МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2021 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТРА"

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

Тема: " Мобільний додаток для організації зустрічей "

Виконавець: Заякін Денис Костянтинович

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

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

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

"____"_____2021 р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Заякіна Дениса Костянтиновича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Мобільний додаток для організації зустрічей» затверджена наказом ректора від 12.10.2021 за № 2228/ст.

2. Термін виконання роботи: з 12.10.2021 по 31.12.2021.

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 додатку, розгляд використаних інстурментів 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	12.10.2021 20.10.2021	
2.	Створення плану дипломного проекту	21.10.2021 01.11.2021	
3.	Консультація з науковим керівником	02.11.2021 14.11.2021	
4.	Розробка розділу «Огляд етапів процесу розробки мобільного додатку»	15.11.2021 25.11.2021	
5.	Розробка розділу «Інструментарій»	26.11.2021 01.12.2021	
6.	Розробка розділу «Демонстрація мобільного додатку»	02.12.2021 10.12.2021	
7.	Оформлення дипломного проекту	11.12.2021 31.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи _____ Юрій МОДЕНОВ
(підпис керівника)

Завдання прийняв до виконання _____ Денис ЗАЯКІН
(підпис випускниці)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Мобільний додаток для організації зустрічей ” складається із вступу, чотирьох розділів, загальних висновків, списку використаних джерел і містить 91 сторінку тексту та 52 рисунка. Список використаних джерел містить 11 найменувань

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

Предметом дослідження є розробка мобільного додатку для організації зустрічей, 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.

Об’єктом дослідження є організація зустрічей за допомогою додатку.

Ключові слова: МОБІЛЬНИЙ ДОДАТОК, SWIFT, АРХІТЕКТУРА ІНТЕРФЕЙС, ФРЕЙМОВОРК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UX – *User Experience* – Досвід користувача.

UI – *User Interface* – Інтерфейс користувача.

VC – *View Contoller* – Контролер виду.

TV – *Table View* – Вид таблиці.

NC – *Navigation Controller* – Контролер навігації.

ВСТУП

Сучасний темп життя змушує нас планувати кожну свою секунду життя. Коли щось стосується організації не лише свого життя а й планування майбутніх зустрічей, конференцій це доставляє нам певших труднощів, адже до уваги потрібно брати не лише власний час але й зіставити із планами інших людей.

Актуальним варінтом вирішенням даної проблеми буде використання додатку для планування, який би містив би у собі весь потрібний функціонал: вибір місця, дати, часу та кількості усаників заходу. Застосунок який би мав до уваги не лише думку однієї особи але усієї групи. Звичайно, ці всі нюанси створюють певні труднощі для подальшою реалізації такої програми, адже це питання охоплює не лишу технічну частину додатку, але й теоретичну частину яка поєднується із соціальними факторами.

Для вирішення задачі створення додатку потрібен покроковий план, який би включав у себе вирішення як технічних так і теоретичних проблем: від виникнення ідеї до реалізації додатку.

Створення додатку починається з виникнення ідеї та мотивації. Потім це виражається у типі додатку та вимогах. Робота із серверною частиною та дизайном інтерфейсу програми плавно переходить у прототип додатку та його тестування. Заключним етапом є публікація додатку.

Тож проведемо детальний аналіз створення мобільного додатку на базі iOS, а саме: типу додатку, етапів розробки, прототипування, фреймворків та результату виконання практичной частини.

РОЗДІЛ 1.
ОГЛЯД ЕТАПІВ ПРОЦЕСУ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

За даними Statista, річний дохід додатків досяг 111 мільярдів доларів. Це показує величезне зростання кількості програм, доступних у Play Store та App Store.

Задовго до цього, поняття про які ми думали в науково-фантастичних фільмах або книгах, стали звичними в багатьох сферах нашого повсякденного життя. Будь то AR, VR, машинне навчання чи Інтернет речей; вони приносять революцію в кожній галузі.

Як усі ці технології вплинули на розвиток розробки мобільних додатків або їх процес? Розробка додатків – це не легка справа. Щоб отримати очікувану вигоду, розробка вимагає вдумливої та покрокової процедури. Тож проведемо аналіз типів та життєвого циклу розробки додатків.

1.1. Визначення мобільного додатку

Мобільний додаток є різновидом програмного забезпечення, призначеного для запуску на мобільному телефоні, наприклад, смартфоні або планшетному ПК. Додатки - це загалом невеликі окремі програмні блоки з обмеженою ємністю. Таке використання прикладного програмування спочатку пропагувалося компанією Apple, а також її магазином додатків, який пропонує величезну кількість застосувань.

На противагу додаткам, призначеним для настільних комп'ютерів, мобільні додатки відходять від вбудованих програмних систем. За всіх рівних умов кожний універсальний застосунок надає роздільну та обмежену корисність. Наприклад, це, як правило, гра, обчислювач чисел або портативний інтернет-браузер.

Початковий мобільний додаток надавав інформацію загального призначення та інформаційні послуги в глобальній мережі, що вміщав в собі електронну пошту, календар, фондовий ринок, списки та інформацію про погоду. Однак попит користувачів мобільних пристроїв, поряд з можливістю розробки мобільного додатку, поширюється і на інші категорії, такі як мобільні ігри, автоматизація виробництва, GPS та інші. Вибухове збільшення числа і різноманітності застосувань призвело до появи великих і різноманітних областей. Багато послуг в даний час потребують допомоги технологій мобільних додатків, таких як визначення місця розташування, інтернет-банкінгу, відстеження, покупки квитків і навіть мобільних медичних послуг.

Найпростіші мобільні програми використовують програми на базі ПК і портують їх на мобільний пристрій. У міру розвитку мобільних додатків ця стратегія стає досить недосконалою. Більш сучасна методологія включає в себе явне зростання для мобільного середовища, використання його обмежень і переваг. Наприклад, додатки, в яких основні функціональні області, як правило, працюють з самого раннього етапу з прицілом на мобільні пристрої, враховуючи, що клієнт не прив'язаний до будь-якої області, як при використанні ПК.

1.2. Типи мобільних додатків

1.2.1. Нативні програми

Нативний мобільний додаток - це той тип програми, в якому він створюється та розробляється для певного типу платформ пристроїв, таких як Android або IOS (рис. 1.1.), із використанням спеціалізованої мови кодування. Щоб створити нативну програму, мова кодування, яку вибирають розробники, повинна мати доступ до платформи пристрою. Типовими функціями програми для цієї категорії можуть бути офлайнові мобільні ігри, додатки-словники тощо.

Рис. 1.1. Нативний додаток

Головною перевагою нативних додатків є їх користувальницький інтерфейс високої якості. Враховуючи всі обставини, дизайнери, які створюють їх, використовують власні пристрої інтерфейсу користувача. Доступ до широкого спектру API-інтерфейсів також допомагає прискорити розробку і розширює можливості використання додатків. Нативні програми повинні бути завантажені з магазинів додатків і безпосередньо впроваджені в пристрої. Саме з цієї причини їм спочатку необхідно пройти суворий процес розподілу.

Зазвичай в грі завантажуються всі зображення, звуки і рівні, щоб користувач міг грати в гру без підключення до Інтернету (деякі ігри вимагають підключення до Інтернету, тому що їм потрібно увійти в систему, купувати або продавати предмети всередині або тому що вони є онлайн-іграми). Ще один поширений приклад нативних мобільних додатків, які всі знають, - Facebook.

1.2.2. Веб-додатки

Веб-додатки - це програмні додатки, які працюють відповідно з власними мобільними додатками і працюють на мобільних пристроях. Однак існують суттєві відмінності між нативними та веб-додатками. Веб-додатки використовують для запуску браузери, і зазвичай вони написані на CSS, HTML або JavaScript. Такі

програми перенаправляють клієнта на URL - адресу, а потім пропонують йому вибрати програму. В результаті веб-додатки змушують клієнтів створювати закладки на такій сторінці для подальшого перегляду. Саме з цієї причини вони вимагають найменшого обсягу пам'яті.

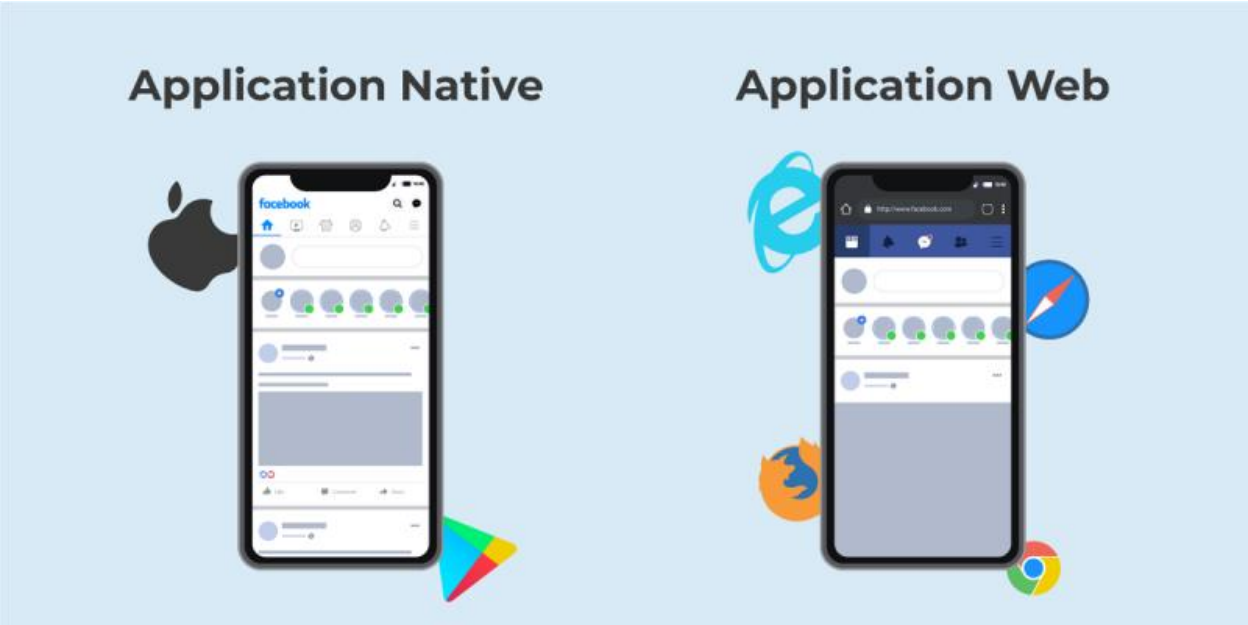

Рис. 1.2. Нативний та веб додатки

Веб-додатки виконують той же метод організації в порівнянні з нативними додатками, але доступ до них здійснюється за допомогою браузера веб-сайту на мобільному пристрої (рис. 1.2.). Вони не є незалежними додатками з точки зору завантаження та встановлення коду на пристрій. Це дійсно адаптивні веб-сайти, які налаштовують свій інтерфейс користувача під пристрій клієнта. При виборі "встановити" веб-додаток, він часто створює закладки для URL-адреси сайту на пристрої.

1.2.3. Гібридні програми

Гібридний додаток поєднує переваги мобільного веб-додатка та нативного додатка. Він побудований за допомогою HTML, CSS, Javascript, працює на мобільному WebView. Однак додаток Hybrid все ще може скористатися перевагами таких функцій пристрою, як захоплення, GPS, вібрація тощо.

Це веб-додатки, які дуже нагадують нативні програми. Вони можуть мати символ програми на головному екрані, адаптивний дизайн, швидку продуктивність, навіть мати можливість функціонувати відключено, однак вони дійсно є веб-додатками, створеними для того, щоб виглядати нативно.

Гібридні додатки будуть написані на основі кросплатформного фреймворку: Cordova, Ionic тощо. Функції мобільного пристрою будуть викликані через API, наданий цією платформою, у формі Javascript. Таке потрібно написати тільки один раз, ці фреймворки автоматично переведуть цю програму в інсталяційні файли для Android та iOS. Деякі програми, які не надто складні в обробці і потребують використання функціональних можливостей пристрою, розроблюються саме за такою методикою.

1.2.4. Різниця між нативним і гібридним мобільним додатком

Нативні та гібридні програми - це два типи мобільних додатків. Незважаючи на їх схожість за зовнішнім виглядом і дизайном, що лежать в їх основі, технології є абсолютно різними. Як випливає з назви, гібридні програми об'єднують онлайн-додатки з власними мобільними додатками. HTML, CSS та JavaScript - це всі веб-технології, які можна використовувати для їх створення. Поширюють їх через магазини додатків, де споживачі можуть завантажувати їх як нативні програми для Android або iOS.

Основна відмінність між гібридними та нативними програмами полягає в тому, що гібридні програми створюються на всіх платформах, тоді як нативні програми розробляються для певних операційних систем (рис. 1.3.). На відміну від унікального додатка для кожної операційної системи смартфона, гібридний додаток є єдиним для всіх платформ і однаково добре функціонує на кожній з них.

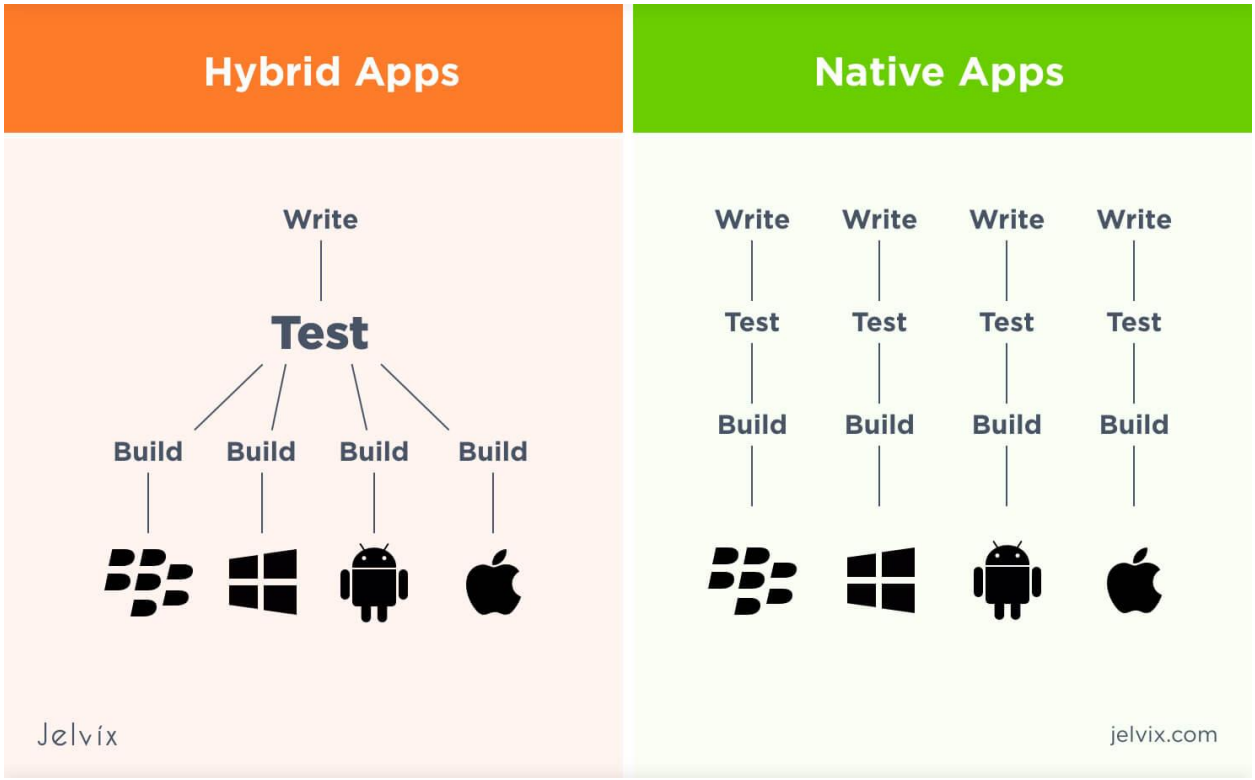

Рис. 1.3. Відмінність гібридного та нативного додатків

Переваги гібридних програм включають мобільність і простоту створення потрібно написати код лише один раз, і гібридне програмне забезпечення працюватиме в багатьох операційних системах. Для розробки кроссплатформенних додатків можна обрати гібридні платформи, такі як Ionic або Cordova. Якщо говорити про нативні мобільні додатки, то вни мають бути створені на мовах, що відповідають платформі, як-от Java, Swift або Objective-C.

Крім того, нативні мобільні додатки можуть отримувати доступ до вбудованих функцій смартфона, таких як камера і мікрофон. Гібридний додаток покладається на плагіни, такі як плагіни Cordova, щоб скористатися перевагами вбудованих можливостей пристрою.

Проаналізуємо ключові переваги нативного додатка:

- **Безпека**. На відміну від гібридних програм, які покладаються лише на безпеку системного браузера, кросплатформні та нативні програми високо захищені від неправомірного використання завдяки різним рівням операційної системи. Крім того, замість того, щоб залежати від сторонньої

системи, кожна власна програма має свої офіційні API, які повністю перевіряються в різних версіях системи. Більш того, завдяки тривалим циклам запуску розробники можуть розраховувати на надійне, ретельно перевірене і більш безпечне програмне забезпечення.

- **Підтримка**. У порівнянні з обслуговуванням гібридного додатка, обслуговування нативного додатка трохи складніше, тому що повинна бути запущена остання версія програми. Тим не менш, користувачі нативних додатків можуть отримувати інформацію про останні оновлення без будь-яких проблем. Завдяки можливості оновлювати максимальну кількість контенту при установці, нативні програми не вимагають постійних оновлень, як гібридні додатки.
- **Продуктивність**. Оскільки нативні програми написані мовами, характерними для екосистеми платформи, вони можуть працювати швидше, забезпечуючи тим самим кращий досвід роботи з користувачем. Замість того, щоб залежати від нативних браузерів, таких як гібридні програми, нативні програми можуть отримати доступ до унікальних елементів і API, оптимізованих для різних пристроїв, що робить їх роботу ефективною та безперебійною.
- **Менше багів**. Використання спільної кодової бази для кількох платформ спочатку економить час, але в довгостроковій перспективі підтримувати її важче, ніж використовувати окрему кодову базу для певної платформи. Незалежні від міжплатформних інструментів, таких як Cordova або Xamarin, нативні програми, швидше за все, відчувають менше помилок. Крім того, розробники можуть отримати доступ до нового SDK, щоб почати розробляти свої власні програми з найновішими функціями, надаючи користувачам програми швидкий доступ до нових функцій платформи.
- **Уніфікований UI / UX**. За допомогою нативних додатків користувачі можуть скоротити і спростити процес звикання до додатку, оскільки весь користувальницький інтерфейс буде стандартним. Що стосується

дизайнерів і розробників, то з уніфікованим користувальницьким інтерфейсом їм також простіше застосовувати кращі стандарти і практики в порівнянні з гібридними додатками.

- **Доступ до повних функцій пристрою**. Оскільки кожна нативна програма розроблена для певної платформи, користувач програми може отримати прямий доступ до апаратного забезпечення пристрою, як-от мікрофон, камера, GPS тощо. Крім того, рідні програми мають велику перевагу в push-повідомленнях, які надходять через APNS (сервер iOS), для яких потрібно ідентифікатор пакету додатків, і аналогічно GCM (хмарний обмін повідомленнями Google).
- **Розширюваність**. Завдяки високій сумісності з однією платформою власні програми можна швидко налаштовувати, що полегшує їх масштабування. Однак, якщо потрібно об'єднати переваги як нативних, так і кросплатформених рішень, то знадобиться розпочати нативно, а потім максимізувати кілька менших модулів програми за допомогою додаткового кросплатформного коду. Це безпечний метод, який застосовують кілька технологічних гігантів на ринку, включаючи Airbnb і Facebook.
- **Автономна продуктивність**. Нативні додатки можуть ефективно працювати без підключення до мережі Інтернет, це тому, що весь вміст програми завантажується прямо під час установки і роботи. Крім того, ці програми включають надійні функції кешування в браузері, що дозволяють зробити всі кешовані ресурси доступними в автономному режимі.

1.3. Визначення розробки мобільних додатків

З технічної точки зору розробка мобільного додатку-це процес або процедура створення програмного забезпечення, що працює на смартфонах. Пізніше додаток завантажується і встановлюється користувачем з магазинів додатків.

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.).

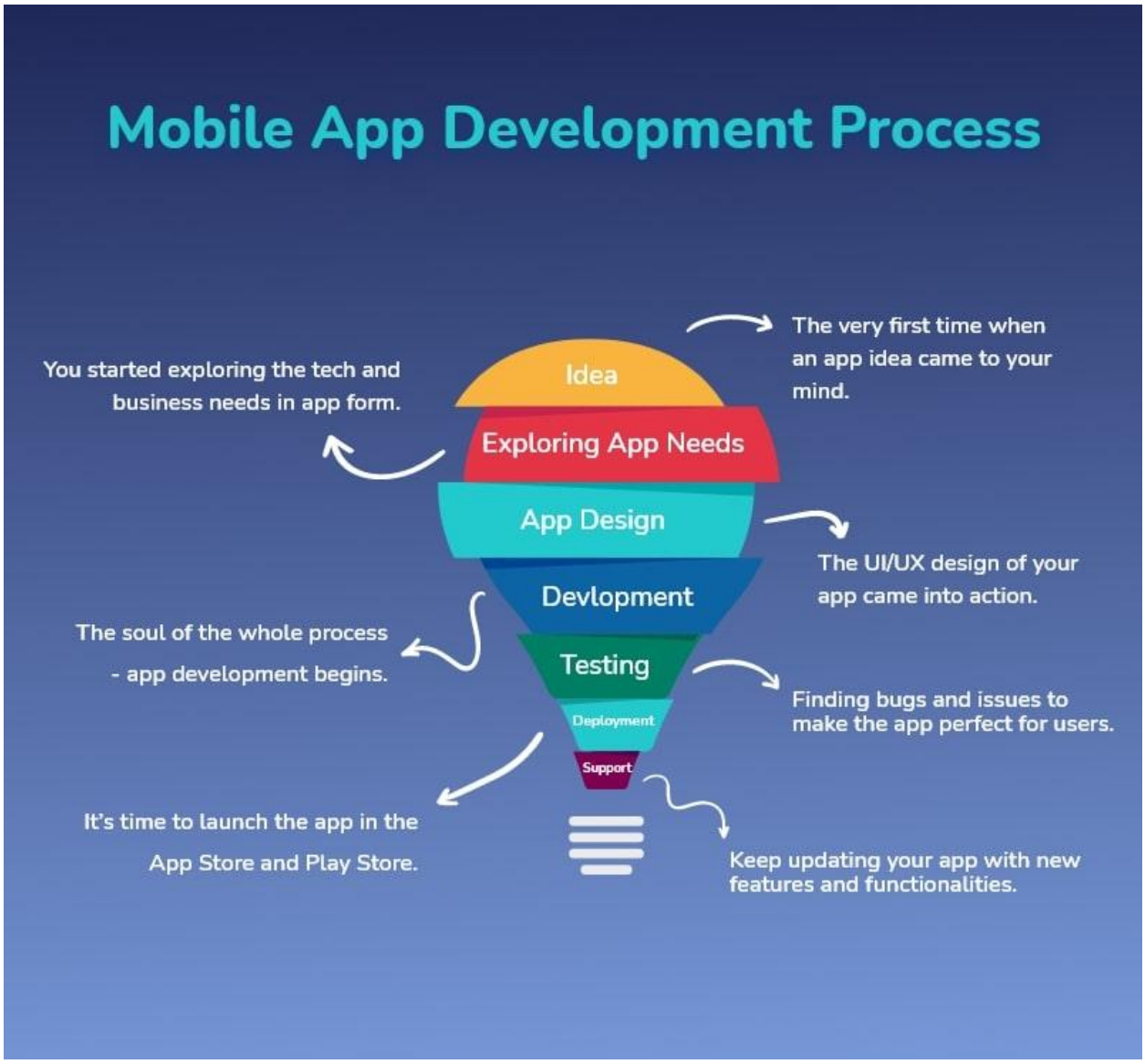

Рис. 1.4. Процес розробки мобільного додатку

1.4. Аналіз ідеї додатка

Кожен процес розробки програми починається з ідеї. Щоб мати чітке бачення застосування, потрібно багато мозкового штурму та досліджень. Чи зрозуміла ідея програми? Якщо це так, то можна безпосередньо перейти до наступного етапу.

Але якщо є сумніви, то потрібно отримати відповідь на п’ять запитань:

1. Яка основна мета додатка?
2. Які проблеми вирішить додаток?
3. Хто є цільовою аудиторією?
4. Чим програма буде відрізнятися від інших?
5. Чому люди повинні продовжувати використовувати додаток?

Саме після того, як вищезазначені моменти будуть зрозумілі, настане час побудувати стратегію створення додатку і провести дослідження ринку.

Створення стратегії для майбутнього мобільного додатка – це все одно, що планувати свою відпустку. Необхідно провести ретельне дослідження ринку у галузі, підібрати кращі варіанти для програми, а потім розрахувати бюджет і встановити тимчасові рамки.

Потрібно дізнатися про поточні ринкові тенденції та проблеми, з якими стикається аудиторія. Тоді остаточним кроком є вирішення проблем користувачів за допомогою розробки мобільного додатка. Далі продумати та дослідити переваги, які продукт приносить клієнтам. Іншими словами, вивчити конкурентне середовище та знайти своє прибуткове місце.

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.).

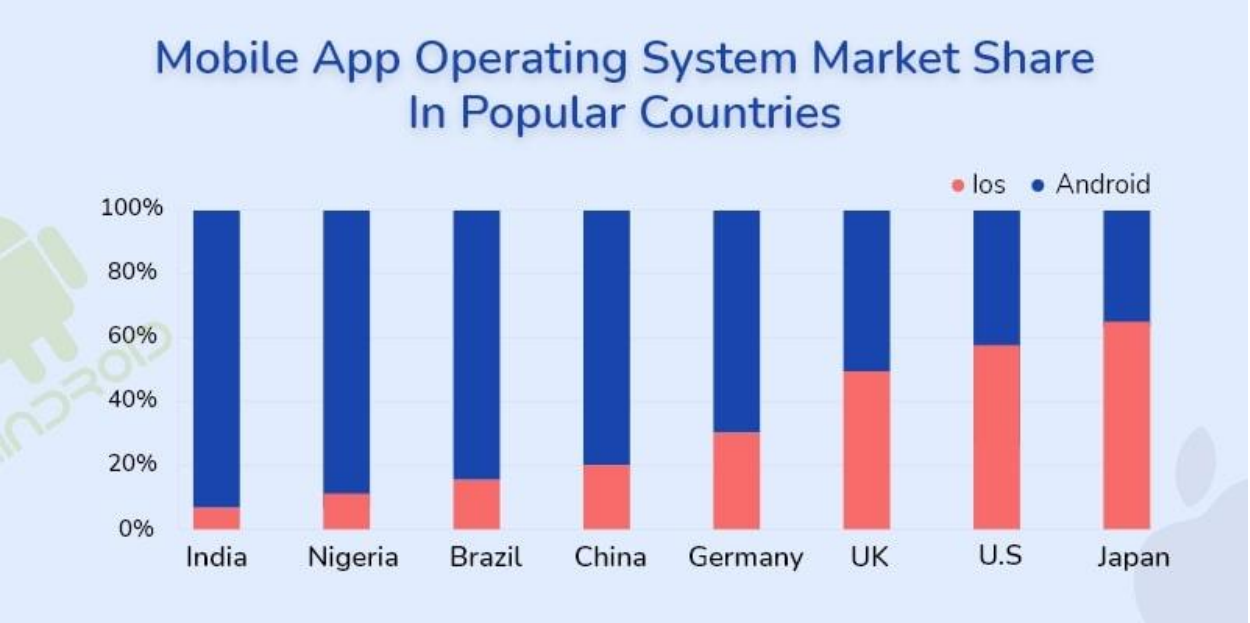

Рис. 1.5. Огляд користувачів мобільних пристроїв Android та iOS у всьому світі

Далі потрібно попрацювати над функціональними можливостями, функціями та бізнес-моделлю додатка. Існують різні типи бізнес-моделей, як-от бізнес-моделі на основі підписки, двосторонній ринок, бізнес-модель freemium, бізнес-модель покупки та реклами в програмі тощо.

Виходячи з галузі, потрібно почати аналізувати ключові функції, які будуть надані користувачам. За допомогою фірми з розробки мобільних додатків потрібно працювати над технологічним стеком. Стек технологій, також званий стеком рішень, технологічною інфраструктурою або екосистемою даних — це список усіх технологічних служб, які використовуються для створення та запуску однієї програми. Впровадження популярного й надійного технологічного стеку є домінуючим елементом для безперебійної роботи додатка.

1.6. Minimum viable product

Minimum viable product (MVP) — це метод розробки, за допомогою якого новий продукт або веб-сайт розробляється з достатніми функціями, щоб задовольнити ранніх користувачів. Остаточний повний набір функцій розробляється лише після врахування відгуків перших користувачів продукту. Існує чотири основні

елементи для створення продукту MVP-функціональність, читаність, дизайн і зручність використання. Нижче наведено список причин, чому потрібен продукт MVP:

- Тестування бізнес-концепцій;
- Один із найпростіших способів отримати відгуки користувачів про програму;
- Підкреслює ідею продукції та перевірити, як вона працює в режимі реального часу;
- Визначити масштаби вдосконалення;
- Створення покращеного продукту з мінімальною кількістю багів або без них;
- Можливість створити більш захищений додаток;
- Домомагає оцінити продуктивність програми та внесіть необхідні зміни в остаточну версію.

1.7. Підписання угоди про нерозголошення

Безпека - це основа будь-якого бізнесу, будь то онлайн або оффлайн. Перш ніж розпочати розробку мобільного додатку, потрібно підписати угоду про нерозголошення з фірмою-розробником програмного забезпечення.

Більшість провідних агентств з розробки додатків завжди підписують NDA перед початком розробки. У NDA міститься кожна деталь розробки програми. Що потрібно враховувати у NDA:

- Найменування сторін, які підписують угоду;
- Період часу реалізації проекту;
- Будь-які винятки з міркувань конфіденційності;
- Ідея проекту залишається в безпеці;
- Заява, в якій згадувалося про належне використання конфіденційної інформації;

- Визначення розуміння проекту, способу зв’язку, функцій програми, її технічного стека, посилань/додатків, обміну файлами і так далі.

1.8. Етапи розробки програми

Зовнішній вигляд мобільного додатку є одним з головних пріоритетів для користувачів. І ось тоді на сцену виходить дизайн UI/UX програми. Будь то першокласна розробка додатків або просто продукт MVP, користувальницький досвід є однійєю з найважливіших частин розобкки додатку, яка безпосередньо впливає на успіх програми.

Служба дизайну мобільних додатків розділена на багато частин. Так само, як і процес розробки додатків, дизайн UI/UX також має свій процес і підрозділи.

Перш ніж перейти до підрозділу проектування додатків, необхідно спочатку розібратися в двох важливих термінах проектування, а саме UI і UX.

1.8.1. Користувальницький інтерфейс

Користувальницький інтерфейс (UI) - це точка, в якій користувачі-люди взаємодіють з комп’ютером, веб-сайтом або додатком. Мета ефективного інтерфейсу користувача-зробити роботу користувача простою і інтуїтивно зрозумілою

Інтерфейс користувача створюється на рівнях взаємодії, які апелюють до людських почуттів (зору, дотику, слуху і багатьом іншим). Вони включають в себе як пристрої введення, такі як клавіатура, миша, трекпад, мікрофон, сенсорний екран, сканер відбитків пальців, електронне перо і камеру, так і пристрої виведення, такі як монітори, динаміки і принтери. Пристрої, які взаємодіють з кількома органами почуттів, називаються "мультимедійними інтерфейсами". Наприклад, повсякденний користувальницький інтерфейс використовує комбінацію тактильного введення (клавіатура і миша) і візуального і слухового виведення (монітор і динаміки).

1.8.2. UX-дизайн

UX-дизайн програми описується як спосіб взаємодії користувача з елементами інтерфейсу додатка. Основна мета дизайнера UX – зв'язати всі елементи користувальницького інтерфейсу і надати користувачеві зручну навігацію.

Крім того, певна ступінь ітеративного аналізу є частиною дизайну UX. Вони створюють каркасну візуалізацію своїх користувацьких взаємодій, а потім інтегрують її в свої проекти.

Єдине, на чому зосереджується дизайнер UX, - це цілісне розуміння того, як користувач хоче взаємодіяти з додатком. Вони дотримуються різних принципів дизайну інтерфейсу користувача, таких як принцип зворотного зв"язку, принцип повторного використання, принцип структури, принцип простоти, і список можна продовжувати.

Отож різниця між UI та UX 1 досить проста: UI дизайнери працюють над визначенням того, як буде виглядати інтерфейс користувача, тоді як UX дизайнери беруть на себе відповідальність за те, як працює інтерфейс користувача (рис. 1.6.).

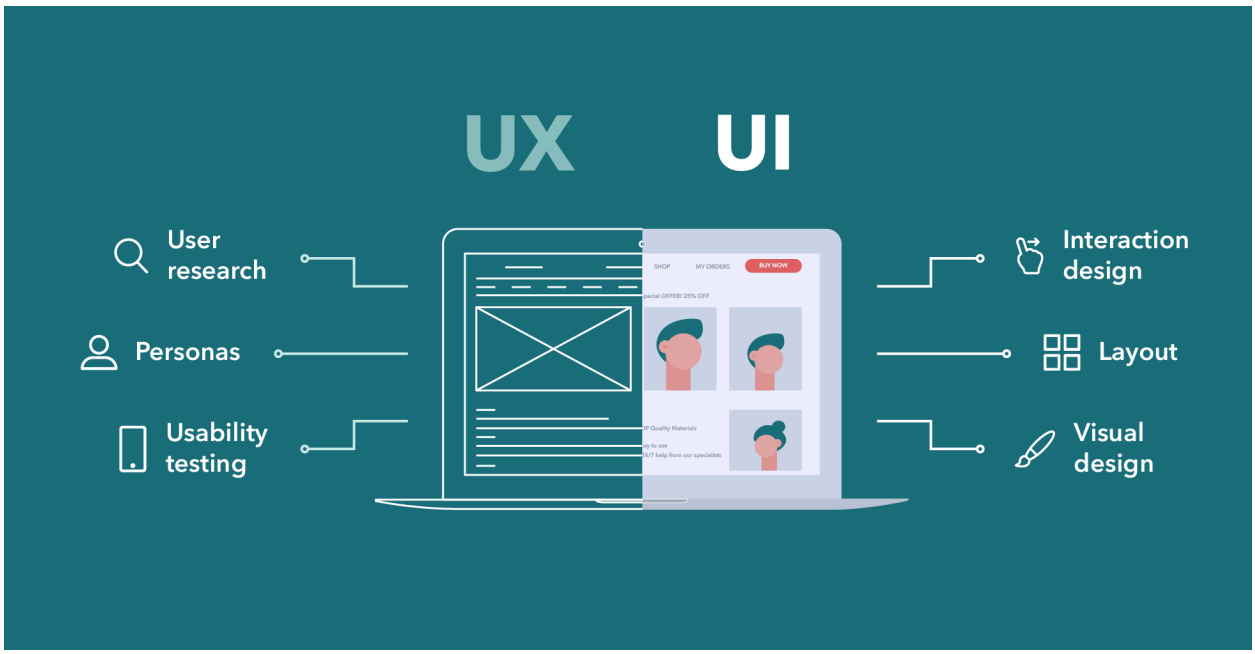

Рис. 1.6. Різниця між UI та UX

1.8.3. Каркасний дизайн програми

Каркас допомагає команді розробників представити своє бачення клієнтам і прийняти рішення про остаточні технічні та дизайнерські рішення.

На відміну від створення ескізу або прототипу програми (макета), створення каркасу продукту - це пошук потенційних дизайнерських та інженерних рішень, а також перевірка того, чи бачать клієнти та агентства з розробки майбутній продукт таким же чином.

Він направляє весь процес розробки програми. Каркас також можна назвати схематичною версією майбутнього продукту, каркасною структурою Програми, а також кресленням програми (рис. 1.7.).

Рис. 1.7. Приклад каркасу

Каркас - це ілюстрація інтерфейсу програми, представлена в 2-мірній формі. Це показує, як люди повинні використовувати продукт. Якщо щось було втрачено на

етапі ескізу, каркас допомагає виділити це, тим самим покращуючи додаток і уникаючи дорогих перебудов коду.

1.8.4. Посібник зі стилю

Посібник зі 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.

Деякі важливі елементи, включені в посібник із стилю інтерфейсу користувача, - це шрифти, кольори, макети, шаблони дизайну, бібліотеки компонентів, графіка, діалоги, зміни, кнопки тощо.

Посібники зі стилю - це більше, ніж документація. Вони являють собою основу для рішень з проектування та реалізації, які впливають на мобільний додаток не тільки в інтерфейсі користувача, але і в UX. Корисним керівництвом по стилю має бути:

- Докладний і інформаційний - точні специфікації та інструкції для кожного елемента;
- Організований - розділи, компоненти та деталі є чіткими та згрупованими;
- Адаптований до майбутніх сценаріїв - важливо враховувати майбутні ситуації в керівництві по стилю, такі як можливі розширення макетів або варіанти значків;
- Специфічний для платформи - повинно містити графіку і компоненти в інформаційних стандартах платформи;
- Простий у використанні і розумінні - має бути доступним для всіх можливих сторін. Використовувана мова повинна дотримуватися власниками продуктів, дизайнерами, розробниками та учасниками SQA.

1.8.5. Макет

У світі дизайну веб-сайтів та мобільних додатків макет - це високоякісне моделювання того, як виглядатиме веб-сайт або мобільний додаток. Макети поєднують структуру та логіку каркасу з високоякісною графікою та елементами інтерфейсу користувача. Таким чином, макети є проміжним етапом у розробці продукту, з набагато більшою візуальною деталізацією, ніж каркас, але без повної функціональності прототипу.

Макети додатків і веб-сайтів корисні, тому що вони допомагають створити майже остаточний дизайн всього мобільного додатка або веб-сайту, не займаючись важкою роботою по створенню складних функцій. Завдяки їх візуальному впливу, макети є відмінним ресурсом для обміну з клієнтом, щоб він міг перевірити прийняті дизайнерські рішення і їх вплив на користувальницький інтерфейс, перш ніж починати працювати над внутрішньою роботою продукту.

Макети мобільних додатків і веб-сайтів також можуть бути корисні для початкового етапу тестування користувачів. Хоча користувачі не можуть взаємодіяти з макетом, вони можуть висловити свою думку про дизайн на відносно ранній стадії розробки продукту. Отримання негативних відгуків користувачів або клієнтів на цьому етапі буде набагато дешевше, ніж отримання їх пізніше.

1.8.6. Прототип

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.

Прототипи - це цінні інструменти перевірки та тестування, які дозволяють перевірити доцільність концепції програми. Одна з найскладніших речей, яку потрібно виправити в додатку, - це інтерфейс користувача (UX).

На щастя, прототипування є винятковим засобом отримання інформації про UX. Оскільки реальні користувачі тестують додаток, то отримується набагато більш точний зворотний зв"язок про користувальницький інтерфейс.

На які питання може відповісти прототипування:

- Чи працює колірна схема?
- Чи можуть користувачі інтуїтивно орієнтуватися в додатку?
- Чи краще використовувати два екрани для певної функції, ніж використовувати тільки один?

Прототипування також дозволяє більш точно порівнювати два або більше варіантів інтерфейсу програми.

Складно вирішити, чи має навігаційне меню розташовуватися праворуч або внизу екрану? Можна протестувати їх за допомогою двох прототипів і подивитися, який з них більше подобається користувачам.

Прототипування - це не тільки відмінний інструмент тестування, але і відмінна дорожня карта проекту. Це дає візуальний інструмент для передачі дизайну, процесу розробкиі функцій програми. Функції та елементи менш відкриті для інтерпретації, що значно зменшує кількість помилок і непорозумінь.

Прототип також залучає людей. Кожен член команди розробників, і навіть постачальники і клієнти, можуть поділитися своїми пропозиціями.

Одним з найбільш значних витрат часу і грошей на будь-який проект розробки є доопрацювання і зміна готового додатка. У більшості випадків це викликано функціями або користувальницьким інтерфейсом, які не відповідають очікуванням під час приймального тестування. Цих змін, як правило, важко уникнути.

Однак прототипування може перевірити ці припущення на самих користувачах. Можливо краще визначити потенційні проблеми і, таким чином, усунути їх на більш ранніх етапах процесу розробки.

1.9. Вибір архітектури мобільного додатку

Мобільна архітектура - це сукупність правил, методів і шаблонів розробки мобільного додатку. Цей набір дозволяє створювати логічні і добре структуровані програми, що відповідають вимогам клієнтів і галузевим стандартам.

При створенні мобільного або веб-додатки вирішальне значення має забезпечення того, щоб кожен компонент виконував свою функцію. Невелика проблема на етапі створення архітектури мобільного додатка може мати далекосяжні наслідки для життєздатності кінцевого продукту.

Під час обговорення архітектури розробки мобільних додатків архітектори розглядають кращі галузеві практики та стандарти. Розробникам необхідно враховувати всі типи бездротових продуктів в цьому аналізі, щоб додаток легко працювало як зі смартфонами, так і з планшетами.

З простої точки зору додатки розробляються з урахуванням трьох різних рівнів (рис. 1.8.). **Рівень даних** включає в себе всі утиліти обробки даних, Сервісні агенти і компоненти доступу до даних. Переходячи від даних, **Бізнес-рівень** включає в себе всі різні робочі процеси і бізнес-об'єкти, а також самі бізнес-компоненти. Нарешті, **рівень представлення** - це місце, де знаходяться процеси та компоненти інтерфейсу користувача (UI).

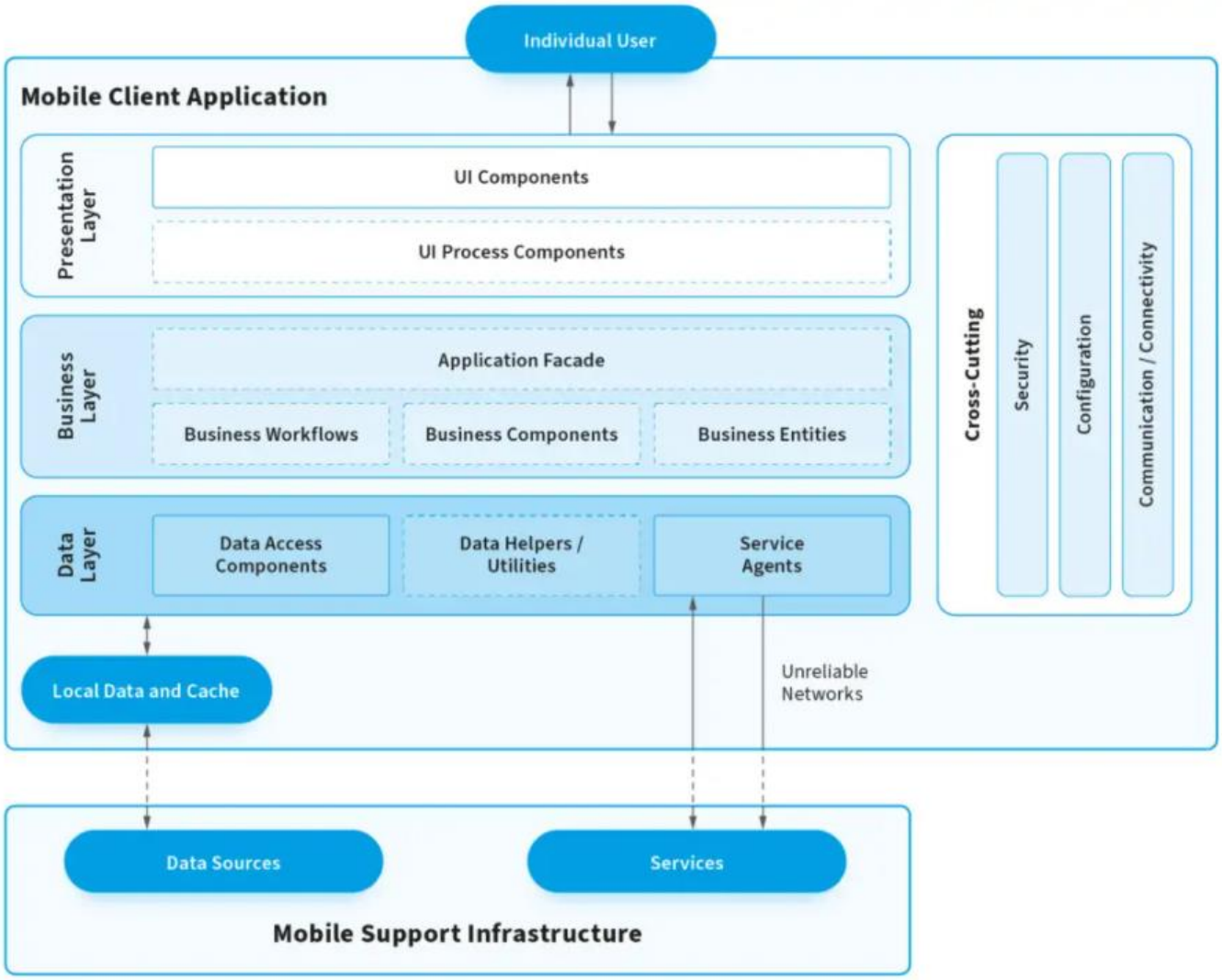

Рис 1.8. Архітектурні рівні

1.9.1. Особливості гарної архітектури

Визначимо деякі особливості гарної архітектури:

- **Збалансований розподіл між ролями**. Аби знизити зв'язанність, обов'язки різних об'єктів повинні бути розділені між різними компонентами. Усе повинно слідувати принципу єдиної відповідальності;
- **Тестованість**. Тестованість не означає тестування. Додаток можна тестувати, коли існує ефективна стратегія тестування, яка використовується для перевірки відповідності деякої реалізації щодо її специфікації. Написання автоматичних тестів стає дуже простим, тому що в той час, коли дороблюється елемент композиції, він стає готовим до незалежного тестування. Ці тести дозволяють розробникам знаходити і

виправляти помилки до того, як додаток буде передано на пристрій користувача;

- **Простота використання і експлуатаційна надійність**. Чим менше коду пишеться, тим менше шансів, що щось піде не так. Чим більше коду, тим більше місць для багів. Якщо код поганий, він буде тягнути за собою увесь проект, тому не слід шукати швидкі рішення, закриваючи очі на більш пізні витрати на обслуговування. Новому розробнику потрібно якомога швидше та простіше освоїтися у проекті.

1.9.2. Чому важливо дбати про вибір архітектури?

Якщо не вибирати архітектуру правильно, то одного дня знайти та виправити баги стане складним та довгим процесом. Звичайно, можна не звертати увагу на вибір архітектури для досить простих програм з декількома екранами та невеликою базою коду, де все можна вмістити в один контролер. Але що, якщо це складна програма с можливістю зростання кількості функціионалу? Тоді можна отримати величезну купу коду в контролері представлення, що робить його не "Model View Controller" а так званим "Massive View Controller". Це може статися, якщо дотримуватися вказівок Apple MVC.

1.9.3. Огляд Apple MVC

Apple запропонували використовувати архітектурний шаблон MVC (модель-представлення-контролер) для UIKit, хоча багато експертів вважають, що це не найкраще рішення.

Шаблон MVC (рис. 1.9.) складається з трьох основних об'єктів:

- **Модель**. Шар, на якому зберігаються дані;
- **Подання**. Шар, який візуально представляє додаток. Його класи не містять жодної специфічної для домену логіки, тому вони вважаються багаторазовими;

- **Контролер**. Проміжний шар між поданням і Model. В ідеалі Controller взаємодіє з абстракцією по протоколу і не знає конкретного уявлення, з яким він має справу.

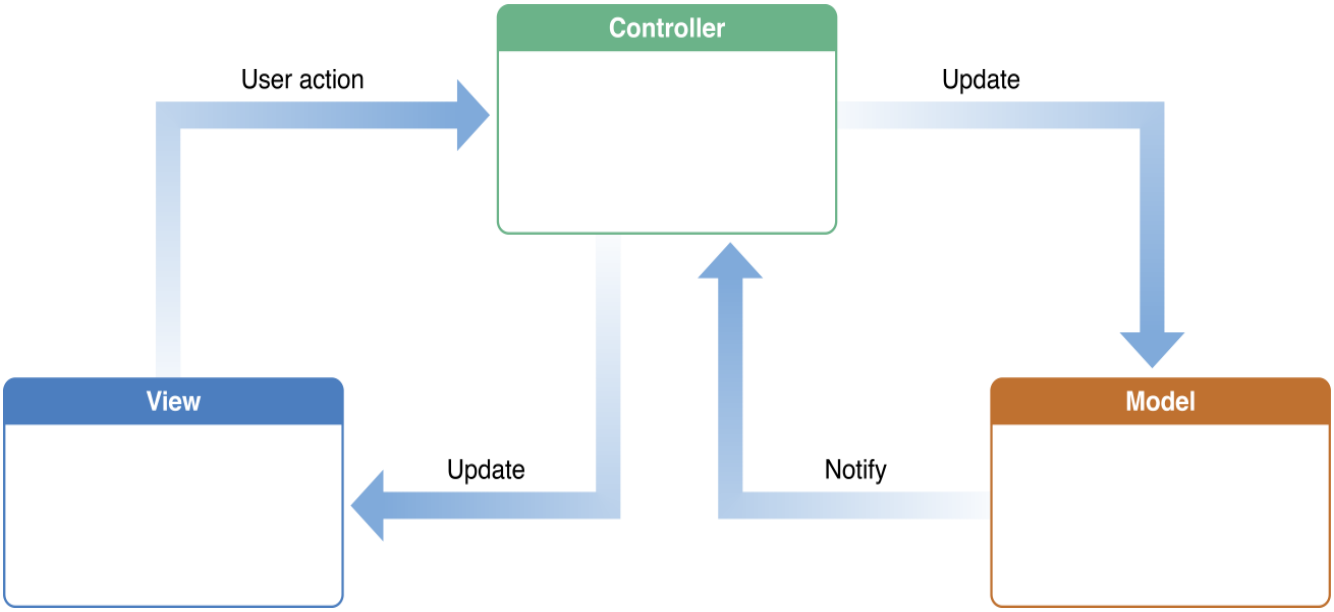

Рис 1.9. Apple MVC

MVC спочатку призначався для розподілу компонентів додатків по окремих частинах. Але проаналізувавши роботу з ним можна виділіти наступні недоліки:

- **Відсутність розподілу**: контролер в кінцевому підсумку виконує всю роботу від обробки користувацьких взаємодій до налаштування уявлень, виконання мережевих викликів, аналіз даних, тощо. Іншими словами перетворюється на Massive View Controller;
- **Низьке охоплення тестуванням**: крім порушення принципу єдиної відповідальності, зв'язанність подання та контролеру збільшується. Таким чином тестування перетворюється на складний процес, який у одаток ще й займає багато часу.

1.9.4. MVP як альтернатива MVC

Архітектура MVP поліпшить ситуацію з Apple MVC додавши основний компонент Presenter (рис. 1.10.).

Рис 1.10. архітектура MVP

Ця архітектура схожа на MVC але з ключовою відмінністю - тепер ViewController розглядається як View. Це означає, що він буде мати тільки код, пов'язаний з View, і нічого більше. А вся логіка буде реалізована в Presenter.

Тоді ролі трохи зміняться:

- **View** тепер складається як з уявлень, так і з контролерів подання з усіма налаштуваннями інтерфейсу користувача та подіями;
- **Presenter** буде відповідати за всю логіку, включаючи реагування на дії користувача та оновлення інтерфейсу користувача (через делегування). і найголовніше, що наш ведучий не буде залежати від UIKit. що означає добре ізольований, отже, легко тестований;);
- **Model** буде точно такою ж.

Важливо відзначити, що MVP використовує шаблон пасивного View. Це означає, що всі дії будуть перенаправлені до Presenter. Що призведе до запуску оновлень користувальницького інтерфейсу з використанням делегатів. Таким чином, View буде тільки виконувати дії і прослуховувати оновлення від Presenter.

Саме ця архітектура і буде використана для створення додатка.

1.10. Розробка мобільного додатку

Розробка додатку розділена на три основні кроки. Перш ніж перейти до детальної версії, проаналізуємо етапи процесу розробки програми.

Мобільний додаток побудовано на трьох основних компонентах, тобто серверних технологіях (Backend), API (інтерфейсах прикладного програмування) і розробки інтерфейсу (Frontend).

Елементи, які користувачі бачать і з якими взаємодіють - це інтерфейс програми, розробка для роботи на стороні сервера-це бекенд, а набір функцій, який дозволяє додаткам отримувати доступ до даних і взаємодіяти з зовнішніми програмними компонентами, операційними системами або мікросервісами це API.

1.10.1. Розробка серверної частини

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

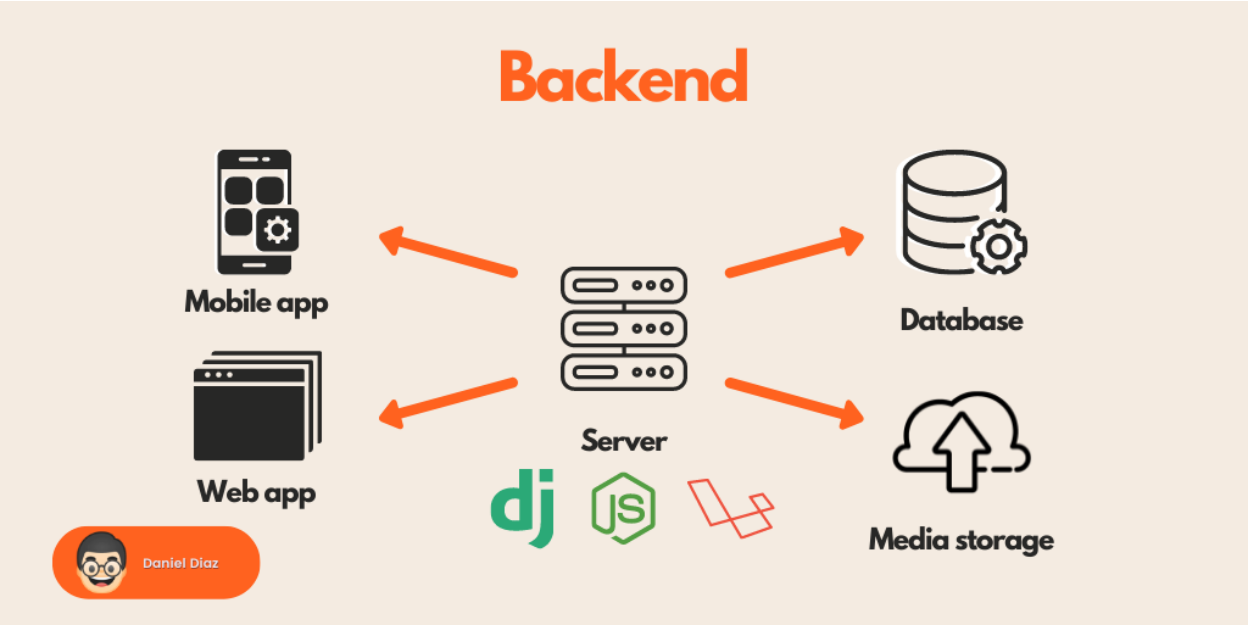

Рис. 1.11. Серверна частина

Серверний розробник - це кваліфікований розробник програмного забезпечення, відповідальний або достатньо кваліфікований, щоб розуміти, планувати, розробляти та тестувати серверну/бізнес-логіку програми (рис. 1.11.). Спільно з іншими членами команди він відповідає за вибір найкращих і відповідних інструментів і технологій для даного проекту.

1.10.2. API

API - це набір програмного коду, який забезпечує передачу даних між одним програмним продуктом та іншим (рис. 1.12.). У ньому також містяться умови цього обміну даними.

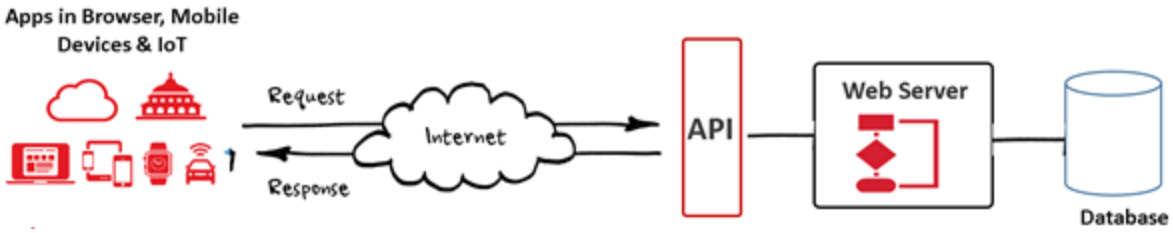

Рис. 1.12. Приклад роботи API

Інтерфейси прикладного програмування складаються з двох компонентів:

- Технічна специфікація, що описує варіанти обміну даними між рішеннями зі специфікацією, виконаною у формі запиту на обробку і протоколів доставки даних;
- Програмний інтерфейс, написаний відповідно до специфікації, яка його представляє.

Програмне забезпечення, якому потрібно отримати доступ до інформації (наприклад, до цін на номери в готелях на певні дати) або функцій (наприклад, до маршруту з пункту а в пункт Б на карті, заснованому на місцезнаходження користувача) з іншого програмного забезпечення, викликає свій API, вказуючи вимоги про те, як повинні надаватися дані/функції. Інше програмне забезпечення повертає дані/функції, запитані попереднім додатком.

1.10.3. Розробка інтерфейсу

Розробка інтерфейсу (Фронт-енд) відноситься до тієї області веб-розробки, яка фокусується на тому, 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

Розробник інтерфейсу зосереджується на тому, що відбувається в передній частині програми, а не на задньому плані. Розробники беруть участь у перетворенні UI/UX дизайну програми в коди, щоб правильно продемонструвати його на екрані.

Фронт-енд розробники працюють над аналізом коду, проектуванням і налагодженням програм разом із забезпеченням безперебійної роботи для користувачів.

Наприклад, відкриваючи додаток YouTube, можна побачити попередні перегляди, відео тощо. Що відбувається при натисканні на відео? У дію вступають деякі візуальні та чуттєві відчуття. 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. Гібридні програми можуть працювати в обох операційних системах з

однаковим кодом. Flutter, React Native та Ionic — найкращі мови для розробки гібридного додатка.

1.10.4. Безпека мобільного додатку

Безпека мобільних додатків – це міра та засіб захисту додатків для мобільних пристроїв від цифрового шахрайства у вигляді зловмисного програмного забезпечення, злому та інших злочинних маніпуляцій. Безпека мобільних додатків може бути реалізована як технологічними засобами, так і поряд з особистими відповідями, і корпоративними процесами, призначеними для захисту цифрової цілісності на мобільних пристроях.

Мобільні додатки часто стають мішенню злочинних елементів, які прагнуть отримати прибуток від компаній і співробітників, які використовують мобільні пристрої, але не беруть участь у належних процесах безпеки мобільних додатків.

Коли мобільний додаток скомпрометовано зловмисним програмним забезпеченням або користувач пристрою завантажує неавторизовану шахрайську програму, яка насправді не запущена офіційно, вони мають високий ризик стати жертвою цифрового шахрайства. Наприклад:

- Облікові дані для фінансового входу викрадають;
- Дані кредитної картки вкрали та перепродали;
- Надання хакерам доступу до їхніх бізнес-мереж;
- Оптова крадіжка особистих даних;
- Їхній пристрій використовується для поширення шкідливого програмного забезпечення на незаражені пристрої;
- Копіювання та сканування приватної інформації повідомлень TXT або SMS.

1.10.5. Рекомендації щодо забезпечення безпеки мобільних додатків

1. **Навчання з цифрової безпеки.** Працівники повинні знати про ризики, які можуть представляти мобільні додатки. Потрібно навчити їх розпізнавати

потенційні атаки, шкідливі сайти і спроби фішингу, а також застосовувати належні процедури реагування.

2. **Активний моніторинг програм-шахраїв.** Необхідно слідкувати як за законними, так і за несанкціонованими платформами завантаження додатків для будь-яких додатків з фірмовим найменуванням додатку що рощроблюється, логотипом або повідомленнями, які могли бути розміщені для залучення нічого не підозрюючих клієнтів. Важливо видалити всі шахрайські програми якомога швидше.
3. **Завантажувати тільки з надійних джерел.** Працівники та клієнти повинні мати список перевірених сайтів для завантаження додатків. Але навіть і в цьому випадкуважилво проявляти підвищену обережність при завантаженні нового додатка і повідомляти про будь-які підозрілі дії.
4. **Покращення безпеки даних.** Встановлення специфічної для бренду стратегії захисту даних і політику, яка передбачає активне збирання та усунення всіх можливих порушень даних.
5. **Уникання збереження паролів.** Не рекомендується використовувати програми, які зберігають паролі у системі або в хмарі, оскільки вони можуть дозволити збирати особисті облікові дані і використовувати їх для злому інших пристроїв або мереж.
6. **Примусове завершення сеансу користувача.** Ніколи не можна дозволяти сеансу користувача залишатися активним після того, як він вийшов з системи або закрив додаток.
7. **Виходити за рамки захисту від шкідливих програм.** Багато ресурсів безпеки мобільних додатків в першу чергу сканують пристрої на наявність відомих шкідливих програм і попереджають користувача про можливість видалити все знайдене. Хоча це відмінний запобіжний захід, корпоративні заходи цифрової безпеки не повинні зупинятися на досягнутому. Можна підключити процедури шифрування, інструменти аналізу поведінки, моніторинг трафіку і багато іншого.

1.11. Тестування

Тестування мобільних пристроїв-це процес, за допомогою якого мобільні додатки перевіряються на функціональність, зручність використання та узгодженість (рис 1.14.).

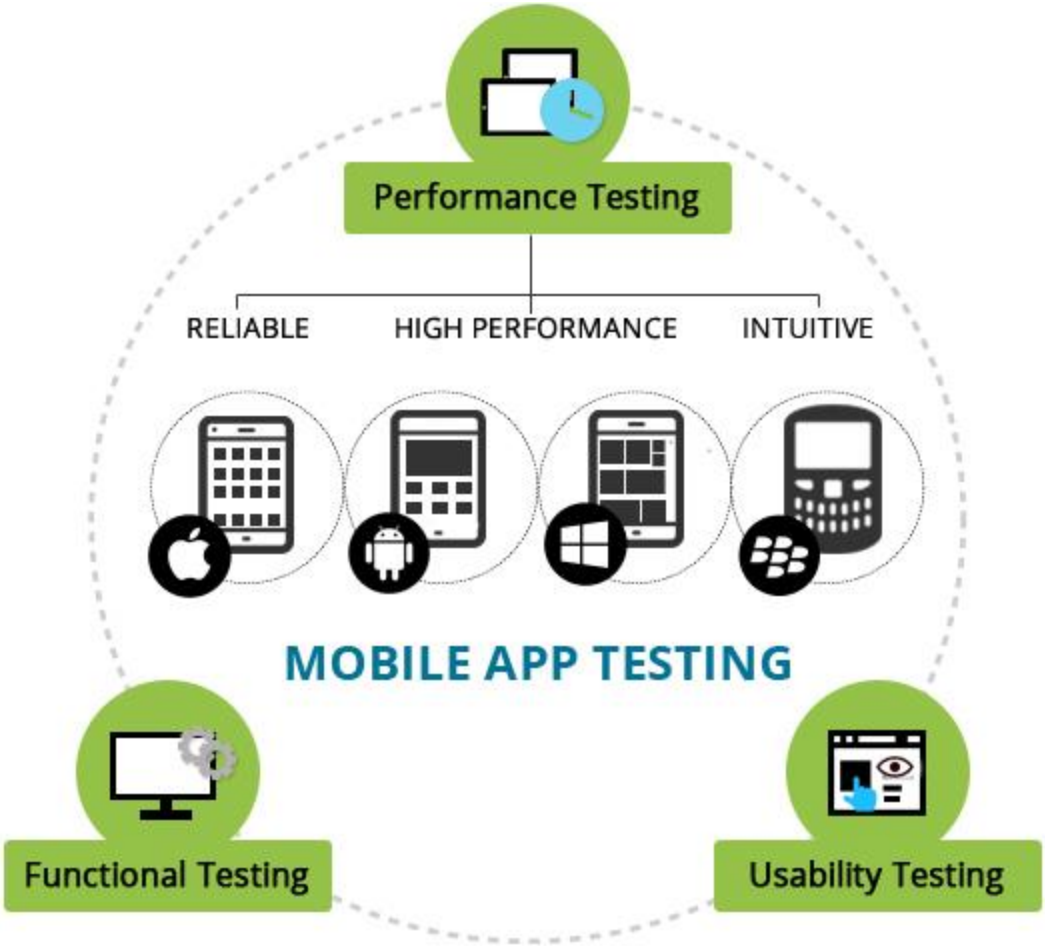

Рис. 1.14. Тестування

Чому важливо мобільне тестування? Сьогоднішні споживачі звикли очікувати безперебійної роботи з високоякісними додатками. Поганий досвід може призвести до втрати користувачів, а також завдати шкоди репутації бренду. Тому комплексна стратегія тестування мобільних додатків має вирішальне значення для процесу розробки.

Якщо не перевіряти можливості мобільних пристроїв, які отримують користувачі, не можливо знати, наскільки добре додаток обслуговує користувачів. Випуск додатку без тестування призводить до жахливих відгуків з однією зіркою і негативних відгуків в соціальних мережах.

Тестування мобільних додатків гарантує, що досвід використання користувачем буде надійним, незалежно від того, який додаток використовується або для якої платформи він був розроблений.

Існує два типи основного тестування для розробки мобільних додатків - функціональне і нефункціональне (рис. 1.15.).

Функціональне тестування - це тип тестування, метою якого є встановлення того, чи працює кожна функція програми відповідно до вимог до програмного забезпечення. Кожна функція порівнюється з відповідною вимогою, щоб визначити, чи відповідає її результат очікуванням кінцевого користувача. Тестування проводиться шляхом надання вибіркових вхідних даних, отримання результуючих вихідних даних і перевірки того, що фактичні вихідні дані збігаються з очікуваними вихідними даними.

Нефункціональне тестування перевіряє всі аспекти, не охоплені функціональними тестами. Воно включає в себе продуктивність, зручність використання, масштабованість і надійність програмного забезпечення. Проводиться нефункціональне тестування, щоб переконатися в дотриманні інтересів кінцевого користувача. Продукт не стане успішним, якщо не вдасться виправдати очікування клієнтів. Нефункціональне тестування має важливе значення для підвищення ринкової вартості продукту.

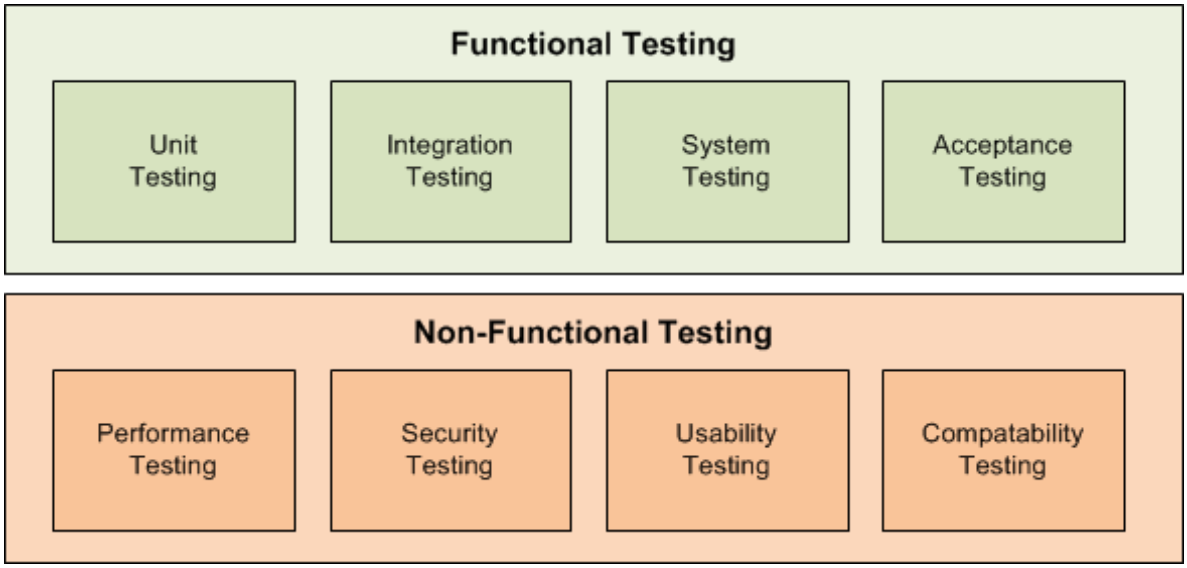

Рис. 1.15. Функціональне та нефункціональне тестування

1.12. Запуск програми та її підтримка

Виходячи з основних цілей, додаток перенаправляється в Google Play або магазин додатків Apple для запуску програми. Однак правила запуску додатків трохи відрізняються для обох цих платформ.

Проаналізуємо запуск додатків для iOS:

1. Зареєструватися в програмі Apple для розробників.
2. Створити обліковий запис iTunes Connect.
3. Підготувати додаток до розповсюдження.
4. Надіслати свою заявку на остаточний розгляд.
5. Автоматично опублікувати мобільний додаток.

Як тільки опублікується додаток в App Store, він пройде перевірку, яка може зайняти від декількох днів до декількох тижнів. Це залежить від того, наскільки точно додаток відповідає рекомендаціям щодо розробки додатків для iOS.

На відміну від iOS, в Android(Google Play) відсутні процеси перевірки. Додаток доступний в Play Store через кілька годин після його відправки.

Після опублікування додатку на платформах, які відповідають за контакт із користувачами, потрібно додати нові функції та версії додатка, щоб він залишалося

привабливим, і саме тут в гру вступають оновлення додатків. У певному сенсі, як тільки додаток виходить на ринок, починається процес розробки нового додатка.

Висновок до першого розділу

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.

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.

.

РОЗДІЛ 2
ІНСТРУМЕНТАРІЙ

2.1. Figma

Figma - це веб-додаток для редагування графіки та розробки інтерфейсу користувача. Його можна використовувати для виконання всіх видів робіт з графічного дизайну, починаючи з веб-сайтів з каркасами, розробки інтерфейсів мобільних додатків, створення прототипів, створення постів в соціальних мережах і всього іншого.

Figma відрізняється від інших інструментів редагування графіки. Головним чином тому, що він працює безпосередньо у вашому браузері. Це означає, що доступ до файлів проекту можна отримати з будь-якого девайсу.

2.2. Середовище розробки Xcode

Xcode - це програма для macOS, створена компанією Apple для розробки додатків. Це єдиний офіційний інструментарій для розробки додатків для IOS чи інших ОС Apple. Xcode використовується для написання коду та створення користувацьких інтерфейсів (рис. 2.1.).

Він пропонує безліч різних функцій і переваг для розробки додатків iOS та включає в себе інструменти, які допомагають розробнику на кожному етапі інженерного процесу.

Дана програма дає можливість створювати застосунки, які у подальшому могли б бути опубліковані у магазині додатків Apple. Використовувати його може як початківець, так і розбробник із багаторічним досвідом.

Xcode включає в себе всі інструменти, необхідні для створення програми в одному програмному пакеті, а саме: текстовий редактор, компілятор і систему збірки. За допомогою Xcode можна писати, компілювати і налагоджувати додаток, а пісся закінчення, можна відправити його в Apple App Store. Він містить ряд інструментів, які допомагають швидко просувати процес розробки, тому досвідчені розробники можуть створювати додатки блискавично, а новачки стикаються з меншою кількістю плутанини і перешкод на шляху створення відмінного додатка.

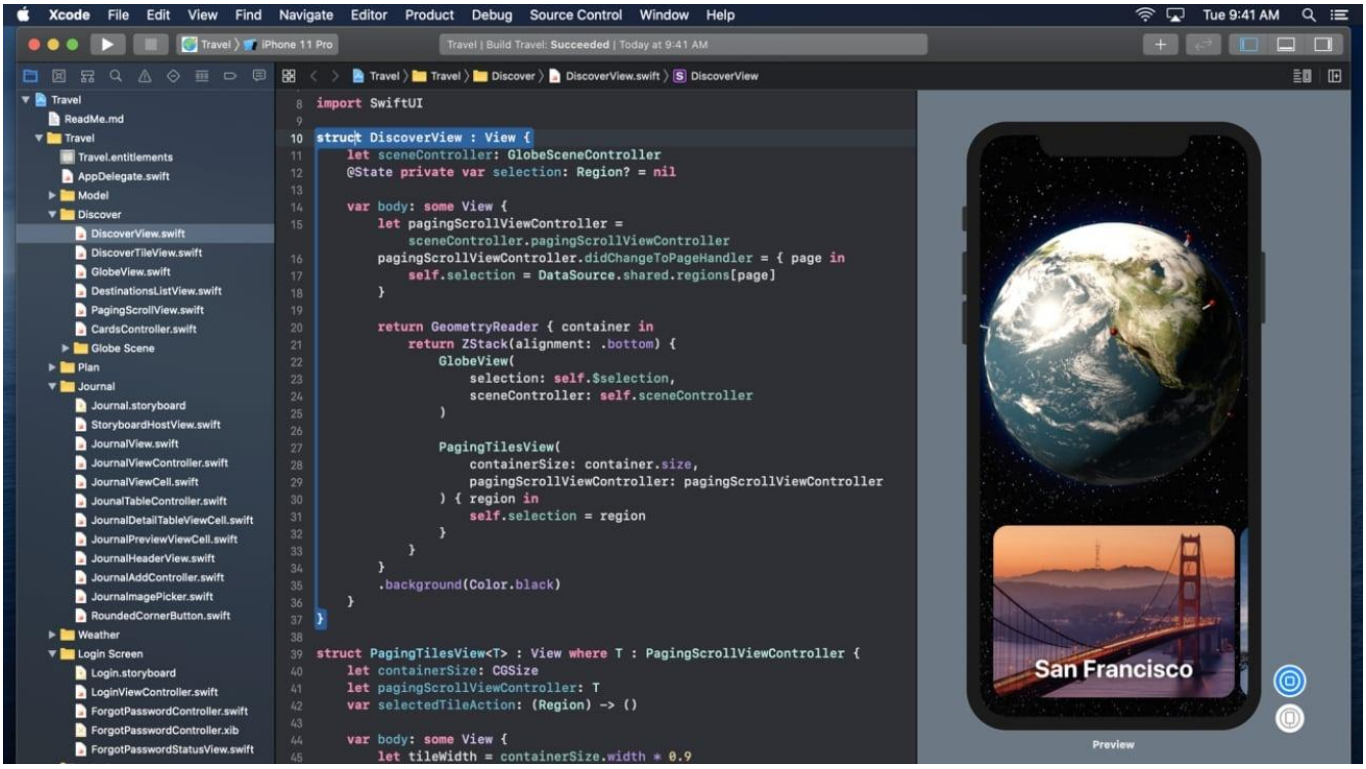

Рис. 2.1. Вигляд Xcode

Як редактор коду, Xcode підтримує величезну різноманітність мов програмування - C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit і Swift. Він використовує моделі програмування Cocoa, Carbon і Java.

Xcode призначений для того, щоб надати розробнику одне єдине вікно для роботи. У ньому є функція перевірки вихідного коду і автозаповнення, що значно спростить написання вихідного коду. При створенні нового проекту є можливість вибрати один з доступних шаблонів, щоб надати базову основу для розширення. Ці функції корисні для нових розробників, оскільки вони дають опору, на яку ви можна спертися в процесі навчання. Розробники із досвідом знайдуть ці функції корисними для оптимізації свого робочого процесу і значно прискорять процес розробки додатків.

Xcode - єдиний підтримуваний Apple спосіб розробки додатків. Існують сторонні рішення, які не вимагають використання Xcode, однак вони не підтримуються Apple, які можуть створювати пробелми при подальшій реалізації коду.

Xcode у світі програмування характеризується відмінними інструментами налагодження, які дозволяють розробникам швидше вирішувати проблеми в своєму додатку. Також він є корисним при організації управління ресурсами зображень і файлів коду завдяки вбудованим інструментам (рис. 2.2.).

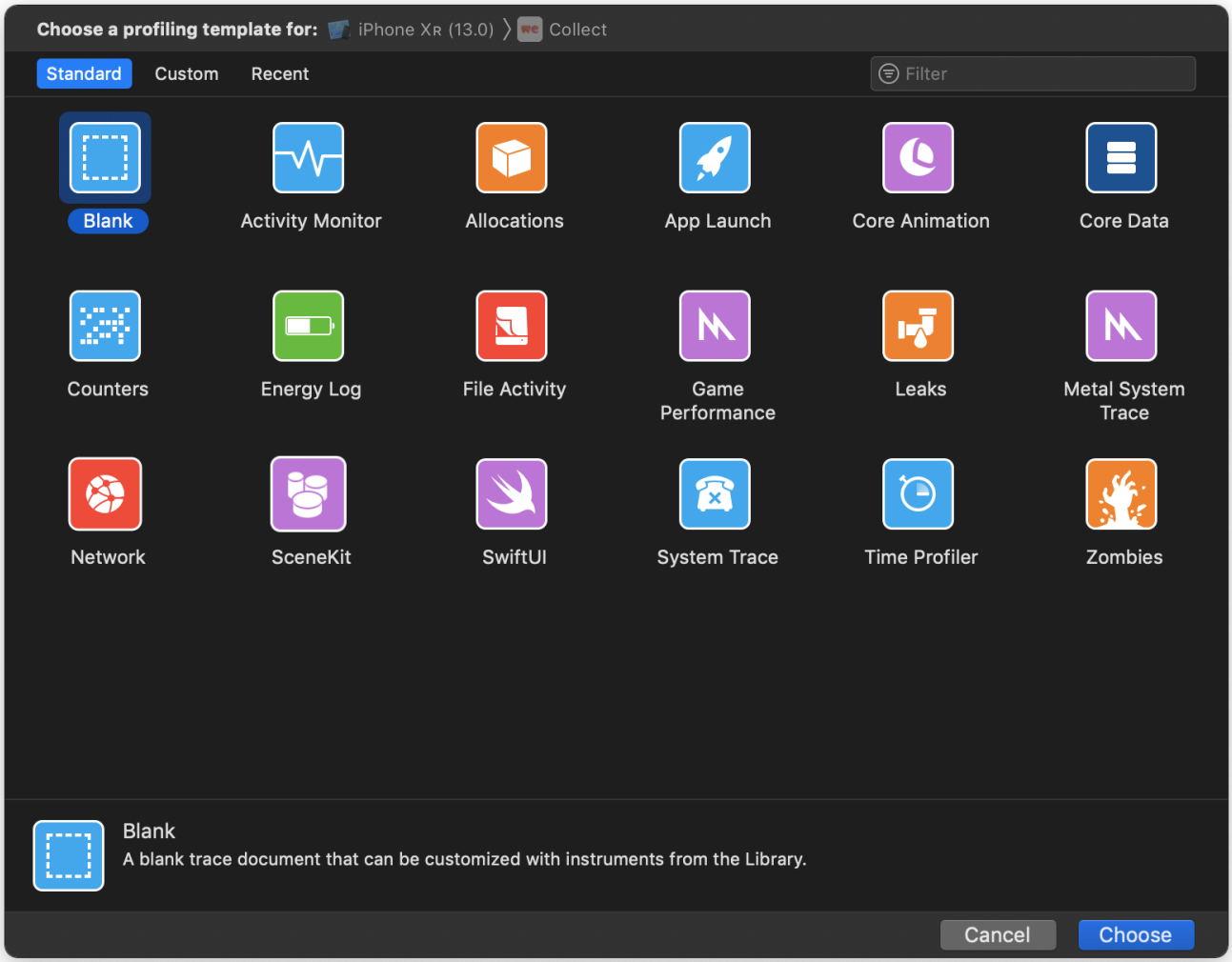

Рис. 2.2. Панель інструментів

Хоча існують сторонні IDE, які дозволять створювати додатки для iOS за межами MacOS, вони не справляються з тестуванням і налагодженням. Для створення ефективноого функціонального додатку на IOS необхідним елементом розробики є тестування та налагодження, яке відсутнє у сторонніх IDE.

Xcode має вбудований інструмент налагодження. Цей інструмент буде запускати додаток в режимі реального часу, а також дозволить переглядати вихідний код підрядково, щоб було можливо перевірити наявність будь-яких помилок. Також можна побачити, скільки процесора використовує додаток і скільки ресурсів він використовує на пристрої в порівнянні з іншими запущеними додатками. Навігатор тестів виконає будь-які додаткові тести, які буде потрібно виконати (рис. 2.2.).

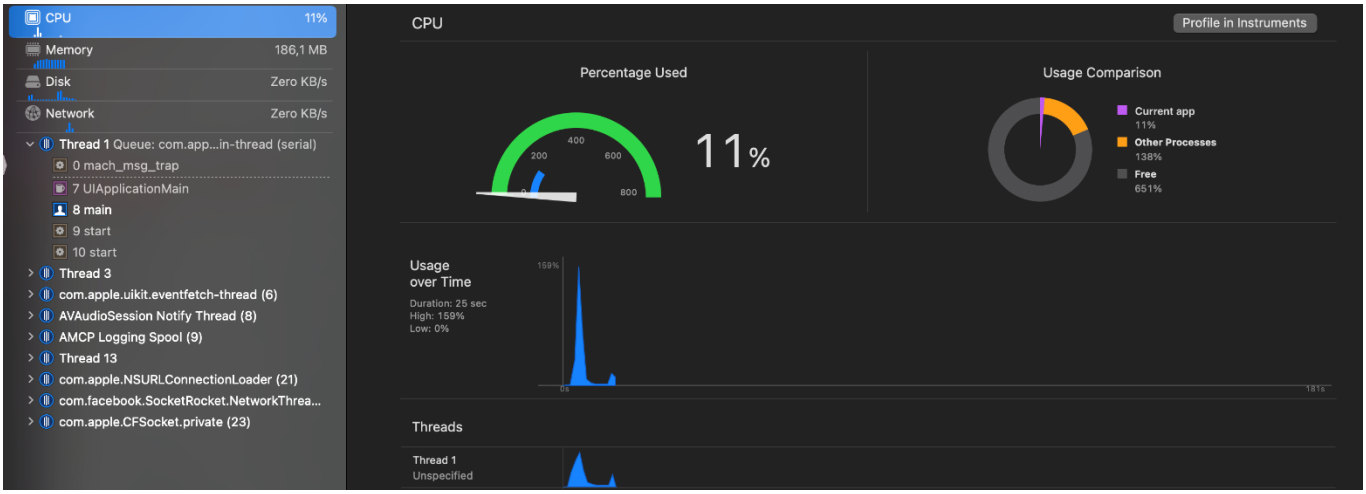

Рис. 2.2. Моніторинг ресурсів

2.3. Swift – мова розробки

Swift - це високопродуктивна системна мова програмування. Вона має чистий і сучасний синтаксис, забезпечує безперешкодний доступ до коду та різних фреймворків використовуючих мови C і Objective-C та є безпечною для пам'яті.

Рис. 2.3. Логотип Swift

Незважаючи на те, що Swift (рис 2.3.) натхненний Objective-C і багатьма іншими мовами, він сам по собі не є мовою, похідною від C. Як повна і незалежна мова, Swift об'єднує основні функції, такі як управління потоками, структури даних і

функції, з високорівневими конструкціями, такими як об'єкти, протоколи, замикання і узагальнення. Swift охоплює модулі, усуваючи необхідність в заголовках і дублюванні коду, яке вони тягнуть за собою.

Swift підтримує сумісність з існуючими програмами, закодованими в Objective-C, і працює з платформами API Cocoa і Cocoa Touch для пристроїв macOS і IOS. У той же час мова позбавляє частину багажу мови С для менш детального та більш ефективного коду. Swift працює краще, ніж Python, в 3,9 рази швидше, сортуючи складні об'єкти, в той час як Objective-C перевершує Python в 2,8 рази. У шифруванні RC4 потужний Objective-C забезпечує продуктивність Python в 127 разів, в той час як Swift забезпечує продуктивність Python в 220 разів.

2.4. Базові фреймворки

Фреймворк - це певний набір правил, ідей або переконань, який використовується для вирішення проблем або для прийняття рішення про те, що робити. Тож проведемо огляд фреймворків, що використовуються у додатку.

2.4.1. UIKit та Foundation

UIKit - це за найбільший фреймворк що відповідає за візуальну частину. Саме він визначає основні компоненти, такі як таблиці, колекції, кнопки, контролери та інші.

Платформа Foundation - це невід'ємна частина набору інструментів розробника iOS. Він надає кореневий клас NSObject і велику кількість основних будівельних блоків для розробки iOS, від класів для чисел і рядків до масивів і словників. Рлатформа Foundation володіє великою потужністю і незамінна для розробки додатків IOS.

Основна мета обох фреймворків подібна, забезпечувати обмін даними та кодом між різними бібліотеками та фреймворками. Core Foundation також включає підтримку інтернаціоналізації. Ключовий компонент цієї підтримки забезпечується через непрозорий тип CFString, який ефективно керує масивом символів Unicode.

2.4.2. MapKit

MapKit - це потужний API, доступний на пристроях iOS, який дозволяє легко відображати карти, позначати місця розташування, доповнювати дані користувача та навіть малювати маршрути або інші фігури зверху.

Apple надає клас MKMapView для роботи з картою (рис. 2.4.). Цей клас відображає карти і надає інтерфейс для навігації по вмісту карти.

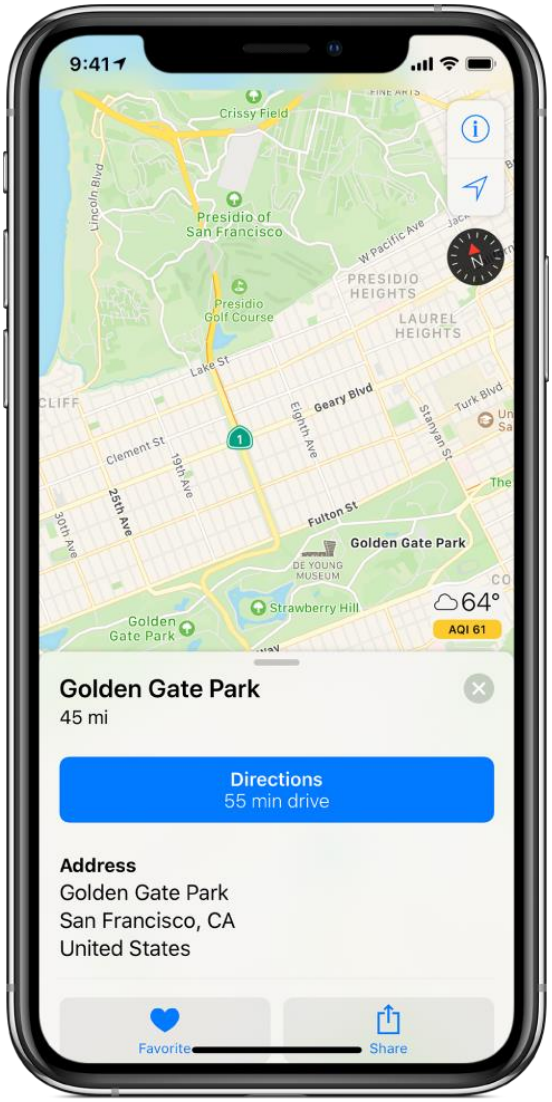

Рис. 2.4. Вигляд MapKit

Цей клас використовується для відображення інформації про карту і для управління вмістом карти з програми. Карту можна центрувати по заданій

координаті, вказати розмір області, яку необхідно відобразити, і забезпечити карту користувальницької інформацією.

При ініціалізації подання карти вказується початковий регіон для відображення, задавши властивість регіону карти. Область визначається центральною точкою і відстанню по горизонталі і вертикалі, так званим проміжком.

Клас MKMapView підтримує можливість анотувати карту спеціальною інформацією. Оскільки карта може містити велику кількість анотацій, у режимах перегляду створюються різні типи об'єктів анотації. Вони використовуються для відображення певної мітки на карті. Анотації в загальному створюються використовуючи вже створені класи (рис. 2.5.). Це дозволяє безпосередньо маніпулювати даними анотацій, але все одно зробити їх доступними для перегляду карти. Кожен об’єкт анотації містить інформацію про розташування анотації на карті разом із описовою інформацією, яка може бути відображена.

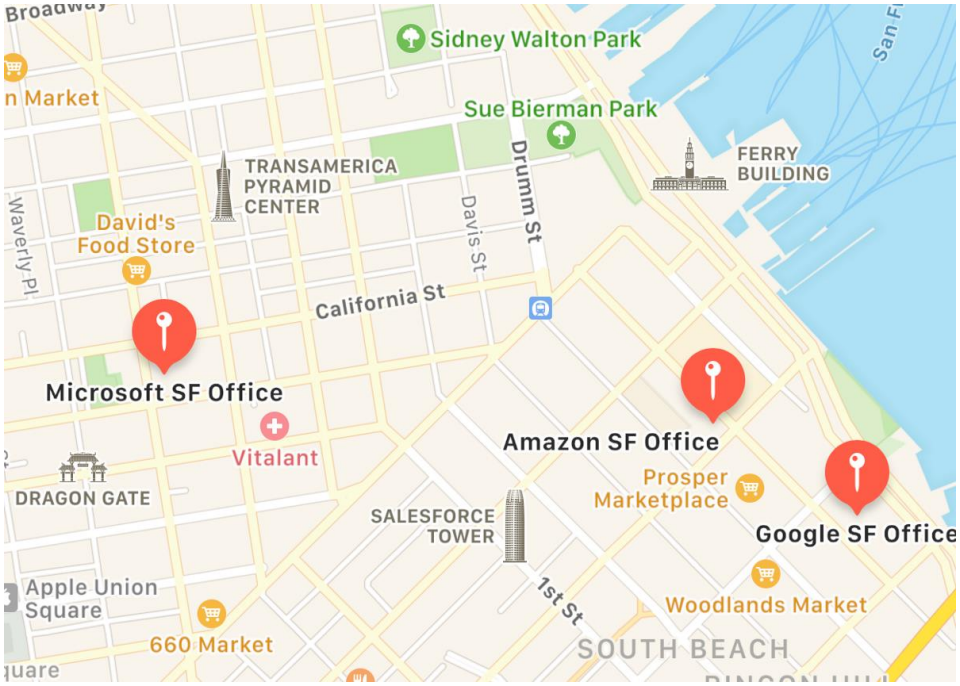

Рис. 2.5. Аннотації

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

2.4.3. Push-повідомлення

Push-повідомлення - це повідомлення, що надсилаються в додаток через службу Push-повідомлень Apple (APNS), навіть якщо програма не запущена або телефон знаходиться в сплячому режимі (рис. 2.6.).

Рис. 2.6. Push-повідомлення

Push-повідомлення можна використовувати для:

- Відображення короткого текстового повідомлення, яке називається попередженням, що привертає увагу до чогось нового у вашому додатку;
- Відтворення звуку повідомлення;
- Встановлення номеру на значку програми (Badge), щоб користувач знав про появу нових елементів;
- Вказати дії, які користувач може зробити, не відкриваючи додаток;
- Показати вкладення мультимедіа;
- Дозволяючи додатку виконувати завдання у фоновому режимі;
- Групувати повідомлення в потоки;
- Редагувати або видаляти доставлені повідомлення;

- Запустити код, щоб змінити своє повідомлення перед його відображенням;
- Відобразити користувальницький інтерактивний користувальницький інтерфейс для повідомлення.

Саме завядки push-повідомленням у додатку можливо буде вчасно відобразити нове повідомлення від інших користувачів.

2.5. Додаткові фреймворки

2.5.1. Cocoapods

Cocoapods — це менеджер залежностей, який використовується для включення сторонніх бібліотек у проекти iOS та Mac. Усі бібліотеки, які потрібно включити у проект, додаються в файл Podfile, а потім просто запускається команда pod на терміналі, щоб встановити додані залежності в podfile (рис. 2.7.).

```
target 'UsersListwithRx' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  def testing_pods

  pod 'RxSwift'
  pod 'RxCocoa'
  pod 'Alamofire'
  pod 'Quick'
  pod 'Nimble'

  end

  # Pods for UsersListwithRx

  target 'UsersListwithRxTests' do
    inherit! :search_paths

    testing_pods

  end

  target 'UsersListwithRxUITests' do

    testing_pods

  end

end
```

Рис. 2.7. Podfile

Cocoapods зосереджений на розповсюдженні коду сторонніх розробників на основі вихідних кодів і дозволяє автоматично інтегрувати у проекти Xcode

2.5.2. Moya

Незалежно від того, чи отримує програма дані програми з сервера, оновлює статус в соціальних мережах або завантажує віддалені файли на диск, мережеві запити допомагають їй працювати. Щоб допомогти з багатьма вимогами до мережевих запитів, Apple надає URLSession, повний мережевий API для завантаження та завантаження вмісту.

Moya - це мережева бібліотека, натхненна концепцією інкапсуляції мережевих запитів безпечним способом, як правило, з використанням перерахувань, що забезпечує впевненість під час роботи з мережевим рівнем. Moya фактично використовує фреймворк Alamofire, надаючи інший підхід до структурування мережевого рівня

Alamofire - це елегантний і компонований спосіб інтерфейсу із мережевими запитами HTTP. Він побудований на основі системи завантаження URL-адрес Apple, наданої платформою Foundation. Ядром системи є URLSession і підкласи URLSessionTask. Alamo fire обгортає ці API та багато інших у простіший у використанні інтерфейс та надає різноманітні функціональні можливості, необхідні для розробки сучасних додатків із використанням мережі HTTP.

Проаналізуємо типову імплементацію. Спочатку створюється Enum (рис. 2.8.) з запитами. Enum - це тип даних, що складається з набору іменованих значень, які називаються членами.

```
enum WeekDay :String {
    case Monday
    case Tuesday

    func day() ->String {
        return self.rawValue
    }

}
```

Рис. 2.8. Приклад Enum

Наступним кроком слід розширити перелік API за допомогою TargetType Moya і визначити необхідні змінні, а саме:

- baseURL: загальний префікс URL-адреси API.
- path: шлях після baseURL. Іноді нам потрібно вводити динамічне значення. Наприклад, щоб отримати відомості про фільм, ми повинні ввести ідентифікатор фільму в шлях, наприклад movie/{movieId}.
- method: метод запиту HTTP. Це можуть бути get, post, put, delete, connect, head, options, patch та trace
- sampleData: можна створити заглушки даних для відповіді аби протестувати.
- task: представляє завдання HTTP. Слід використовувати requestPlain, якщо не додається жодного параметра чи тіла запиту. Щоб додати параметри запиту, наприклад: ?api_key=xxx. слід вибрати параметри requestParameters. Крім того, можна використовувати requestJSONEncodable для тіла з кодованою моделлю запиту.
- header: заголовки, які будуть використані в запиті.

2.5.3. Swift Messages

SwiftMessages — це дуже гнучка бібліотека презентацій контролера перегляду та перегляду для iOS.

Перегляди повідомлень і контролери перегляду можуть відображатися у верхній, нижній або центральній частині екрана, або за панелями навігації та панелями вкладок. Існують інтерактивні жести звільнення, зокрема веселі, засновані на фізики. Кілька режимів затемнення фону та багато іншого.

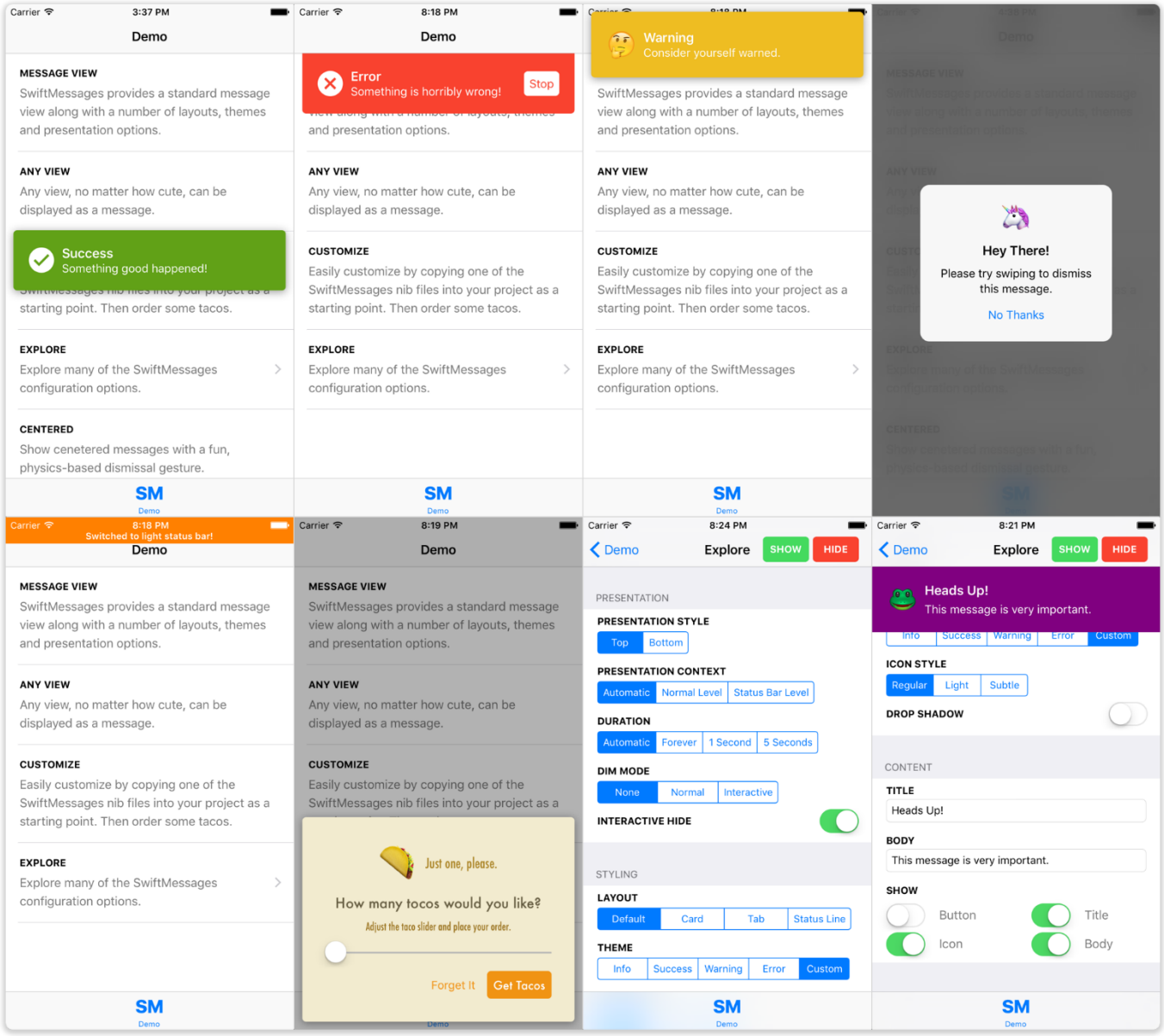

Рис. 2.9. Вигляд SwiftMessages

На додаток до численних параметрів конфігурації, SwiftMessages надає кілька гарних макетів і тем (рис. 2.9.). Але SwiftMessages також зручний для розробника, що означає, що його можна повністю та легко налаштувати на власний смак.

Основним використанням цієї бібліотеки у додатку буде відображення системних та вхідних повідомлень. Яскравим прикладом системного повідомлення буде показати статус REST запиту доступною для користувача мовою. Як от «Зустріч було успішно створено!» - після успішної відповіді з серверу, або ж у разі негативної – «Упс, не вдалося створити зустріч :(Спробуйте пізніше».

У випадку з повідомленнями від інших користувачів, то після обробки вхідних push-повідомлень, уся інформація буде відображена використовуючи саме цю бібліотеку у стилі додатка.

2.5.4. Kingfisher

Kingfisher — це бібліотека реалізована для Swift, що завантажує та кешує зображення з інтернету. Цей проект значною мірою натхненний іншою популярною бібліотекою популярним SDWebImage. Перевагою бібліотеки Kingfisher є саме чисте використовування бібліотеки на мові Swift.

Проведемо аналіз особливих можливостей бібліотеки. Все в Kingfisher йде асинхронно, не тільки завантаження, але і кешування. Це означає, що ніколи не потрібно турбуватися про блокування потоку інтерфейсу користувача. Він також надає прості у використанні API.

Бібліотека надає багатошаровий кеш. Кеш - це зарезервована область пам'яті до якої завжди є швидкий доступ. Завантажені зображення будуть кешуватися як в пам’яті, так і на диску. Тому їх не потрібно завантажувати знову, і це може значно як підвищити рівень додатка, так і полегшити його створення, адже не треба витрачати час на розробку локального сервісу, що буде займатися розподілом та зберіганням картинок.

Має удосконалене управління кешем. Можна встановити максимальну тривалість або розмір кешу. А кеш-пам’ять буде очищено автоматично, щоб запобігти захопленню занадто великого ресурсу. За замовчуванням а дані зберігаються близько тижня у відповідному типі пам'яті.

Kingfisher є сучасним фреймворком та використовує NSURLSession та новітню технологію GCD, що робить його міцною та швидкою структурою. NSURLSession — це API, наданий Apple для розробників для взаємодії з протоколами HTTP і HTTPS. NSURLSession надає можливість створювати сеанси для програми для виконання таких завдань, як передача даних. Grand Central Dispatch (GCD) — це низькорівневий API для керування одночасними операціями.

Є можливість скасування завдання обробки. Можна скасувати процес завантаження або отримання зображення, якщо він більше не потрібен. Це є дуже важливим моментом для зберігання правильної роботи додатка та використання усіх типів його пам'яті.

Завядяки низькому рівні зв'язанності Kingfisher складається з незалежних компонентів. Тому є можливість використовувати завантажувач або систему кешування окремо. Або навіть створити власний кеш на основі коду Kingfisher.
Бібліотека має влсані параметри розпакування зображення у фоновому режимі перед його відтворенням, що може покращити продуктивність інтерфейсу користувача.
Зображення можна вспавити у UIImageView безпосередньо з URL-адреси. UIImageView це представлення зображення у мові Swift.

Ця бібліотека буде використана у додатку для завантаження таких зображень як профілі, картинки місць зустрічі тощо.

2.5.5. Firebase Cloud Messaging

Щоб отримати push-сповіщення, пристрій має бути зареєстрований у службі Apple Push Notification Service (APN), отримавши унікальний маркер пристрою. Після реєстрації пристрою можна надсилати на нього push-сповіщення, надіславши запит до APN за допомогою маркера пристрою. Усе це спілкування має відбуватися з якогось веб-сервера.

Одним з способів реалізувати власний веб-сервіс для зв’язку з APN є Firebase Cloud Messaging (FCM). З Firebase Cloud Messaging (FCM) під рукою є проста у використанні система. FCM обробляє хмарний аспект push-повідомлень, дозволяючи надсилати та отримувати push-повідомлення без створення власного веб-сервісу.

2.5.6. Lokalise

Lokalise-це платформа для керування локалізацією та перекладом. Це один з найкращих способів адаптації веб і мобільних додатків, ігор, Інтернету речей, програмного забезпечення або цифрового контенту для міжнародних ринків.

Lokalise являє собою систему управління перекладом (TMS - translation management system), яка допомагає командам автоматизувати, управляти і перекладати контент (текстові рядки, документи) (рис. 2.10.).

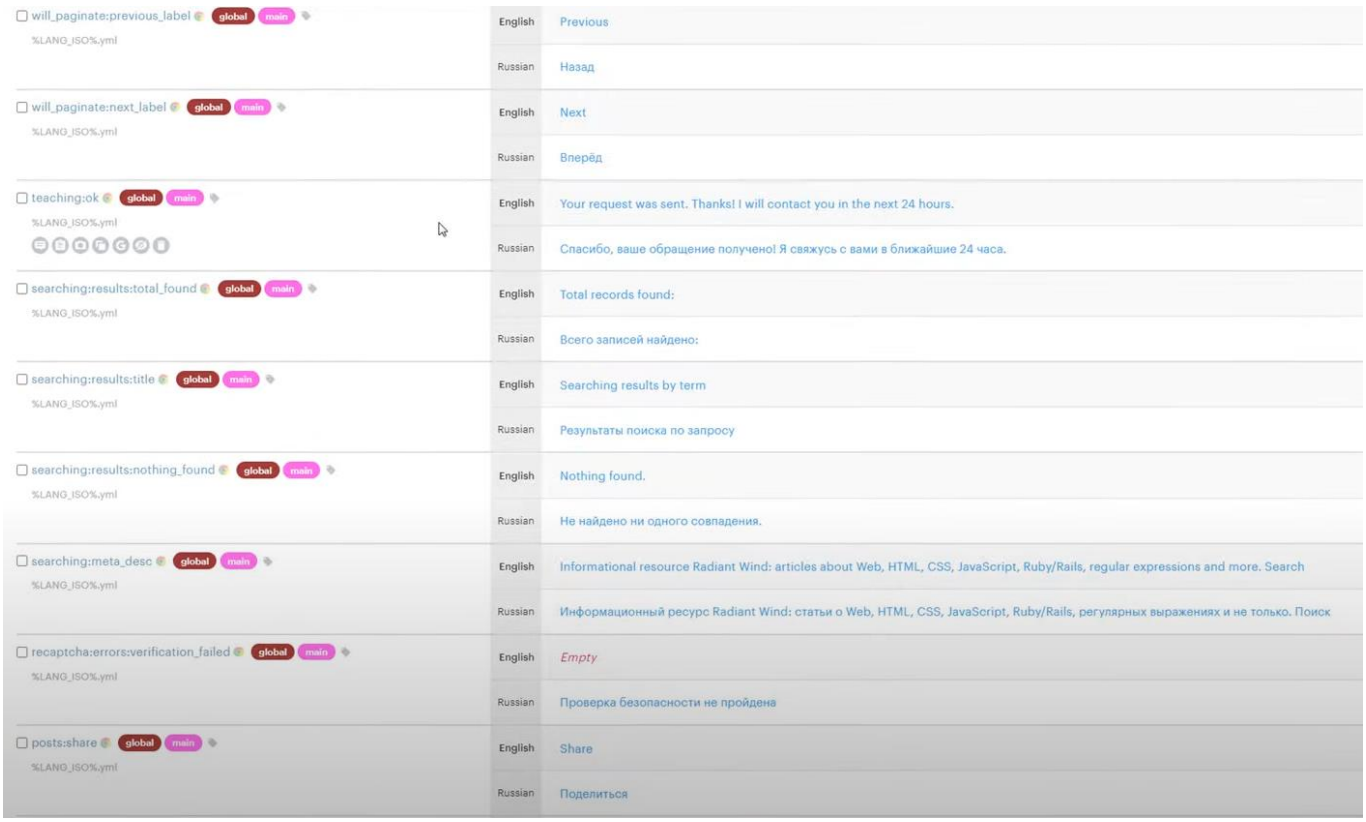

Рис. 2.10. Вигляд проекту

За допомогою Lokalise можливо:

- Перевести файли локалізації;
- Співпрацювати та керувати всіма проектами з локалізації програмного забезпечення на одній платформі;
- Впровадити гнучкий робочий процес локалізації;
- Додавати скріншоти для автоматичного розпізнавання і зіставлення з текстовими рядками в проектах;
- Налаштовувати автоматизовані робочі процеси за допомогою API, використовувати веб-гачки або інтегруватися з іншими сервісами такими як Figma;

- Попередній перегляд в режимі реального часу того, як переклади будуть виглядати у веб або мобільному додатку;
- Замовити професійні переклади у перекладачів Lokalise або використовувати машинний переклад.

Для кожного члена розробки він буде корисним:

- Розробникам, які хочуть скоротити рутинні ручні операції за рахунок автоматизації процесу перекладу і локалізації.
- Менеджерам проектів і локалізації, які хочуть прискорити процес локалізації і більш ефективно управляти всіма проектами і командами в одному місці.
- Перекладачам, які хочуть забезпечити високоякісні переклади, використовуючи скріншоти, коментарі, машинний переклад, контекстні редактори та інші інструменти.
- Дизайнерам, які хочуть використовувати двосторонню інтеграцію з Sketch, Figma або Adobe XD, або, в якості альтернативи, налаштовують робочі процеси створення скріншотів.

2.5.7. Chatto

Chatto — це легкий фреймворк Swift для створення додатків чату. Він був розроблений для розширення та продуктивності. Поряд із Chatto є ChattoAdditions, супутня структура, яка включає клітинки для повідомлень і розширюваний компонент введення.

Chatto — це фреймворк Swift для полегшення розробки додатків для чату. На рівні інтерфейсу користувача він керує UICollectionView, де відображаються повідомлення, і надає заповнювач для компонента введення. UICollectionView - це об’єкт, який керує впорядкованою колекцією елементів даних і представляє їх за допомогою настроюваних макетів (рис. 2.11.).

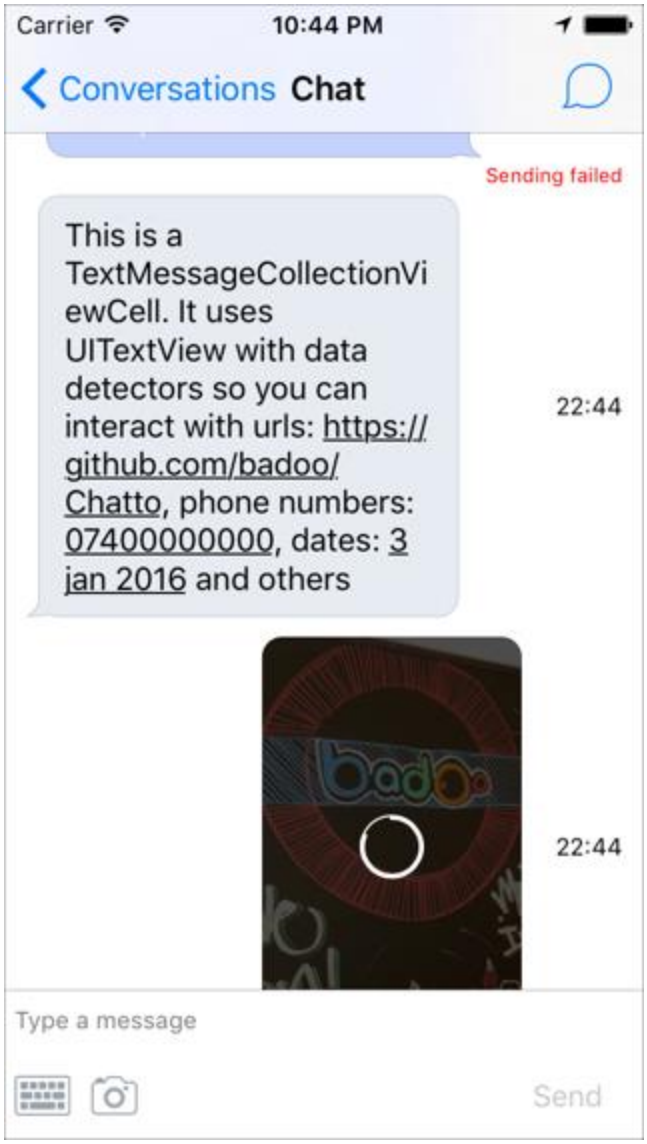

Рис. 2.11. Вигляд чату

Функції Chatto:

- Керує UICollectionView, дбаючи про обчислення змін та виконання оновлень
- Заповнювач для компонента введення в нижній частині екрана;
- Забезпечує підтримку інтерактивного закриття клавіатури та автоматично налаштовує вставки, коли вона з'являється;
- Надає макет за замовчуванням для UICollectionView;
- Обчислює цей макет у фоновій черзі.

- Активує сповіщення до джерела даних, щоб отримати більше повідомлень (розбивка на сторінки);
- Заохочує чистий код, роз'єднуючи презентацію вашого повідомлення на різних доповідачів.

Функції ChattoAdditions:

- BaseMessageCollectionViewCell: базова комірка для повідомлень чату, яка підтримує піктограму, коли повідомлення не відіслалося, аватар і мітку часу. Це загальний контейнер для фактичного повідомлення;
- TextMessageCollectionViewCell: конкретний підклас для текстових повідомлень, що підтримує виявлення посилань;
- PhotoMessageCollectionViewCell: конкретний підклас для фотоповідомлень, що підтримує передачу індикатора прогресу;
- ChatInputBar: розширювана панель введення для надсилання тексту, фотографій та користувацьких типів повідомлень.

2.5.8. Swiftgen

SwiftGen – це інструмент для автоматичного генерування Swift-коду для ресурсів проектів (наприклад, зображень, локалізованих рядків тощо), щоб зробити їх безпечними для типу.

Використання цього має кілька переваг:

- Уникання будь-якого ризику помилки при використанні рядка;
- Автозаповнення;
- Уникання ризику використання неіснуючої назви активу;
- Все це буде забезпечено компілятором і таким чином легко уникнути ризику збою під час виконання.

Крім того, його можна повністю налаштувати завдяки шаблонам Stencil, тож навіть якщо він поставляється із попередньо визначеними шаблонами, можна створити свій власний, щоб створити будь-який код, який відповідає відповідним потребам та рекомендаціям!

SwiftGen надається як єдиний інструмент командного рядка, який використовує файл конфігурації для визначення різних синтаксичних аналізаторів для запуску (залежно від типу вхідних файлів, які потрібно проаналізувати) та їх параметрів.

Кожен синтаксичний аналізатор, описаний у файлі конфігурації (рядки, шрифти), зазвичай відповідає типу вхідних ресурсів для аналізу (файли рядків, файли IB, файли шрифтів, файли JSON, тощо), що дозволяє генерувати константи для кожного типа цих вхідних файлів (рис. 2.12.).

Щоб використовувати SwiftGen, необхідно просто створити файл YAML swiftgen.yml, а потім відредагувати його, щоб адаптувати його до проекта. У файлі конфігурації має бути перераховано всі парсери, які потрібно викликати, і для кожного синтаксичного аналізатора список входів/виходів/шаблони/параметрів для його використання.

```yaml
strings:
  inputs: Resources/Base.lproj
  outputs:
    - templateName: structured-swift5
      output: Generated/Strings.swift
xcassets:
  inputs:
    - Resources/Images.xcassets
    - Resources/MoreImages.xcassets
    - Resources/Colors.xcassets
  outputs:
    - templateName: swift5
      output: Generated/Assets.swift
```

Рис. 2.12. Приклад SwiftGen

2.5.9. SwiftLint

SwiftLint — чудовий інструмент для дотримання стилю Swift. Працюючи з різними розробниками з різним рівнем досвіду та різними уподобаннями, важко зберегти послідовність у тому, як написаний код. Крім того розробники досить ліниві аби перевіряти наявність проблем із лінтингом щоразу, коли робиться commit, тому краще дозволити виконати роботу інструменту та CI. SwiftLint – це інструмент, який

допомагає команді налаштувати деякі правила стилю на основі своїх потреб і переваг, а пізніше, використовуючи CI, застосовувати ці правила для всієї команди розробників.

Після встановлення SwiftLint можна інтегрувати в Xcode (або інші редактори, такі як AppCode), щоб ви могли отримувати попередження та помилки під час створення проекту та відображати їх у IDE (рис. 2.13.).

SwiftLint пропонує безліч правил, і потрібно використовувати файл swiftlint.yml для їх налаштування. Застосування довжини рядка, файлу або тіла функції та відображення деяких попереджень під час використання примусового приведення — це одні з найпростіших, але дійсно корисних правил, з яких ви можете почати. Інші правила можуть полягати в застосуванні приватних IBActions та IBOutlets або видавати помилку під час використання застарілих конструкторів.

```
force_cast: warning
force_try: warning

line_length:
  warning: 120
  error: 500

file_length:
  warning: 600
  error: 1000

function_body_length:
  warning: 40
  error: 120

opt_in_rules:
  - empty_count
  - force_unwrapping
  - legacy_constant
  - legacy_constructor
  - private_action
  - private_outlet
```

Рис. 2.13. Приклад swiftlint.yml

Незважаючи на те, що SwiftLine пропонує велику кількість правил і гнучкість для налаштувань, іноді не можна уникнути не дотримування правила. SwiftLint також підготувлений для такого сценарію. Можна використовувати коментар у файлі коду, щоб вимкнути перевірку певного правила для певного рядка.

2.5.10. Realm

Realm — це мобільна база даних, яка працює безпосередньо в телефонах, планшетах або носимих пристроях.

Realm є альтернативою Apple Core Data. База даних Realm працює швидше, ніж Core Data і SQLite, і з нею простіше працювати. Вона безкоштовна для необмеженого використання, якщо не потрібно використовувати хмарні функції Realm.

Переаги Realm:

- Швидко виконує операції;
- Не займає багато пам'яті, оскільки створений для мобільних пристроїв;
- Модель даних є об'єктно-орієнтованою;
- Дозволяє писати менше коду ніж нативна Apple Core Data;
- локальна база даних Realm зберігає дані на диску, тому додаток може викноувати роботу як в онлайн так і офлайн режимі;
- Ефективно використовує пам'ять та дисковий простір.

Проведемо аналіз основних елементів бази даних. Realm або екземпляри Realm - це серце фреймворку, це точка доступу до основної бази даних. Екземпляри будуть створюватися за допомого ініціалізатора Realm.

Object (Об'єкт) - це модель Realm. Процес створення моделі визначає схему бази даних. Для того, щоб створити модель, необхідно просто створити підклас Object і визначати поля, які потрібно зберегти як властивості (рис. 2.14.).

```
class MessageObject: Object {
    @objc dynamic var id = 0
    @objc dynamic var message = ""
    @objc dynamic var recipientName = ""
    @objc dynamic var senderName = ""
    @objc dynamic var isRead = false
    @objc dynamic var isReceived = false
    @objc dynamic var status = false
    @objc dynamic var timestamp = 0.0
    @objc dynamic var isIncoming = false
}
```

Рис. 2.14. Приклад Object

Write Transactions (Запис операцій) - будь-які операції в базі даних повинні виконуватися в рамках операцій write, які відбуваються за допомогою виклику write(_:) у екземлярів Realm. Також необхідно обробити помилку, якщо така буде за допомогою блоку do catch (рис. 2.15.).

```
func addMessage(messageObject: MessageObject) {
    do {
        try realm.write({
            realm.add(messageObject)
        })
    } catch {
        print("Failed to add message in DataSource")
    }
}
```

Рис. 2.15. Приклад Write Transaction

Queries (Запити) - саме за допомогою запитів із бази даних можно вибрати потірбні об'єекти. Найпростіша форма запиту - це виклик objects() у екземпляра Realm, передаючи клас Object, який шукається. Якщо пошукові запити складніші, можна використовувати предикати, вибудовуючи запити, а також організуючи результати пошуку. Предикати можуть дозволити фільтрувати дані в пам’яті

гнучкими та неймовірно потужними способами, не вимагаючи від підтримувати зростаючий набір високоспецифічних API.

Results (Результати) - це контейнер даних, що отримується на запити. Вони мають багато спільного зі звичайними масивами, зокрема синтаксис сабскрипта для захоплення елемента індексі (рис. 2.16.).

```
func getAllMessages() -> [MessageObject] {
    let realm = initRealm()
    let results = realm.objects(MessageObject.self)
    return Array(results)
}
```

Рис. 2.16. Приклад отримання Results

2.5.11. StompClientLib

Бібліотека Stomp Client - це клієнт stomp в Swift. Він використовує SocketRocket Facebook як залежність від websocket. SocketRocket написаний на Objective-C, але частина StompClientLib написана на Swift, і її використання є швидким. Ви можете використовувати цю бібліотеку у своїх проектах Swift 5+, 4+ і 3+.

STOMP-це простий текстовий протокол обміну повідомленнями. Він визначає сумісний дротовий формат, щоб будь-який з доступних клієнтів STOMP міг взаємодіяти з будь-яким брокером повідомлень STOMP для забезпечення простої і широкої сумісності обміну повідомленнями між мовами і платформами (на веб-сайті STOMP є список реалізацій клієнта і сервера STOMP.

Брокер повідомлень-це проміжний програмний модуль, який переводить повідомлення з формального протоколу обміну повідомленнями відправника в формальний протокол обміну повідомленнями одержувача.

API WebSocket дозволяє веб-додаткам обробляти двонаправлений зв'язок з серверним процесом простим способом. Розробники використовували XMLHttpRequest ("XHR") для таких цілей, але XHR робить розробку веб-додатків, які обмінюються даними з сервером і назад, надмірно складною. XHR - це в основному

асинхронний HTTP, і оскільки потрібно використовувати складну техніку, таку як довго зависаючий GET, для відправки даних з сервера в браузер, прості завдання швидко ускладнюються. Протокол передачі гіпертексту (HTTP) - це прикладний протокол для розподілених, спільних гіпермедійних інформаційних систем, який дозволяє користувачам обмінюватися даними у всесвітній павутині (рис. 2.17.).

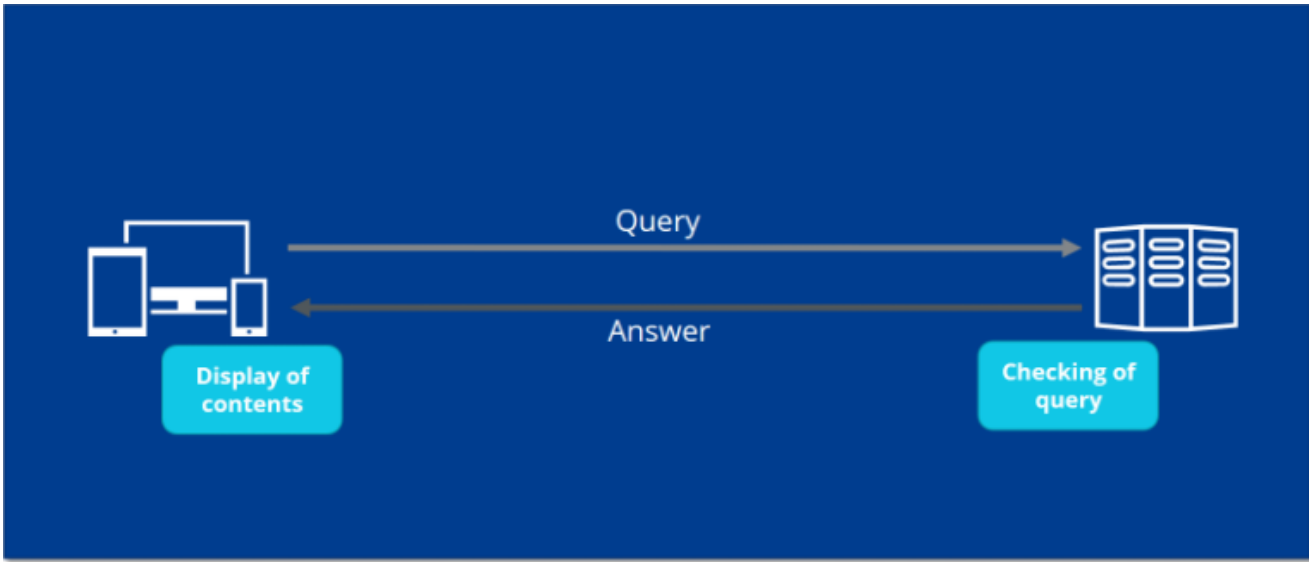

Рис. 2.17. Спрощена демонстрація роботи HTTP

На відміну від XMLHttpRequest, WebSockets надають реальний двонаправлений канал зв'язку у браузері. Як тільки буде отримано підключення до WebSocket, можна відправляти дані з браузера на сервер і отримувати дані з сервера в браузер за допомогою обробника подій.

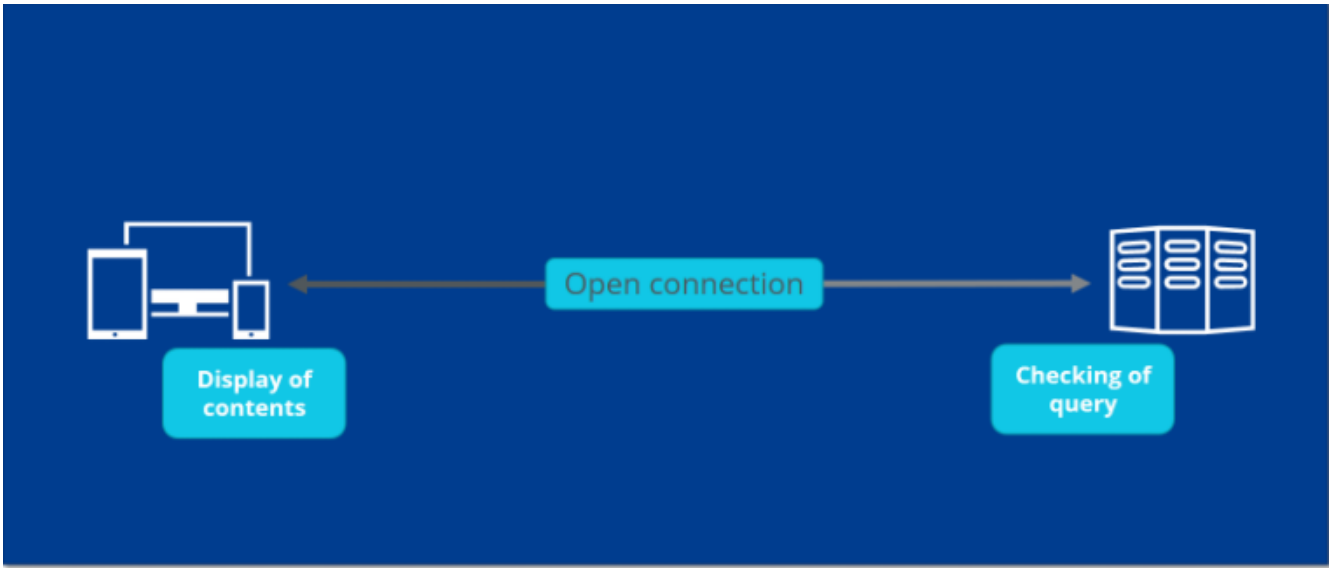

Рис. 2.18. Демонстрація роботи WebSocket

З'єднання за допомогою WebSocket починається з "рукостискання". Спочатку клієнт відправляє HTTP-запит, що містить заголовок "оновлення". У разі якщо сервер підтримує даний протокол, то він приймає протокол та потім відправляє власний заголовок. Якщо з'єднання установлене то «рукостискання» можна вважати успішним. З'єднання буде використовувати протокол WebSocket (рис. 2.18.).

Веб-сайти надсилають дані за допомогою системи обміну повідомленнями на основі фреймів, яка допомагає зменшити обсяг переданих даних, які не є корисним навантаженням. Дані передаються у вигляді повідомлень, що складаються з одного або декількох кадрів, що містять корисне навантаження. Кожен кадр передується 4-12 байтами даних про корисне навантаження, щоб його можна було правильно відновити. Це корисно, оскільки сервер або клієнт можуть передавати стільки даних, скільки їм подобається, без усіх накладних витрат на заголовок, які виникають при традиційних http-запитах.

2.6. Контроль версій

Системи контролю версій - це категорія програмних засобів, які допомагають записувати зміни, внесені у файли, відстежуючи зміни, внесені в код.

Програмні продукти розробляється у групою розробників,отже вони можуть розроблятися в різних місцях, і кожен з них вносить свій внесок у певний вид функціональності. Тому, щоб внести свій внесок у продукт, вони повинні внести зміни до коду. Система контролю версій - це свого роду програмне забезпечення, яке допомагає команді розробників ефективно обмінюватися інформацією та керувати (відстежувати) всіма змінами, які були внесені до вихідного коду, а також інформацією про те, хто вніс і які зміни були внесені. Кожен розробник працює у своїй окремій гілці, отже зміни не будуть зливатися в основній гілці, і як тільки зміни будуть одобрені, вони будуть залиті з основну гілку. Це не тільки упорядковує вихідний код, але і підвищує продуктивність, роблячи процес розробки плавним.

Переваги системи контролю версій:

- Підвищує швидкість розробки проекту за рахунок забезпечення ефективної співпраці;
- Зменшує ймовірність помилок і конфліктів при розробці проекту за рахунок відстеження кожної невеликої зміни;
- Допомагає у відновленні у випадку будь-якої катастрофи або непередбаченої ситуації;
- Підвищує продуктивність, прискорює доставку продукції і підвищує кваліфікацію співробітників за рахунок поліпшення комунікації і допомоги;
- Співробітники можуть вносити свій внесок з будь-якого місця, незалежно від різних географічних місць, через цю систему контроля версій;
- Для кожного окремого учасника проекту підтримується окрема робоча копія;
- Інформує нас про те, хто, що, коли, чому були внесені зміни.

У якості системи контролю версій було обрано Sourcetree. Sourcetree - це настільний клієнт з графічним інтерфейсом користувача, який спрощує взаємодію з репозиторіями Git (рис. 2.19.).

Він має інтуїтивно зрозумілий графічний інтерфейс для репозиторіїв, що усуває розрив між користувачем і Git. Програмні розробники отримують велику вигоду від використання Sourcetree, оскільки це дозволяє їм зосередитися на написанні коду і бути більш продуктивними.

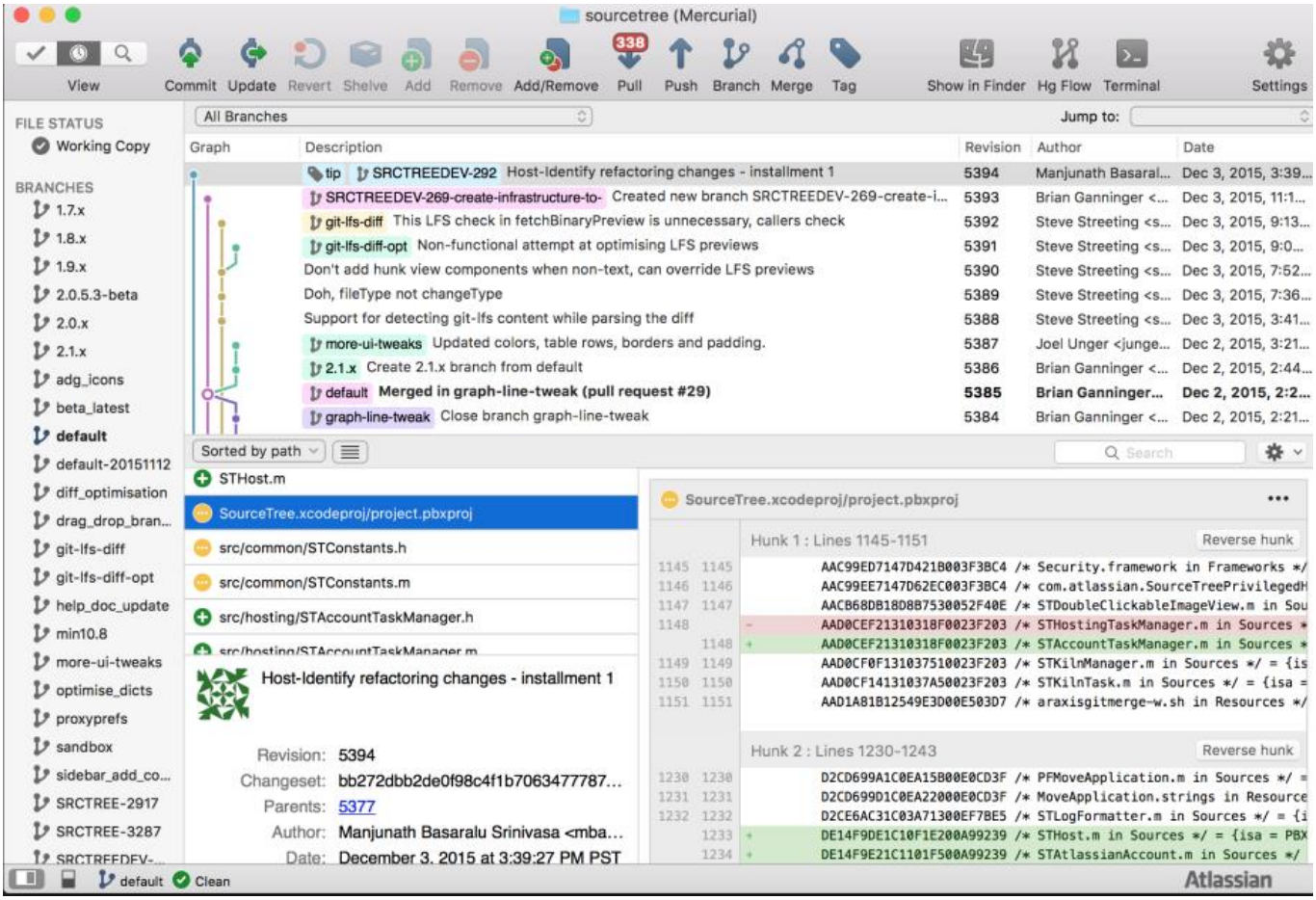

Рис 2.19. Вигляд Sourcetree

Висновок до другого розділу

Було розглянуто середовище розробки Xcode, корисні інструменти, що були використані у ході розробки. Також було розглянуто сучасну мову розробки Swift, завдяки якій розробляються додатки для iOS/MacOS систем.

Детально проаналізовано базові, тобто ті що доступні одразу в середовищі розробки Xcode, та додаткові фреймворки, які були додані до проекту за допомогою Cocoapods. Одними із найпопулярніших фреймворків серед розробників є – Moya та Kingfisher. Кожний фреймворк значно полегшує та пришвидшує розробку мобільного додатка.

Контроль доступу у свою чергу дозволяє з легкістю розробляти додаток у команді та надає можливість повернутися до будь-якого стану розробки додатку у потрібний час.

РОЗДІЛ 3.
ДЕМОНСТРАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Структура

На рис. 4.1. можна побачити усю Navigation Area проекту. Усі файли докупи складають xcodeproj. Файли такого типу використовуються для збереження файлів проекту Xcode та для збереження результатів певного етапу розробки.

В Resources зберігаються зображення та шаблони кольорів. За допомогою SwiftGen бібліотеки їх легше викуористовувати у коді.

AppDelegate це клас, що відповідає за обробку певних стані додатку. Наприклад коли додаток буде звернутий відправляти на сервер, що користувач вийшлов з онлайн режиму.

У Components зберігаються користувацькі елементи що відображаються однаково у всьому додатку. Яскравим прикладом може слугувати поле вводу.

Extension це окрема папка з розширенням функціоналу певних класів. Прикладом є розширення функціоналу UILabel для підкреслення тексту.

Services складається з файлі із різними сервісами. Сервіс це клас, який відповідає за певну ділянку функціональності програми. NetworkService відповідає за роботу із REST запитами та чат, DataSource за роботу з БД і так далі.

В UI розподілені класи та структури що відносяться до відповідних екранів згідно з архітектурою.

Launch Screen відповідає за відображення екрану завантаження на самому початку запуску додатка.

Info Plist містить список ключів для конфігурації проекту.

PodFile визначає які додаткові врейморвки є у проекті.

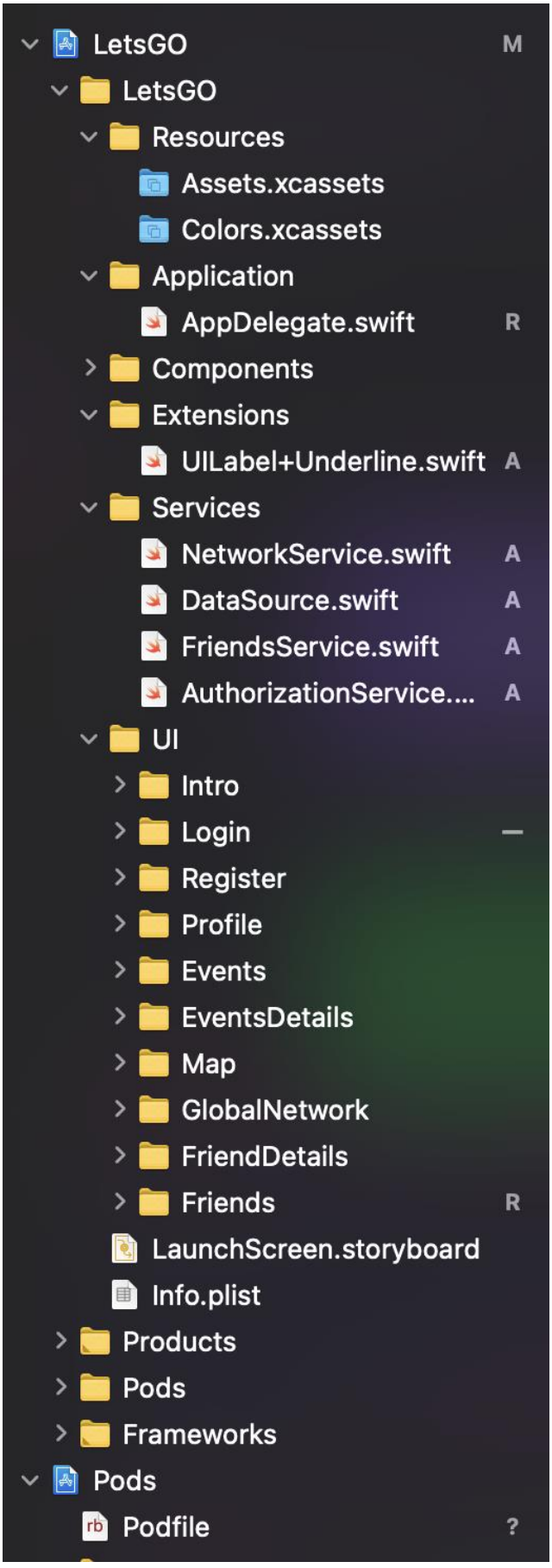

Рис 4.1. Navigation Area

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

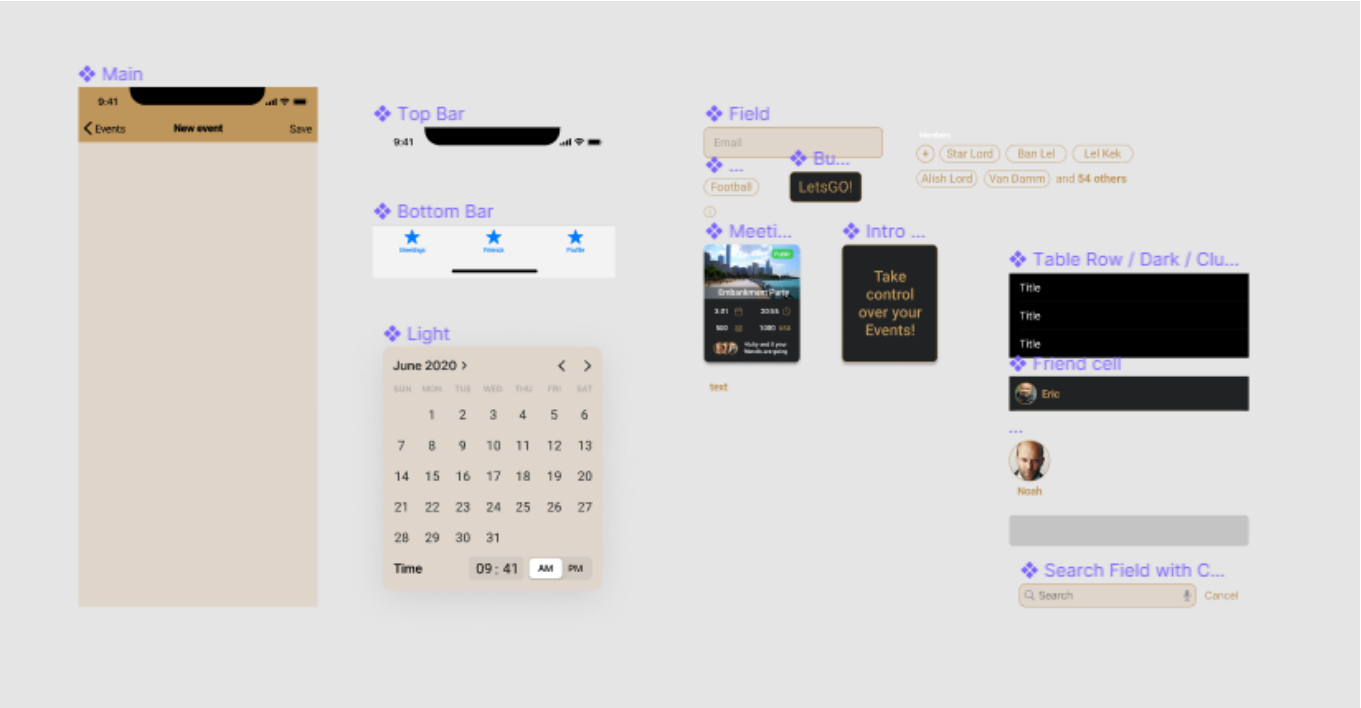

Рис 4.2. Посібник зи стилю

3.2. Огляд функціоналу додатка

Кожний екран у додатку відображає власний View Controller. За перехід між екранам відповідає Navigation Controller. Navigation Controller -це контейнер VC, який керує одним або декількома дочірніми VC в інтерфейсі навігації. В інтерфейсі цього типу одночасно відображається тільки один дочірній VC. Вибір елемента в контролері виду виводить новий контролер виду на екран за допомогою анімації, тим самим приховуючи попередній контролер виду.

Tab Bar Controller відображає вкладки в нижній частині вікна для вибору між різними режимами і для відображення уявлень для цього режиму

При відкритті додатка перевіряється булеве значення «isFirstEnter» в User Defaults. User Defaults - це plist (property list) файл у кореневій папці додатку, що використовується для зберігання простих данних. Структура plist файла схожа до

Dictionary (колекції ключ-значення). Якщо значення «isFirstEnter» false, то відображається екран вступу (рис. 4.3.).

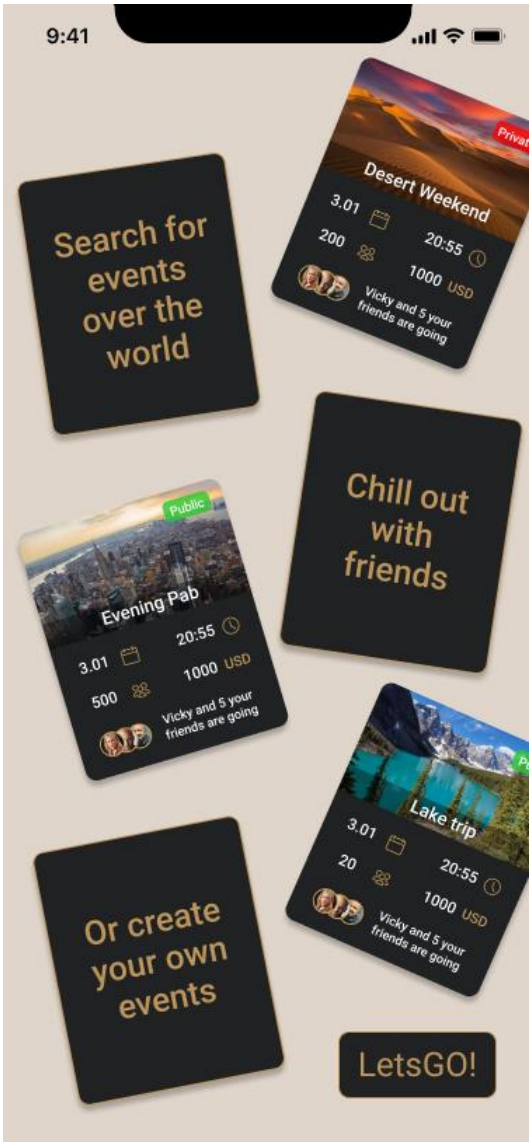

Рис 4.3. Екран вступу

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

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

Далі користувач може натиснути на кнопку «LetsGO», що знаходиться у правому нижному кутку. Вона реалізована за допомогою UIButton, елементу управління, який виконує заданий код у відповідь на дії користувача. Після натискання, Navigation Controller виконує перехід на наступний екран – Login (рис .4.4.).

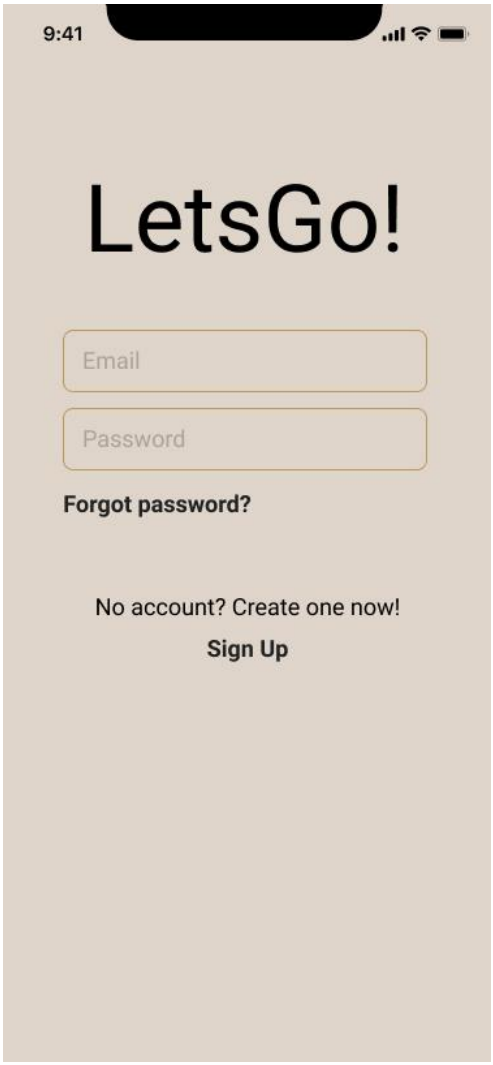

Рис 4.4. Логін

Поля для вводу на даному екрані - це елементи що були створені із стандартних ементів UIKit для перевикористання. Таким чином усі поля вводу у додатку будуть виглядати однаково та не будуть займати багато коду для створення та використання. Будь які зміни у кольорі або функціоналі можна легко внести змінивши значення у масиві налаштувань. А у випадку, коли до проекту приєднається новий розробник, то

йому потрібно буде ознайомитись із елементом в одному місці аби з легкістю розуміти та використовувати його в інших місцях.

Поля для вводу складаються з UIView та UILabel. Певні налаштування виду придають однакове відображення.

Функціонал програми покриває ситуацію коли користувач забуває пароль, тож натиснувши на кнопку «Forgot password», він отримає докладну інструкцію з вказівками на пошту.

Якщо ж користувач не має власного акаунту то може зареєструватися натиснувши на кнопку SignUp. Navigation Controller покаже екран реєстрації (рис. 4.5.).

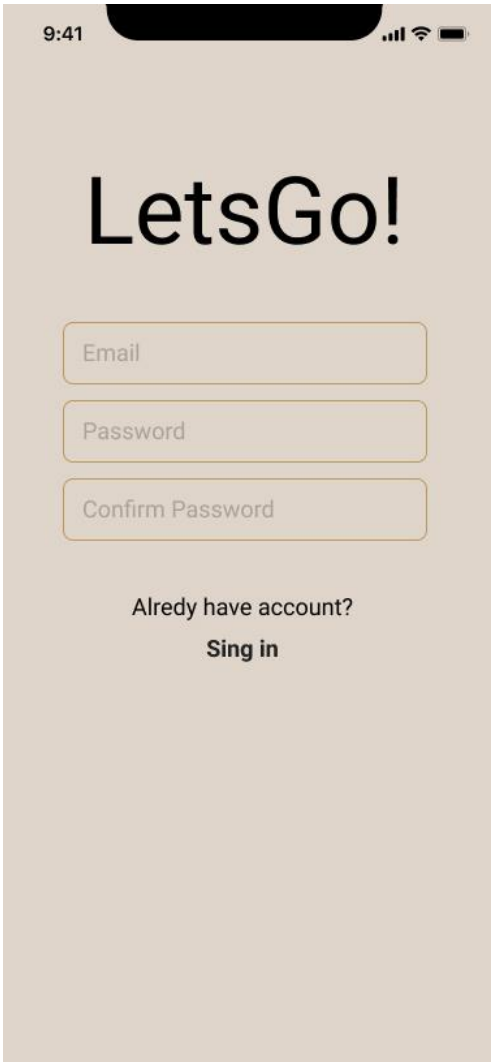

Рис. 4.5. Екран реєстрації

Після введення даних Navigation Controller знову покаже екран входу з попередньо заповненою електронною адресою. Користувач повинен підтвердити свою електронну адресу після чого зможе ввійти у свій акаунт.

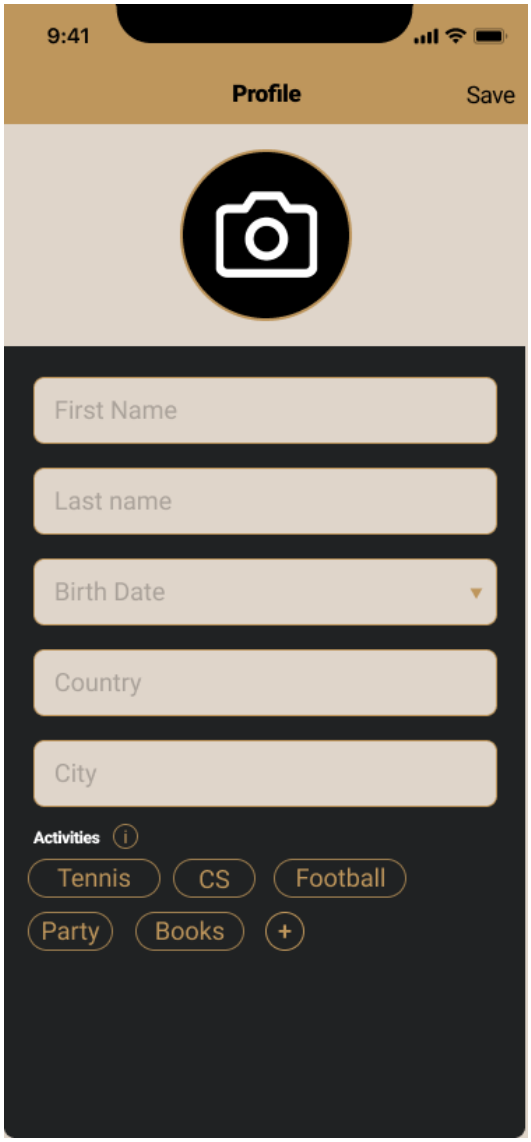

Рис. 4.6. Екран заповнення профілю

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

Аби вибрати фотографію профілю потрібно натиснути на кнопку із іконкою фотокамери, за чим в свою чергу слідує відкриття UIImagePickerController (рис. 4.7.), VC що керує системними інтерфейсами для фотографування, запису відео та вибору елементів із медіа-бібліотеки користувача.

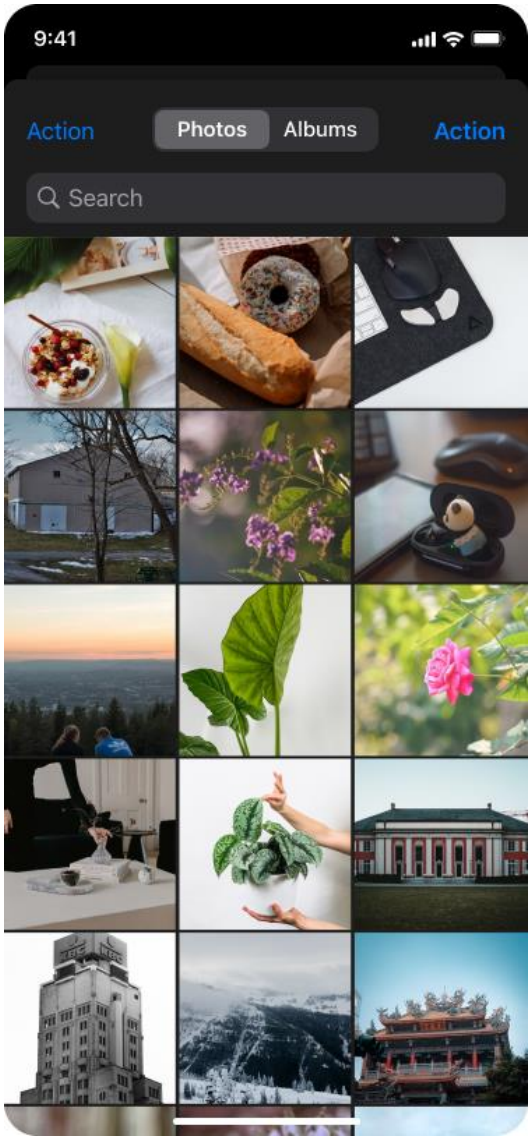

Рис. 4.7. Вигляд UIImagePickerController

На екрані заповнення профілю є функція додавання так званих «Activities». Завдяки ним буде значно легше підібрати подію, що буде співпадати із запитами користувача. Натиснувши на кнопку «Info» з’явиться сповіщення (рис. 4.8.) у якому буде описано для чого дана функція.

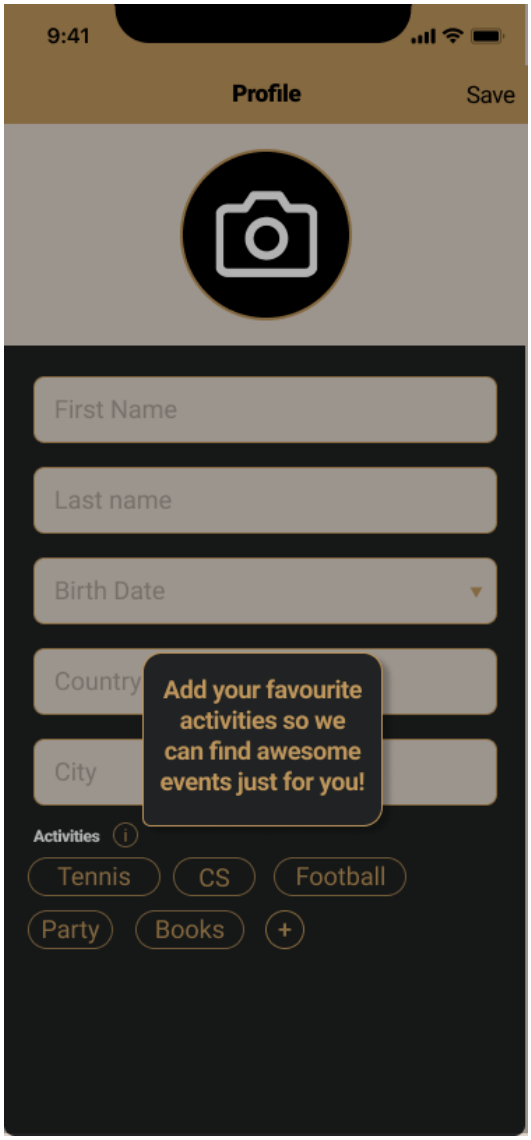

Рис 4.8. Приклад сповіщення

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.

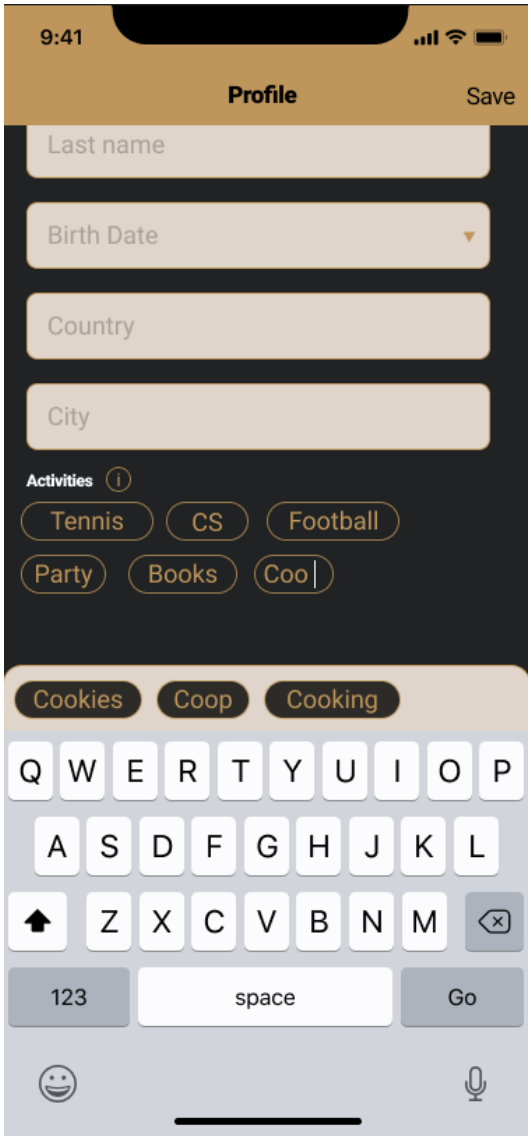

Рис. 4.9. Додавання видів діяльності

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

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 зправа.

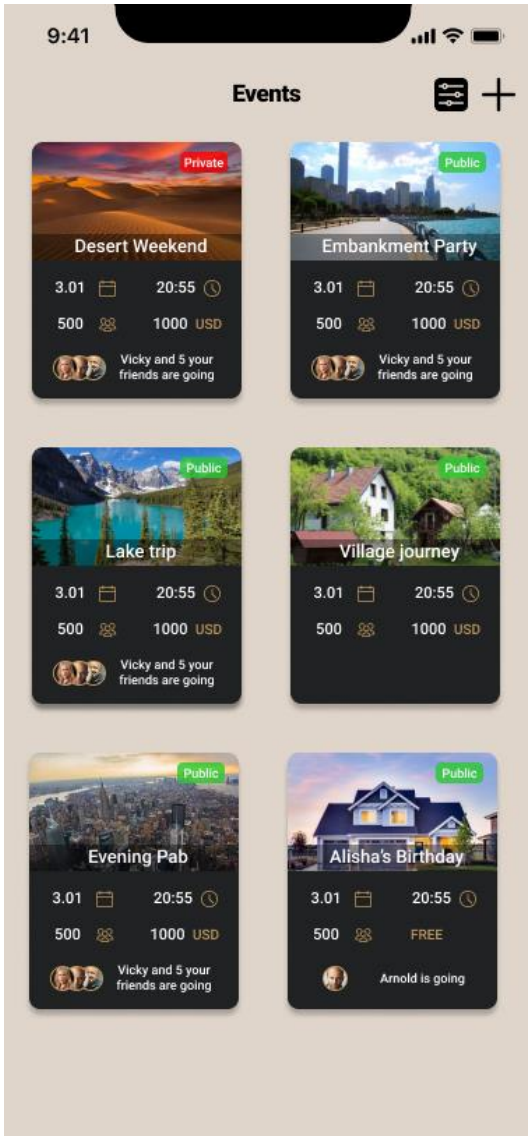

Рис. 4.10. Екран із зустрічами

Список зустрічей 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).

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

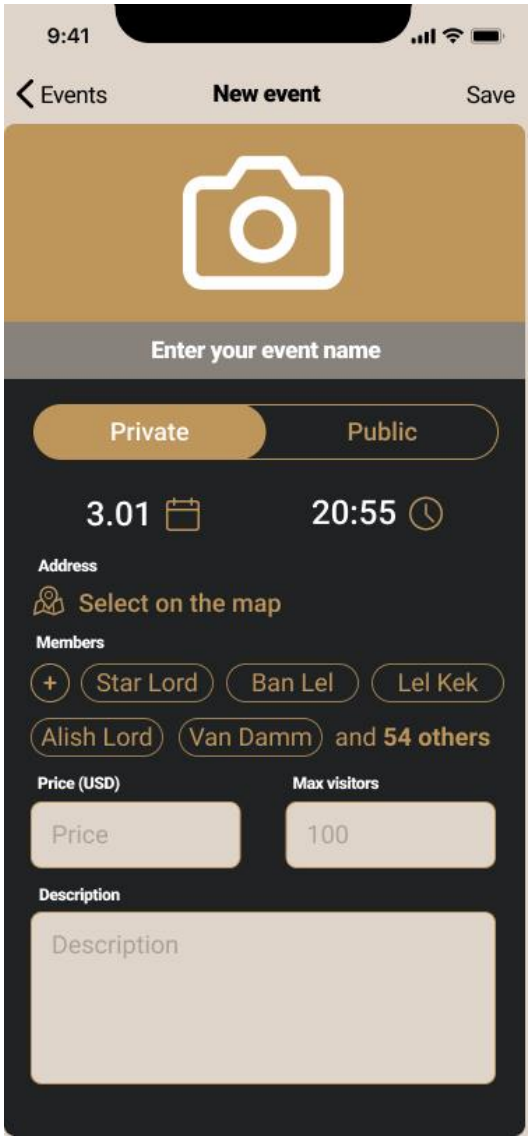

Рис. 4.11. Створення зустрічі

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

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

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

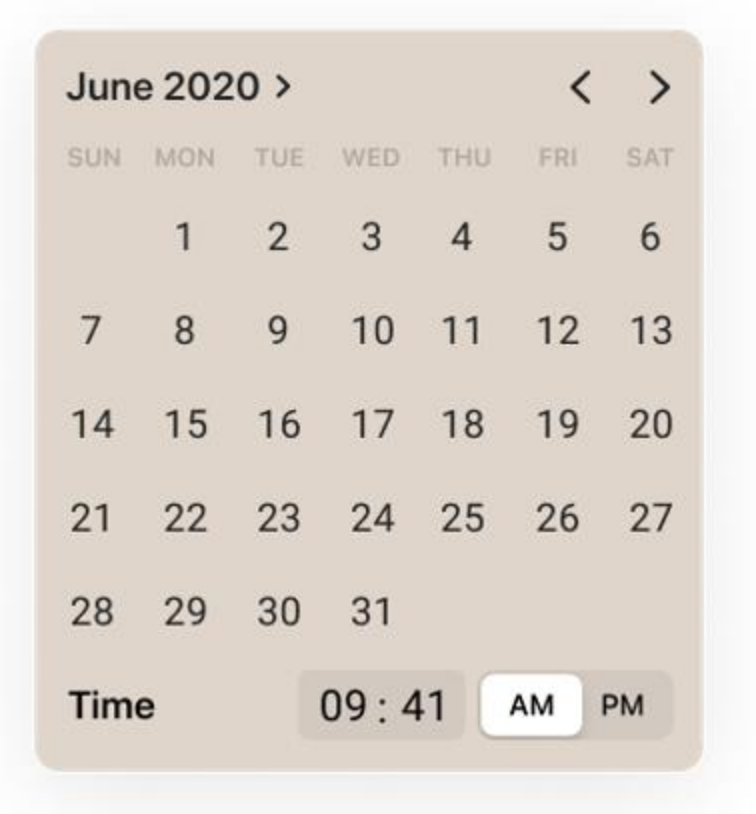

Рис 4.12. DatePicker

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.

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

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

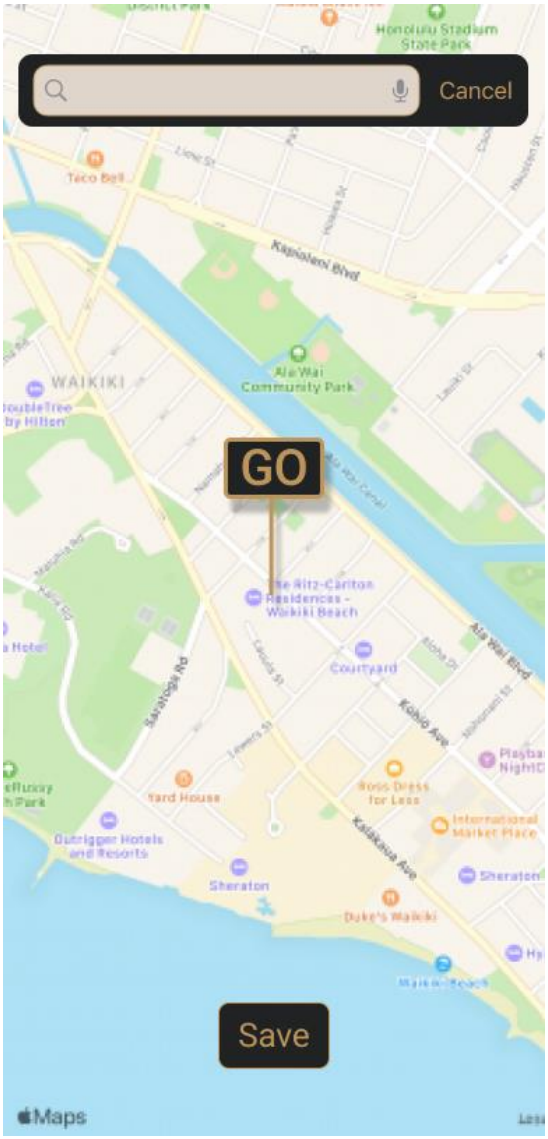

Рис. 4.13. Вибір місця на карті

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.

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

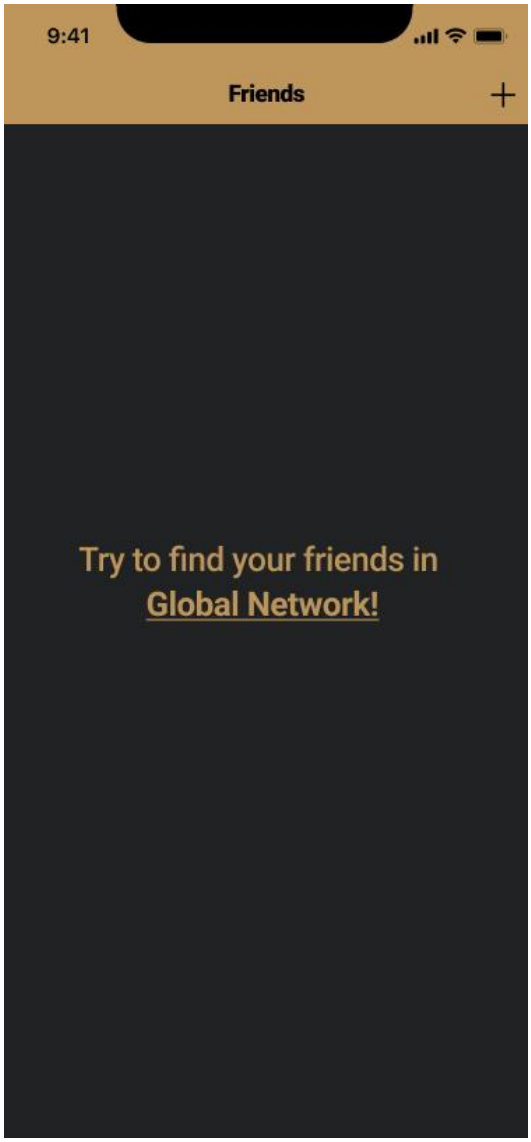

Рис 4.14. Екран друзів

Якщо у користувача не буде жодних друзів у рамках додатку, то буде продемонстрований надпис «Try to find your friends in Global Network!». Натиснувши на підкреслений надпис, відкриється екран глобальної мережі з усіма користувачами (рис. 4.15.).

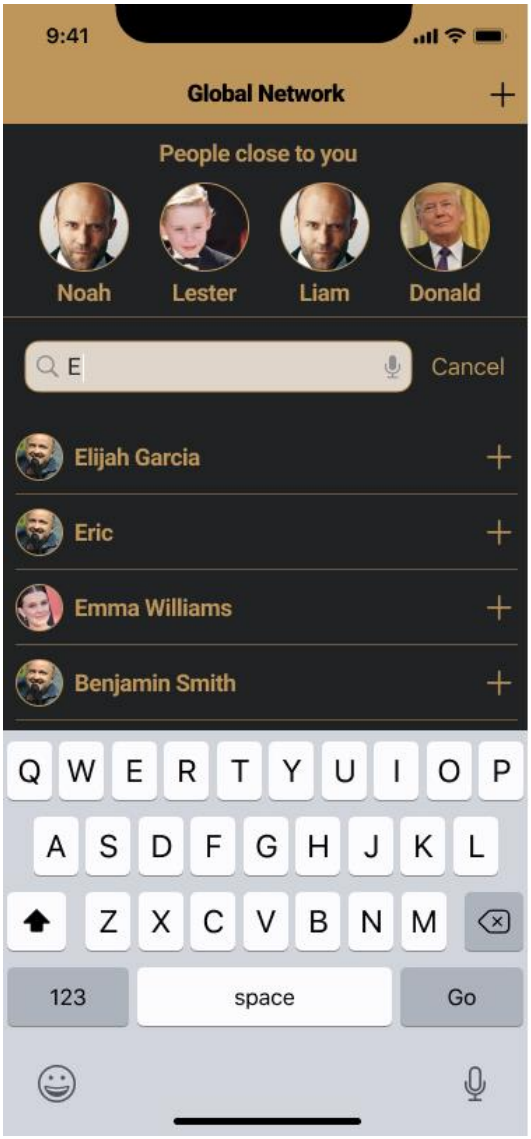

Рис 4.15. Екран глобальної мережі

Тут можна буде знайти як своїх друзів, так і інших користувачів по всьому світу. Основою цього екрану є UITableView - подання, яке представляє дані за допомогою рядків в одному стовпці. Це типовий приклад використання TV. Окремими функціоналом можна визначити можливість додати у друзі людей що знаходяться поруч. Панель буде прихована якщо користувач заборонить доступ до свого місцезнаходження.

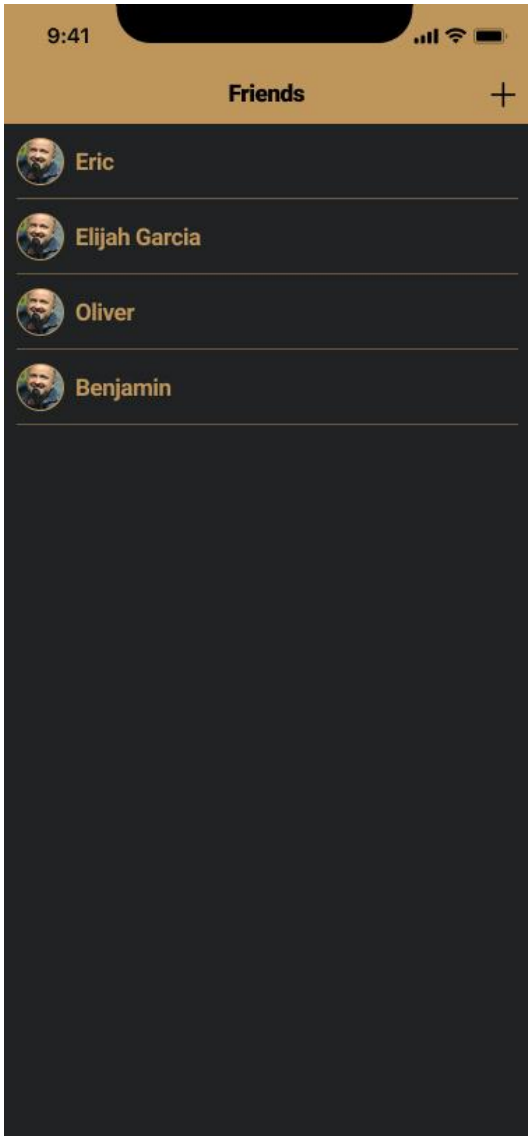

Рис. 4.16. Заповнений екран з друзями

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

Також для кожної зустрічі буде своя кімната з чатом (рис. 4.17.), що буде реалізована за допомогою фреймворку з набором для чата Chatto та БД Realm. Chatto має готові набори повідомлень, що значно полегшує створення чата. Звичайно баду має архітектуру, розроблену спецально для оптимізації роботи чату, завядки чому чат працює плавно та без перебоїв.

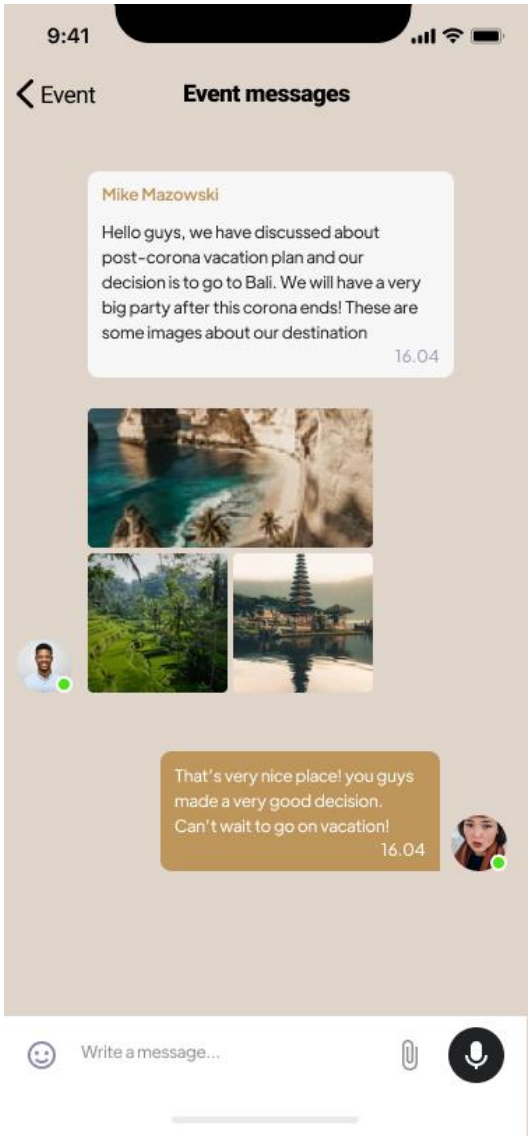

Рис 4.17. Чат

У свою чергу БД Realm забезпечує швидкий запис повідомлень на диск для подальшого відтворення.

За допомогою UITabBar (рис. 4.18.) – елементу керування, який відображає одну або кілька кнопок на панелі вкладок для вибору між різними підзавданнями, представленнями чи режимами програми., буде можливо переключатися між основними екранами.

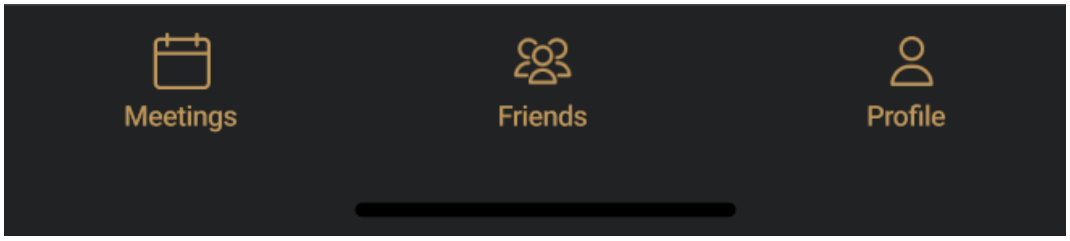

Рис 4.18. Tab Bar

3.3. Перспективи розвитку

На данний момент створено основний функціонал додатку, однак він ще знаходиться у стадії розробки. Планується додавання таких функцій як:

- Відображення користувачів на карті;
- Розширення створення зустрічі списками із завданнями;
- Інтегрування із соціальними мережами;
- Можливість переписуватися й у індивідуальному порядку

ВИСНОВКИ

Отже у ході дипломної роботи було розроблено додаток із основним потрібним функціоналом. Проаналізували типи мобільних додатків, визначили вимоги проекту та різні етапи розробки. Розглянули архітектуру та її значення у створенні додатка. Було проведено аналіз стандартної архітектури Apple MVC, виявлено її недоліки знайдено альтернативу у вигляді MVP.

Розглянули базові та додаткові фреймворки, що були використані у створенні додатку. Кожний фреймворк не тільки полегшував реалізацію потрібного функціоналу, але й пришвидшував розробки, зашкодивши «винаходу велосипеда».

Створивши макет додатку з використанням потужного інструмента Figma, втілили додаток у життя за допомогою мови програмування Swift.

Додаток був протестований не лише на симуляторі а й на реальному пристрої де був доведений до відсутності багів та найбільш плавної відповіді на дії користувача.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

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