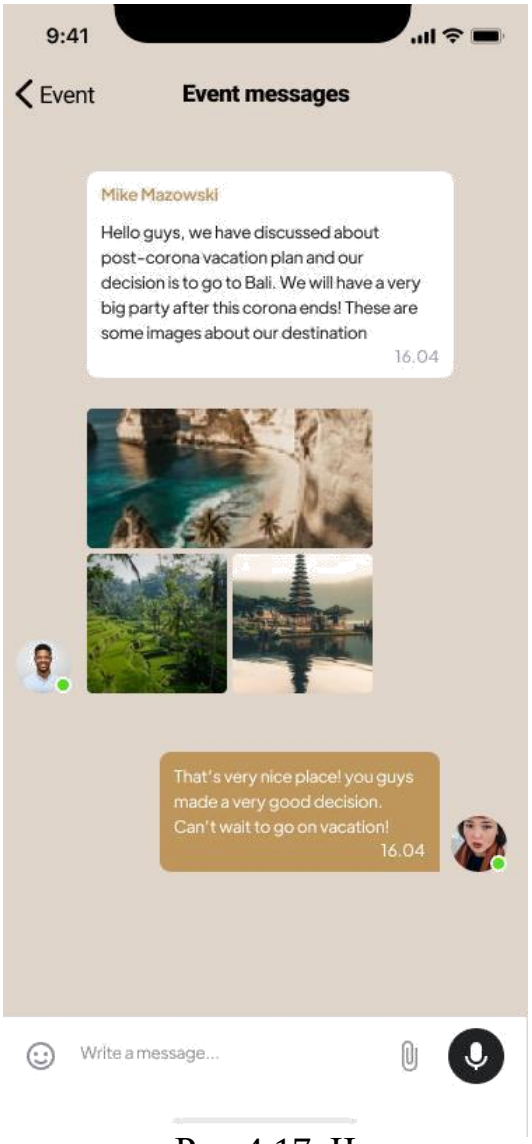

Рис 4.17. Чат

Використання БД Realm забезпечує швидкий запис повідомлень на диск для подальшого відтворення.

Елемент керування UITabBar (рис. 4.18.) дозволяє переходити між різними основними екранами, представленнями чи режимами програми.

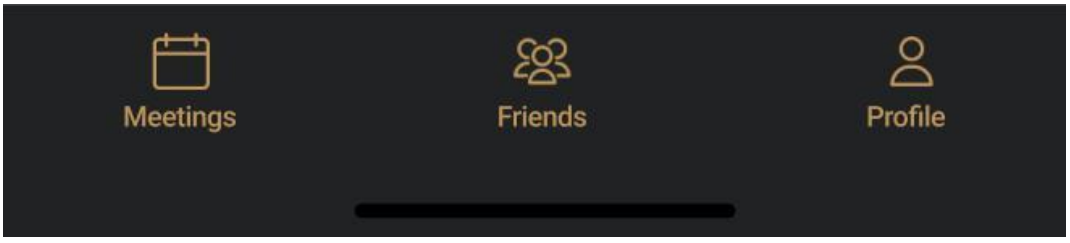

Рис 4.18. Tab Bar

Висновки до розділу 3

Розглянули структуру та основний функціонал застосунку, та його роботу.

ВИСНОВКИ

Досліджено базові та додаткові фреймворки, які використовувалися у розробці додатка, полегшуючи реалізацію функціоналу та прискорюючи процес.

Під час виконання дипломної роботи був розроблений додаток із необхідним функціоналом. Розглянуті різні типи мобільних додатків, визначено вимоги проекту та розглянуті етапи розробки.

Використано інструмент Figma для створення макету та реалізації його Swift. Тестування додатка проведено на різних етапах, включаючи симулятор та реальний пристрій, для підтвердження відсутності багів та здатності забезпечувати плавну відповідь на взаємодію з користувачем.

Також проведено аналіз архітектури, зокрема стандартної архітектури Apple MVC, виявлені недоліки, і було знайдено альтернативу у вигляді MVP.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Mixpanel. [Електронний ресурс] режим доступу: https://mixpanel.com (дата звернення 26.10.2021 р). – Назва з екрана.
2. Indeed. [Електронний ресурс] режим доступу: https://www.indeed.com (дата звернення 10.11.2021 р). – Назва з екрана.
3. Toptal. [Електронний ресурс] режим доступу: https://www.toptal.com (дата звернення 15.11.2021 р). – Назва з екрана.
4. Perfecto. [Електронний ресурс] режим доступу: https://www.perfecto.io (дата звернення 17.11.2021 р). – Назва з екрана.
5. Ray Wenderlich. [Електронний ресурс] режим доступу: https://www.raywenderlich.com (дата звернення 24.11.2021 р). – Назва з екрана.
6. Web Design. [Електронний ресурс] режим доступу: https://webdesign.tutsplus.com (дата звернення 24.11.2021 р). – Назва з екрана.
7. Software Testing Help. [Електронний ресурс] режим доступу: https://www.softwaretestinghelp.com (дата звернення 25.11.2021 р). – Назва з екрана.
8. Data Science. [Електронний ресурс] режим доступу: https://datascience.eu/ru (дата звернення 25.11.2021 р). – Назва з екрана.
9. Course Report. [Електронний ресурс] режим доступу: https://www.coursereport.com (дата звернення 17.11.2021 р). – Назва з екрана.
10. PCMag [Електронний ресурс] режим доступу: https://www.pcmag.com (дата звернення 22.10.2021 р). – Назва з екрана.
11. Coder Lessons. [Електронний ресурс] режим доступу: https://coderlessons.com (дата звернення 25.11.2021 р). – Назва з екрана.