Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

РОЗРОБКА ДОДАТКУ УНІВЕРСАЛЬНОГО СКАНЕРА/ГЕНЕРАТОРА КОДІВ З ІСТОРІЄЮ ТА СИНХРОНІЗАЦІЄЮ ЧЕРЕЗ ICLOUD

Виконав: студент 4 курсу групи 2КН-20Б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Кмитюк Д.С.
(прізвище та ініціали)

Керівник: к. т. н. доцент каф. КН
Арсенюк І. Р
(прізвище та ініціали)
«7» червня 2024 р.

Рецензент: : к. т. н. доцент каф. САІТ
Жуков С. О.
(прізвище та ініціали)
«7» червня 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«7» червня 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизаціїКафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)Галузь знань – 12 Інформаційні технології Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«7» червня 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Кмитюку Дмитру Сергійовичу

1. Тема роботи: Розробка додатку універсального сканера/генератора кодів з історією та синхронізацію через iCloud
2. Керівник роботи: Арсенюк Ігор Ростиславович,
Затверджені наказом вищого навчального закладу "11.03.2024 року № 80
3. Термін подання студентом роботи 27.05.2024 р.

Вихідні дані до роботи:

4. Мова програмування Swift, SwiftUI, Середовище розробки XCode;
Програмне забезпечення XCode;
Формати графічних файлів – JPEG,PNG;
Підтримка інтерфейсу баз даних SMSS
5. Зміст текстової частини (перелік питань, які потрібно розробити):
   - аналіз предметної області;
   - постановка задачі;
   - аналіз архітектурних рішень;
   - обгрунтування вибору інструментів технічної реалізації;
   - розробка основних функцій;
   - тестування та публікація;
   - аналіз подальших перспектив додатку;
6. Перелік ілюстративного матеріалу (з точним зазначенням обов'язковихкреслень):
   - Приклад архітектури додатку
   - Демонстрація робочого додатку

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Арсенюк І. Р., доцент каф. КН		

8. Дата видачі завдання 07.03.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви робіт	Термін виконання		Примітка
		початок	закінчення	
1	аналіз предметної області	27.05.2024	28.05.2024	
2	постановка задачі	29.05.2024	30.05.2024	
3	аналіз архітектурних рішень	31.05.2024	01.06.2024	
4	підбір інструментів для реалізації	02.06.2024	03.06.2024	
5	розробка додатку	04.06.2024	05.06.2024	
6	тестування додатку	06.06.2024	07.06.2024	
7	публікація додатку	08.06.2024	09.06.2024	
8	визначення перспектив розвитку додатку	10.06.2024	11.06.2024	
9	розробка інструкція користувача	12.06.2024	13.06.2024	

Студент _____ **Кмитюк Д. С.**
(підпис) *(прізвище та ініціали)*

Керівник роботи _____ **Арсенюк І. Р.**
(підпис) *(прізвище та ініціали)*

АНОТАЦІЯ

Бакалависька дипломна робота складається з 82 сторінок, формату А4, на яких є 20 рисунків, список використаних джерел містить 14 найменувань.

У бакалаврській роботі проведено аналіз предметної області створення інтрнет-магазинів. Досліджено особливості реалізації програмного забезпечення на основі інтелектуальної обробки даних та засобів для сканування та генерування кодів. Розроблено архітектурну модель додатку для сканування та генерації. Проведено порівняння існуючих рішень та визначення переваг і недоліків.

Здійснено програмну реалізацію усіх компонентів програмного забезпечення додатку для сканування та генерації на платформі SwiftUI. Проведено тестування додатку, розміщено додаток на платформі AppStore та розглянуто перспективи подальшого розвитку.

ABSTRACT

The bachelor's thesis consists of 82 pages, A4 format, with 20 figures, the list of references contains 14 titles..

The bachelor's thesis analyzes the subject area of creating online stores. The peculiarities of implementing software based on intelligent data processing and tools for scanning and generating codes were investigated. An architectural model of the scanning and generation application was developed. The existing solutions are compared and their advantages and disadvantages are identified.

The software implementation of all software components of the scanning and generation application on the SwiftUI platform was carried out. The application was tested, the application was placed on the AppStore platform and the prospects for further development were considered.

ЗМІСТ

ВСТУП

Актуальність дослідження. В сучасному світі мобільні додатки відіграють важливу роль у повсякденному житті користувачів, надаючи зручні інструменти для вирішення різноманітних завдань. Однією з корисних функцій, що широко використовується, є сканування та генерація QR-кодів та інших типів штрих-кодів. QR-коди активно застосовуються в маркетингу, логістиці, торгівлі, та багатьох інших сферах. Тому розробка універсального додатку для сканування та генерації кодів з додатковими можливостями є актуальною темою для дослідження. Коди стали невід'ємною частиною сучасного світу, вони відображають технологічні та соціальні процеси, які визначають наше суспільство. Використання кодів сприяє розвитку інновацій, створенню нових можливостей для бізнесу, поліпшенню рівня життя та забезпеченню безпеки інформації. У світі, що стрімко змінюється під впливом цифровізації, існування зручних та функціональних інструментів для роботи з кодами є надзвичайно важливим.

Метою дослідження є розробка універсального додатку сканера/генератора кодів з історією пошуку та синхронізацією через iCloud, що забезпечить зручне користування на різних девайсах

Об’єктом дослідження є процеси комунікацій за допомогою кодів в бізнесі та повсякденному житті

Предметом дослідження є архітектурні рішення, методи передачі інформації та синхронізації даних, що застосовуються в скануванні кодів.

Задачі дослідження включають:

Аналіз предметної області: формулювання вимог до функціональності ти дизайну майбутнього додатку.

Постановку задачі: формулювання вимог до функціональності та дизайну майбутнього додатку

Аналіз архітектурних рішень: вибір найбільш зручних архітектурних підходів для реалізації додатку

Розробку функціоналу: створення інтерфейсу, проектування бази даних, підключення хмари

Тестування: перевірка працездатності та надійності розробленого додатку

Аналіз результатів тестування: оцінка ефективності роботи додатку, виявлення та усунення можливих недоліків

Визначення перспектив розвитку додатку: оцінка можливостей подальшого вдосконалення та масштабування додатку

Апробація роботи

Робота апробувалась на наступній конференції:

Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення. Доповідь: «Обґрунтування доцільності розробки універсального додатку сканера/генератора кодів з історією пошуку та синхронізацією через iCloud» [1].

Публікації: електронні тези конференції Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення. «Інформаційні системи і технології» [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ДОДАТКУ СКАНЕРА/ГЕНЕРАТОРА КОДІВ З ІСТОРІЄЮ ПОШУКУ ТА СИНХРОНІЗАЦІЄЮ ЧЕРЕЗ ICLOUD

1.1 Аналіз предметної області сканування кодів

Розвиток технологій сканування кодів є важливим розділом у галузі інформаційних технологій. Від моменту винаходу штрих-коду у 1948 році Норманом Джозефом Вудлендом та Бернардом Сільвером до сучасних додатків, що використовують QR-коди, цей шлях демонструє значний технологічний прогрес і зміну підходів до обробки інформації. Впровадження перших комерційних систем сканування у 1970-х роках дало потужний імпульс розвитку технологій автоматичної ідентифікації. Важливою віхою стало створення UPC (універсального товарного коду) у 1974 році, який широко використовується в ритейлі по всьому світу.

У 1994 році, винахід японської компанії Denso Wave – QR-код – змінив підхід до зберігання та швидкого доступу до інформації. QR-коди забезпечують високу щільність даних та швидке зчитування, що робить їх ідеальними для використання в мобільних додатках і маркетингових кампаніях.

Сучасні мобільні додатки, такі як Google Lens та Apple Wallet, дозволяють користувачам сканувати різноманітні типи кодів, включаючи штрих-коди та QR-коди, забезпечуючи зручний спосіб взаємодії з цифровим контентом та послугами. Важливим фактором їхнього успіху є інтеграція з хмарними сервісами, такими як iCloud, що дозволяє зберігати історію сканування та синхронізувати її між різними пристроями.

Сканування та генерація кодів стали важливими інструментами у багатьох сферах, таких як роздрібна торгівля, логістика, охорона здоров'я, маркетинг та багато інших. Використання QR-кодів, штрих-кодів та інших

типів кодів значно полегшує процеси обміну інформацією, ідентифікації товарів та доступу до цифрових ресурсів.

До основних тенденцій у цій сфері належать:

- Зростання популярності QR-кодів: QR-коди використовуються для різноманітних цілей — від маркетингових кампаній та електронних квитків до безконтактних платежів і доступу до веб-сайтів.
- Розширення використання штрих-кодів: Штрих-коди залишаються незамінними у роздрібній торгівлі та логістиці для ідентифікації товарів та відстеження запасів.
- Поява нових типів кодів: Окрім традиційних QR-кодів і штрих-кодів, з'являються нові формати, такі як Data Matrix, Aztec та PDF417, які забезпечують більшу гнучкість та обсяг інформації.
- Інтеграція з іншими технологіями: Сканування кодів все частіше інтегрується з технологіями доповненої реальності (AR), IoT та блокчейн, що відкриває нові можливості для взаємодії з цифровими даними.

Мобільні додатки для сканування кодів повинні відповідати ряду вимог для забезпечення зручності та ефективності використання:

- Зручний інтерфейс користувача: Інтерфейс повинен бути інтуїтивно зрозумілим та простим у використанні, забезпечуючи швидкий доступ до основних функцій.
- Підтримка різних типів кодів: Додаток має підтримувати сканування та генерацію різних типів кодів, таких як QR-коди, штрих-коди, Data Matrix тощо.
- Висока швидкість та точність сканування: Використання алгоритмів, що забезпечують швидке та точне розпізнавання кодів навіть за умов недостатнього освітлення або пошкодження

коду.

- Збереження історії пошуку: Можливість збереження сканованих кодів для подальшого використання та аналізу.
- Синхронізація даних: Підтримка синхронізації з хмарними сервісами, такими як iCloud, для доступу до даних з різних пристроїв.

На сьогоднішній день, без кодів не обходиться жодна бізнес-зустріч. Люди використовують коди починаючи з банального сканування меню у закладах, закінчуючи оплатами у інтернет-магазинах. Тому, розробка універсального сканера-генератора буде гарним рішенням для кожної людини з різних сфер [1-4].

1.2 Аналіз сучасної класифікації кодів

QR-коди (Quick Response Codes) - це двовимірні матричні коди, які складаються з чорних і білих квадратів на квадратному полі. Вони можуть кодувати велику кількість інформації: від тексту до посилань на веб-сайти. QR-коди широко використовуються в маркетингу для посилань на веб-сторінки, в мобільних додатках для здійснення платежів, у квитках на заходи та у логістиці для відстеження товарів.

Штрих-коди (Barcodes) - це одновимірні лінійні коди, які складаються з паралельних ліній різної товщини та інтервалів між ними. Вони кодують інформацію у вигляді цифр або букв. Найбільш поширені в роздрібній торгівлі для кодування товарів, у виробництві для відстеження компонентів, а також у логістиці для сканування ідентифікаторів вантажів.

Aztec-коди – це двовимірні матричні коди, що складаються з концентричних квадратів і зони даних навколо них. Вони були розроблені для швидкого сканування, навіть на невеликих площах. Використовуються

у транспорті для мобільних квитків, у медицині для маркування пацієнтів і зразків, а також у промисловості для мікро-маркування.

Data Matrix-коди – це ще один вид двовимірних матричних кодів, що складаються з чорних і білих комірок, організованих у квадратну або прямокутну матрицю. Широко використовуються в промисловості для маркування деталей та компонентів, у фармацевтиці для маркування ліків, а також у логістиці для відстеження товарів.

PDF417-коди – це двовимірні штрих-коди, які використовують декілька рядів і колонок для кодування інформації. Вони здатні зберігати великі обсяги даних. Використовуються у перевезеннях для кодування великих обсягів даних на вантажних етикетках, у фінансовій сфері для кодування чеків і квитанцій, а також у сфері державних послуг для документів і посвідчень.

Кожен тип коду має свої унікальні переваги та сферу застосування. Вибір коду залежить від потреб конкретного бізнесу або процесу, обсягів даних, які потрібно кодувати, і умов, в яких ці дані будуть використовуватись. Розуміння відмінностей між ними дозволяє ефективно використовувати технологію кодів для підвищення ефективності і точності бізнес-процесів.

1.3 Аналіз об’єкту проектування

1.3.1 Постановка задачі на оптимізування додатку під різні пристрої

Адаптація додатку під різні пристрої – це важлива частина розробки,що додає комфортного користування на екранах різного розміру. Оптимізація мобільного додатка під різні розміри екранів є критично важливою для забезпечення якісного користувацького досвіду. По-перше,

користувачі використовують різноманітні пристрої, від маленьких смартфонів до великих планшетів, і додаток повинен бути зручним та функціональним на всіх цих пристроях. По-друге, оптимізація під різні розміри екранів сприяє збереженню естетичного вигляду та читабельності інтерфейсу, що знижує ймовірність відтоку користувачів через поганий дизайн. Крім того, адаптивний дизайн дозволяє додатку залишатися конкурентоспроможним на ринку, де користувачі очікують бездоганного та безперебійного використання.

Для коректної роботи та адаптації додатку система повинна відповідати наступним вимогам:

- Використання SwiftUI фреймворку для створення адаптивного дизайну. SwiftUI надає готові компоненти та стилі які дозволяють швидко та ефективно реалізувати адаптивність додатку
- Тестування додатку на різних девайсах для забезпечення коректного відображення елементів

Найважливішим аспектом в оптимізації під різні девайси є використання SwiftUI, він значно спрощує процес оптимізації, адже розмір елементів налаштовується автоматично, завдяки декларативному підходу програмування.

1.3.2 Характеристика та аналіз додатків-аналогів

На ринку існує багато додатків для сканування та генерації кодів, які пропонують різні функціональні можливості. Розглянемо декілька найбільш популярних рішень:

- QR Code Scanner & Barcode Reader: Один з найпопулярніших додатків для сканування QR-кодів, який відрізняється високою швидкістю та точністю роботи.
- QR & Barcode Reader: Цей додаток підтримує широкий спектр

форматів кодів та забезпечує високу точність сканування.

- QR Scan Master: Додаток з простим інтерфейсом, який підтримує сканування та генерацію різних типів кодів, а також збереження історії пошуку.
- QR Code: Додаток, що забезпечує додатковий захист від фішингових атак при скануванні кодів.

Для більш детального аналізу, переглянемо один із додатків на платформі AppMagic. На рисунку 1.1 зображено кількість користувачів, які завантажили цей додаток. На рисунку 1.2 зображено заробіток додатку

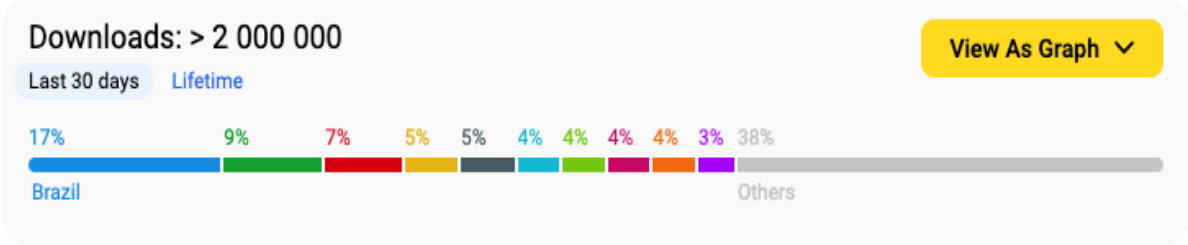


Рисунок 1.1 – «Кількість завантажуваннь додатку “QR & Barcode Reader”»

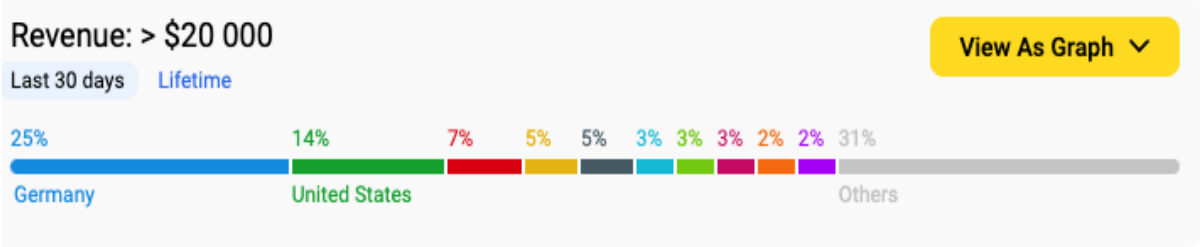


Рисунок 1.2 – «Кількість коштів, які заробив додаток “QR & Barcode Reader”»

Цифри дійсно вражають, але це звичайний додаток, який використовує прості інструменти та не містить ніякої синхронізації з хмарним середовищем.

Аналіз додатків у предметній області сканування кодів показує, що даний напрямок є важливим та перспективним для розвитку мобільних додатків. Сучасні тенденції вказують на зростання популярності QR-кодів, розширення використання штрих-кодів та появу нових типів кодів. Основні вимоги до мобільних додатків включають зручний інтерфейс, підтримку різних типів кодів, високу швидкість та точність сканування, збереження історії пошуку та синхронізацію даних. Огляд існуючих рішень на ринку допомагає визначити ключові функціональні можливості та особливості, які повинні бути враховані при розробці власного додатку.

1.3.3 Постановка задач на розробку

Отже, коли інструменти для розробки підібрані та визначено архітектуру, можна проектувати розробку. Перш за все, потрібно спроектувати основні чотири екрани:

- Екран сканування
- Екран історії пошуку
- Екран генерації
- Екран налаштувань

Екран сканування – на цьому екрані буде зображено камеру, яка, на основі штучного інтелекту MLKit, буде виявляти та сканувати коди. Камеру можна налаштувати під себе на екрані налаштувань

Екран історії пошуку – на цьому екрані, у вигляді списку буде зображено усі відскановані та згенеровані коди. Також доступні фільтри, для того, щоб користувач щвидше зміг знайти потрібний код.

Екран генерації – на цьому екрані, користувач зможе згенерувати потрібний йому код та кастомізувати під свої потреби. Також можна одразу поширити цей код у соцмережі.

Екран налаштувань – на цьому екрані користувач зможе налаштувати додаток на свій розсуд. Доступні функції:

- увімкнути/вимкнути автоматичне сканування
- увімкнути/вимкнути спалах при скануванні
- увімкнути/вимкнути мультисканування
- придбати преміум-версію додатку

Наступним етапом буде розробка локальної бази даних, для використання історії пошуку. Після створення бази даних, необхідно підключити її до iCloud [5-7].

1.4 Висновки

Розвиток технологій сканування кодів та їх інтеграція в сучасні мобільні додатки демонструє важливий етап в еволюції інформаційних систем. Від винаходу штрих-коду у 1948 році до появи QR-кодів та їхнього широкого застосування, технології сканування зазнали значних змін, спрямованих на поліпшення обробки та доступу до інформації.

Сьогодні мобільні додатки для сканування та генерування кодів відіграють критичну роль у багатьох аспектах бізнесу та повсякденного життя. Вони пропонують користувачам широкий спектр можливостей, від зручного сканування товарів у магазинах до створення персоналізованих кодів для різних цілей. Інтеграція з хмарними сервісами, такими як iCloud, забезпечує збереження та синхронізацію даних, що дозволяє користувачам мати постійний доступ до своєї історії сканувань і налаштувань з будь-якого пристрою.

Наш додаток для сканування та генерування кодів відповідає сучасним вимогам і включає такі ключові функції:

Підтримка різних типів кодів: Додаток дозволяє сканувати штрих-коди, QR-коди та інші формати, що робить його універсальним інструментом для користувачів.

Генерація кодів: Можливість створювати власні коди дозволяє користувачам обмінюватися інформацією та використовувати додаток у бізнесі.

Хмарна синхронізація: Інтеграція з iCloud забезпечує збереження історії сканувань та синхронізацію налаштувань, що є ключовою перевагою для зручності та безперервності використання.

Зручний інтерфейс: Додаток розроблений з урахуванням принципів інтуїтивного користування, що спрощує доступ до всіх його можливостей та функцій.

Розробка такого додатку є актуальною у сучасних умовах, оскільки він відповідає потребам користувачів у зручному та ефективному управлінні інформацією. Поєднання можливостей сканування та генерації кодів, збереження та синхронізації даних з використанням хмарних сервісів, дозволяє створити конкурентоспроможний продукт, який відповідає вимогам ринку електронної комерції та інформаційних технологій.

Загалом, впровадження індивідуального додатку для сканування та генерування кодів з інтеграцією iCloud представляє собою значний крок вперед у забезпеченні ефективного і зручного способу обробки та збереження інформації. Це відкриває нові можливості для бізнесу та користувачів, дозволяючи їм максимально ефективно використовувати переваги сучасних технологій.

Проєкт має великий потенціал завдяки поєднанню універсальності та гнучкості налаштувань. Аналіз існуючих рішень показав, що, хоча популярні додатки є зручними та функціональними, вони мають обмежені можливості налаштування та специфічні недоліки. Цей проєкт, навпаки, буде легко адаптуватися під різні типи бізнесів і товарів, забезпечуючи

оптимальний користувацький досвід завдяки адаптивному дизайну та врахуванню індивідуальних потреб користувачів.

2 АНАЛІЗ ЗАСОБІВ ТА МЕТОДІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ДОДАТКУ СКАНЕРА/ГЕНЕРАТОРА КОДІВ

2.1 Вибір архітектури

2.1.1 Загальний опис архітектури для додатку

Вибір архітектури для мобільного додатку є критичним етапом у процесі розробки, який значно впливає на успіх кінцевого продукту. Архітектура додатку визначає, як його компоненти взаємодіють один з одним, як вони обробляють дані та виконують функції, а також наскільки легко додаток можна масштабувати, підтримувати та розширювати в майбутньому. Розглянемо кілька ключових причин, чому вибір правильної архітектури є важливим:

Правильно обрана архітектура забезпечує оптимальну продуктивність додатку. Наприклад, розподіл обробки даних між клієнтською та серверною частинами дозволяє зменшити навантаження на пристрій користувача та покращити швидкість реакції додатку. Неправильний вибір архітектури може призвести до затримок, повільної роботи або навіть до збоїв у роботі, що негативно вплине на користувацький досвід.

Мобільні додатки, особливо ті, що розраховані на велику аудиторію, повинні бути здатні обробляти зростаючі обсяги користувацьких запитів і даних. Вибір масштабованої архітектури дозволяє легко додавати нові функції та розширювати систему без значних змін у базовому коді. Наприклад, використання мікросервісної архітектури полегшує додавання нових компонентів без порушення роботи вже існуючих.

Модульна та добре структурована архітектура полегшує підтримку та обслуговування додатку. Чіткий розподіл відповідальностей між компонентами спрощує діагностику проблем і їх вирішення. Крім того, це

дозволяє різним командам працювати над різними частинами додатку незалежно, що прискорює процес розробки та знижує ймовірність конфліктів в коді [6-9].

Архітектура додатку також визначає, як дані передаються і зберігаються, що безпосередньо впливає на безпеку додатку. Вибір архітектури з урахуванням безпеки дозволяє ефективно захистити дані користувачів від несанкціонованого доступу та загроз. Наприклад, використання шифрування при передачі даних та ізольованих середовищ для обробки конфіденційної інформації може значно підвищити рівень безпеки. Мобільні додатки часто потребують адаптації до нових вимог або змін у бізнес-логіці. Гнучка архітектура дозволяє швидко вносити зміни та інтегрувати нові технології або сервіси. Наприклад, використання архітектури, що базується на компонентному або сервіс-орієнтованому підході, дозволяє додавати або змінювати окремі функції без значних переробок всього додатку.

Добре спланована архітектура сприяє створенню зручного та інтуїтивного користувацького досвіду. Вона дозволяє забезпечити стабільну роботу додатку, швидкий відгук на дії користувача та безперебійну інтеграцію з іншими системами. Відповідність архітектури специфічним вимогам користувачів і бізнесу гарантує, що додаток буде відповідати очікуванням і потребам кінцевих користувачів.

У сучасному світі більшість мобільних додатків не існують в ізоляції. Вони повинні взаємодіяти з різними сервісами та платформами, такими як соціальні мережі, системи обробки платежів або хмарні сховища. Вибір архітектури, яка підтримує легку інтеграцію з зовнішніми сервісами, є ключовим для успішної роботи додатку в складній екосистемі.

У випадках, коли над додатком працюють великі або розподілені команди, архітектура відіграє вирішальну роль у координації та управлінні роботою. Чітка і зрозуміла архітектура дозволяє ефективніше розподіляти

завдання між командами, що прискорює процес розробки та знижує ризики виникнення помилок.

Вибір архітектури для мобільного додатку є стратегічним рішенням, яке визначає його довгостроковий успіх і здатність відповідати вимогам ринку. Це рішення впливає на всі аспекти розробки та експлуатації додатку – від продуктивності та масштабованості до зручності користування та безпеки. Успішний мобільний додаток потребує ретельного планування архітектури, яка забезпечить гнучкість, надійність і ефективність протягом всього життєвого циклу продукту.

2.1.2 Аналіз існуючих архітектурних паттернів

В сучасінй iOS розробці найчастіше використовують п’ять архітектур:

- MVC(Model View Controller)
- MVP(Model View Presenter)
- VIPER(View Iterator Presenter Router Entity)
- TCA(The Composable Arcitecture)
- MVVM(Model View ViewModel)

Детальніше пройдемось по кожній:

MVC (Model-View-Controller) є класичною архітектурною парадигмою, що широко використовується у розробці iOS додатків. Вона надає чіткий спосіб організації коду, розділяючи додаток на три основні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Це розділення сприяє більш структурованій розробці, полегшує підтримку та розширення додатків. Давайте детально розглянемо, як кожен з цих компонентів функціонує в контексті iOS розробки.

Model (Модель) - фундаментальна частина додатку, відповідальною за управління даними і логікою бізнес-процесів. Вона містить дані, правила

їх обробки та взаємодії. В iOS розробці моделі можуть включати різні типи структур даних і сервіси, такі як локальні бази даних (наприклад, Core Data), мережеві API клієнти, або прості структури для зберігання даних.

View (Представлення) - відповідає за відображення даних та взаємодію з користувачем. Воно відображає інформацію з моделі та оновлюється відповідно до змін в даних. У iOS, представлення зазвичай реалізовані за допомогою UIView або його підкласів, таких як UILabel, UIButton, UITableView тощо. Представлення не містить логіки для обробки дій користувача чи маніпуляцій з даними — це завдання контролера.

Controller (Контролер) - посередник між Моделлю та Представленням. Він управляє логікою додатку, відповідає за обробку дій користувача, взаємодіє з моделлю для отримання та оновлення даних і передає ці дані представленню для відображення. В iOS контролери зазвичай реалізовані як підкласи UIViewController. Контролер слухає події від представлення і реагує на них, часто оновлюючи модель або саме представлення.

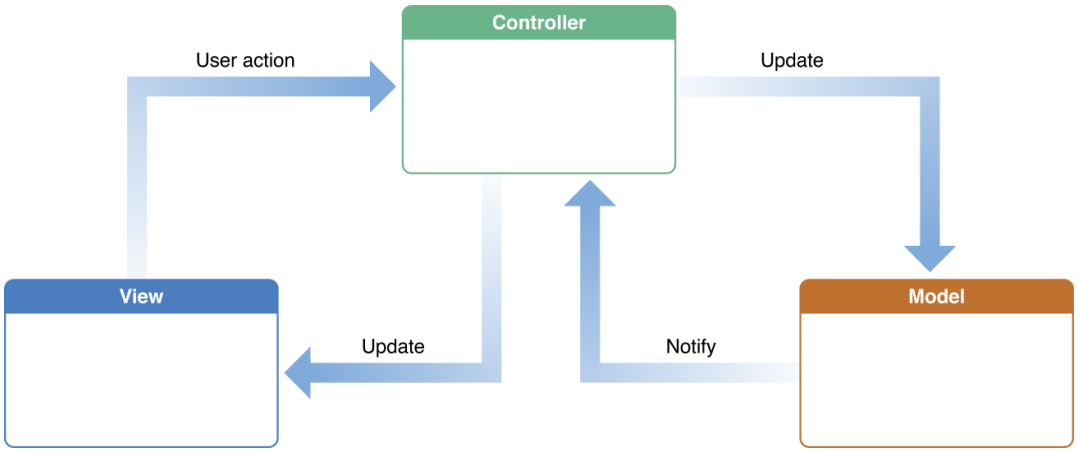


Рисунок 2.1 - «Приклад роботи архітектури MVC»

Архітектура MVC є фундаментальним підходом у розробці iOS додатків, що забезпечує чітку організацію коду. Вона дозволяє розділити додаток на незалежні компоненти, кожен з яких виконує свою конкретну

функцію. Це робить додатки більш підтримуваними, гнучкими та зручними у розширенні. Проте, важливо правильно організувати роботу компонентів, щоб уникнути проблем з масштабованістю та підтримкою в майбутньому. MVC залишається популярним вибором завдяки своїй простоті та ефективності в багатьох сценаріях розробки.

VIPER(View-Interactor-Presenter-Entity-Router) — це архітектурний паттерн, який надає більш модульний і розділений підхід до організації коду в iOS додатках порівняно з класичною архітектурою MVC. Він забезпечує суворий розподіл обов'язків між компонентами, що полегшує підтримку, розширення та тестування додатків. VIPER є популярним вибором для великих і складних додатків, де необхідно чітко розмежувати бізнес-логіку, інтерфейс користувача і навігацію.

View (Представлення) - відповідає за відображення даних і взаємодію з користувачем. Воно отримує інструкції від Презентера і відображає дані відповідно до них. Представлення є пасивним компонентом, тобто воно лише виконує вказівки Презентера, не обробляючи бізнес-логіку самостійно.

Interactor (Інтерактор) - є центром бізнес-логіки додатку. Він обробляє запити від Презентера, маніпулює даними і повертає результат назад до Презентера. Інтерактор також взаємодіє з Моделлю (Entities) для отримання або зберігання даних. Він не знає деталей реалізації Представлення, фокусуючись виключно на логіці додатку.

Presenter (Презентер) - Презентер є посередником між Представленням і Інтерактором. Він отримує дані від Інтерактора, перетворює їх у формат, що підходить для відображення, і передає Представленню для відображення. Презентер також реагує на взаємодії користувача, повідомляючи про них Інтерактору. Презентер не містить посилань на UIKit або будь-які деталі реалізації інтерфейсу, він зосереджений на презентації даних.

Entity (Модель) - Модель або Entity у VIPER представляє структуру даних додатку. Вона включає об'єкти, які зберігають основну інформацію про додаток, наприклад, дані з бази даних або мережевих API. Вона не містить бізнес-логіки, а лише визначає формат даних, з якими працює Інтерактор.

Router (Маршрутизатор) - Маршрутизатор відповідає за навігацію між екранами додатку. Він управляє переходами і координацією між різними модулями VIPER. Маршрутизатор визначає маршрути для переходу до інших модулів і здійснює ці переходи, забезпечуючи ізоляцію логіки навігації від інших компонентів.

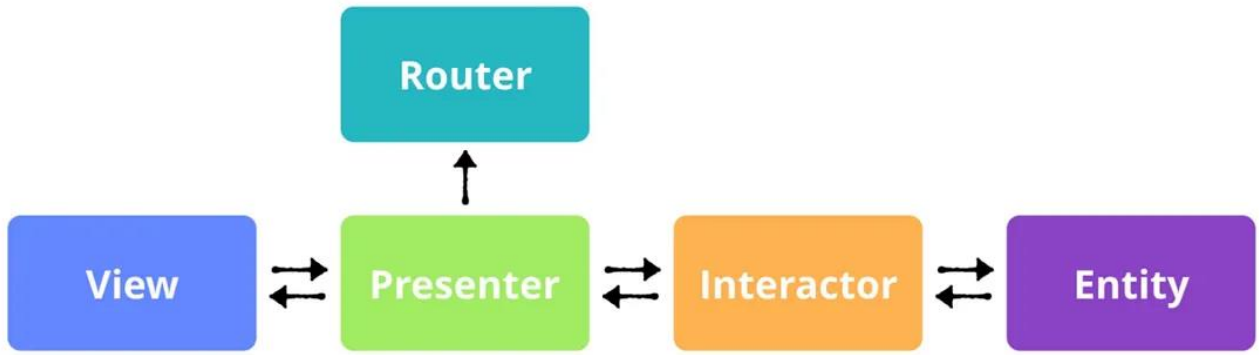


Рисунок 2.2 – «Схема роботи архітектури VIPER»

Архітектура VIPER відома своїм суворим розподілом обов'язків між компонентами додатку, що робить її потужним інструментом для розробки складних та масштабних iOS-додатків. VIPER надає:

- Чітка модульність: Кожен компонент VIPER має визначені завдання, що зменшує залежність між частинами системи і сприяє більш чистому та зрозумілому коду.

- Масштабованість: Завдяки своїй структурованій природі, VIPER полегшує додавання нових функцій та підтримку великих проектів, що постійно зростають.
- Покращене тестування: VIPER розбиває логіку на невеликі компоненти, що спрощує тестування і підвищує надійність системи.

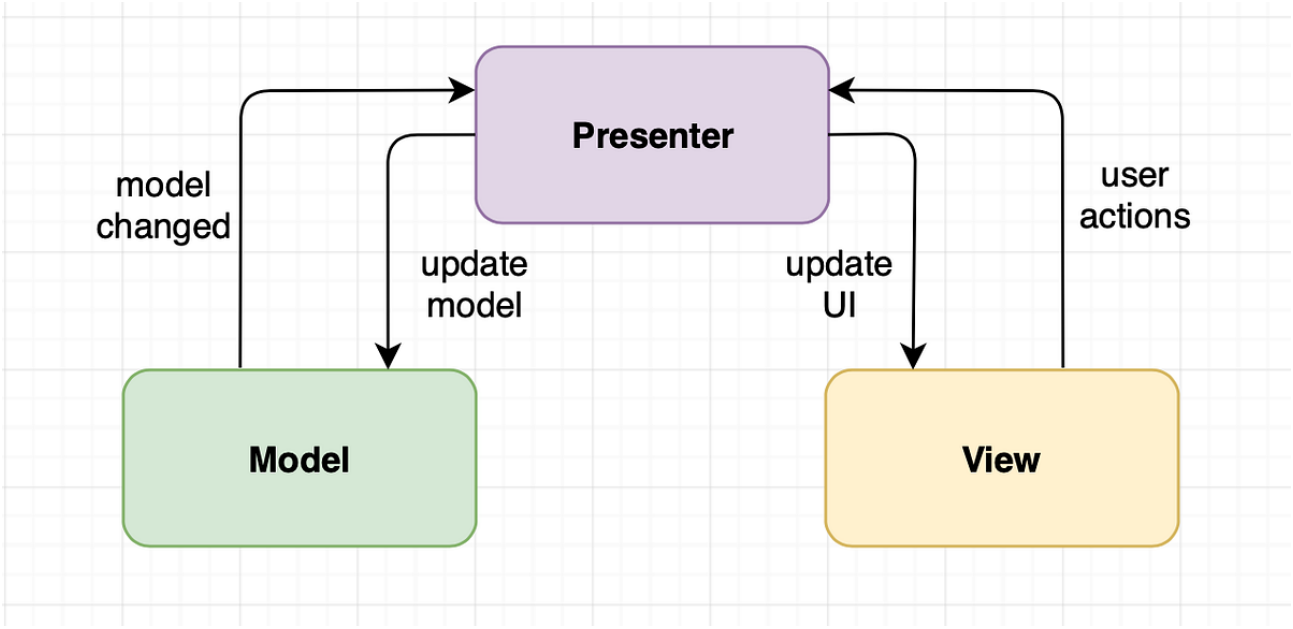
Попри численні переваги, VIPER має свої виклики. Він може збільшити складність проекту, оскільки кожна функція розділена на п'ять різних компонентів, що може призвести до більшого обсягу коду і складності в координації між ними. VIPER є відмінним вибором для великих і складних додатків, які потребують суворого розподілу обов'язків, високої масштабованості та легкої підтримки. Це робить його ідеальним для команд, що працюють над проектами з довготривалими планами розвитку і розширення.

MVP (Model-View-Presenter) — це архітектурний паттерн, який пропонує структурований підхід до розробки мобільних додатків. MVP надає розділення обов'язків між компонентами додатку, поділяючи його на три основні частини: Model (Модель), View (Представлення) та Presenter (Презентер). Цей паттерн полегшує підтримку коду, дозволяє краще тестувати компоненти і сприяє більш чіткому розподілу логіки.

Model (Модель) - представляє дані додатку та бізнес-логіку, що відповідає за обробку цих даних. Вона взаємодіє з джерелами даних, такими як бази даних або веб-сервіси, і надає їх у форматі, зручному для подальшої обробки. Модель у MVP не знає нічого про Представлення або Презентер, вона лише зберігає і маніпулює даними.

View (Представлення) - відповідає за відображення даних і взаємодію з користувачем. Воно отримує дані від Презентера і відображає їх. Представлення у MVP є пасивним компонентом, що означає, що воно просто виконує команди Презентера, не містячи бізнес-логіки.

Presenter (Презентер) - є центральною частиною архітектури MVP, що відповідає за зв'язок між Моделлю та Представленням. Він отримує дані від Моделі, обробляє їх і передає Представленню для відображення. Презентер також реагує на взаємодії користувача через Представлення, передаючи запити до Моделі. Презентер не містить посилань на UIKit або конкретні деталі інтерфейсу, зосереджуючись на презентації та логіці обробки даних.



«Рисунок 2.3 – Схема роботи архітектури MVP»

Архітектура MVP (Model-View-Presenter) є популярним вибором для розробки iOS додатків завдяки своїй здатності чітко розділяти логіку представлення, обробки даних і взаємодії з користувачем. MVP підходить для середніх та великих додатків, забезпечуючи:

- Чітке розділення обов'язків: Кожен компонент у системі має свої конкретні обов'язки, що спрощує підтримку та розвиток додатку.
- Полегшене тестування: Оскільки Presenter не залежить від UI-компонентів, його можна легко тестувати за допомогою модульних тестів, що підвищує надійність додатку.

- Гнучкість і масштабованість: MVP дозволяє легко додавати нові функції та змінювати існуючі, не зачіпаючи інші компоненти системи.

Однак, MVP має і свої виклики. Для простих додатків може виникнути надмірність коду, що додає складності до проекту. Крім того, підтримка чіткого поділу ролей вимагає додаткових зусиль у координації між компонентами.

MVP є оптимальним рішенням для проектів, які потребують добре організованого коду з чітким розподілом обов'язків і підтримкою. Це робить його ефективним вибором для багатьох iOS-додатків, від середніх до великих за складністю.

The Composable Architecture (TCA) — це модель архітектури для розробки iOS-додатків, що спирається на принципи простоти, чистоти та модульності. Ця архітектура дозволяє розбити додаток на невеликі, самодостатні компоненти, які легко розуміти, тестувати та розвивати. Основні концепції TCA включають:

Стан додатку - Усі дані, що використовуються у додатку, представлені у вигляді незмінної моделі. Це дозволяє зберігати консистентність та передбачуваність у стані додатку.

Ефекти та обробники подій - Дії, такі як мережеві запити або анімації, обробляються у вигляді чистих функцій, що дозволяє керувати зовнішніми впливами та підтримувати чистоту коду.

Редуктор - Це функція, що приймає поточний стан додатку та подію, і повертає оновлений стан. Редуктор дозволяє систематично змінювати стан додатку відповідно до користувацьких дій або зовнішніх подій.

Тестування - TCA сприяє тестуванню завдяки своїй модульності та простоті. Кожен компонент може бути протестований окремо, що сприяє забезпеченню надійності та якості коду.

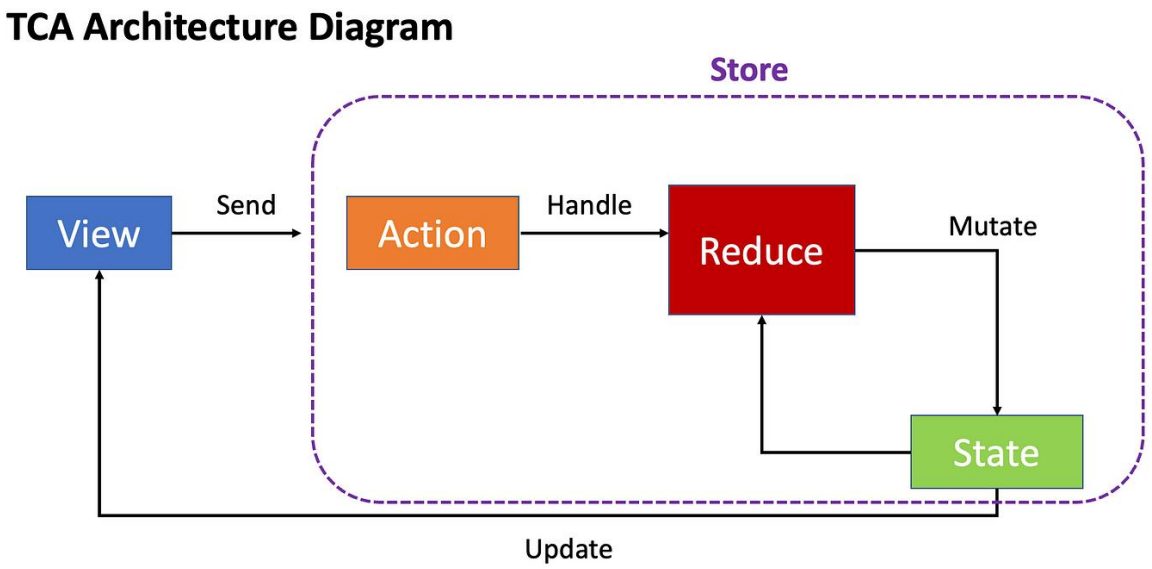


Рисунок 2.4 – «Схема роботи архітектури TCA»

The Composable Architecture — це потужний інструмент для розробки iOS-додатків, що спрощує розуміння та управління станом додатку. З його допомогою розробники можуть швидко та ефективно створювати додатки, які є надійними, легкими для розуміння та тестування.

MVVM (Model-View-ViewModel) - це популярний підхід для розробки iOS-додатків, що дозволяє чітко розділити логіку, представлення та дані додатку.

Модель (Model) - Відповідає за управління даними додатку та бізнес-логікою. Модель може бути незалежним класом або структурою, що забезпечує доступ до даних та виконання операцій з ними.

Вигляд (View) - Представлення даних користувачу. Відображає інформацію та реагує на взаємодію користувача. У контексті iOS це може бути UIView або SwiftUI View.

ViewModel - Проміжний шар між Моделлю та Виглядом. ViewModel містить логіку, яка підготовлює дані для відображення у Вигляді та обробляє користувацькі дії. Вона також може включати логіку валідації та навігації.

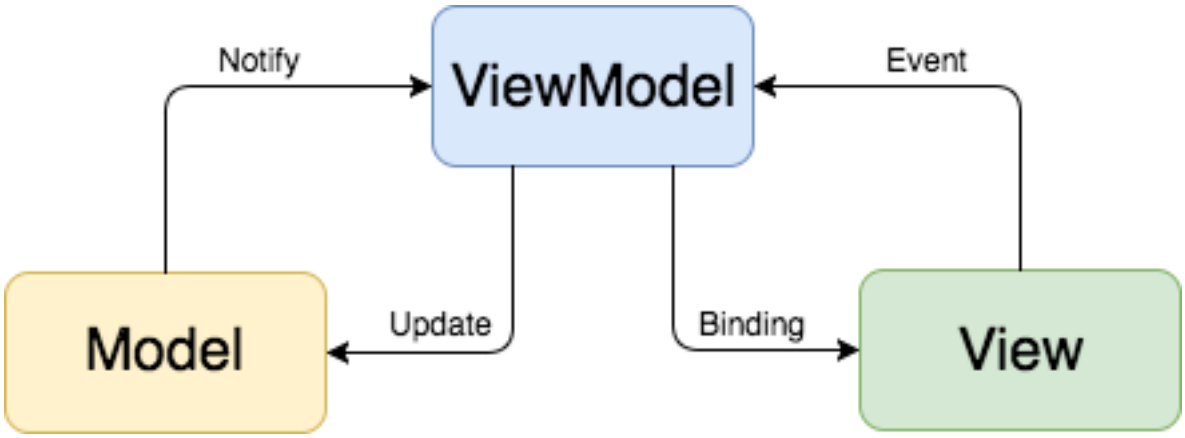


Рисунок 3.5 – «Схема роботи архітектури MVVM»

MVVM дозволяє розбити додаток на незалежні компоненти, що полегшує розуміння коду та його підтримку. Він також сприяє тестуванню завдяки відокремленню логіки представлення від бізнес-логіки. Завдяки MVVM розробники можуть створювати додатки, які є ефективними, масштабованими та легкими для розуміння.

Архітектура MVVM є ідеальним вибором для нашого додатку. Вона надасть нам зручність у роботі з багатьма компонентами додатку, що ми буде реалізовано.

- Модель (Model): Ваша модель буде відповідати за управління даними, пов'язаними з кодами та їх історією пошуку. Вона також буде відповідати за взаємодію з iCloud для синхронізації даних між пристроями.
- Вигляд (View): У Вигляді ви реалізуєте інтерфейс для сканування та відображення згенерованих кодів, а також відображення історії пошуку. Цей компонент буде відповідати за відображення даних та взаємодію з користувачем.
- ViewModel: ViewModel буде відповідати за підготовку даних для відображення у Вигляді та за обробку взаємодії користувача. Вона

буде відділена від Вигляду і Моделі, що дозволить зробити ваш код більш чистим та масштабованим.

- Синхронізація з iCloud: Використання MVVM також спрощує імплементацію синхронізації з iCloud, оскільки це може бути вбудованою частиною вашої Моделі, яка відповідає за роботу з даними.

Крім того, MVVM підтримує підхід реактивного програмування, що може бути корисним для створення динамічного та реактивного інтерфейсу користувача. Він також спрощує тестування окремих компонентів, що є важливим аспектом розробки додатку [9-11].

2.2 Підбір інструментів

2.2.1 Підбір інструмента для сканування

Не менш важливим аспектом є підібр інструмента для сканування кодів, адже користувачеві потрібно надати максимально швидкий та надійний досвід у скануванні. Одним з таких інструментів є MLKit.

ML Kit від Google - це повнофункціональний інструмент для машинного навчання, який надає широкий спектр можливостей для розпізнавання об'єктів, тексту, а також для аналізу зображень і відео. У вашому додатку для сканування кодів використання ML Kit може стати важливим компонентом, який допоможе забезпечити надійність та точність процесу сканування.

- Висока точність та надійність: ML Kit використовує передові моделі машинного навчання, які навчаються на великих обсягах даних, що дозволяє забезпечити високу точність та надійність розпізнавання кодів. Це важливо, особливо в контексті сканування кодів, де точність є критичною для коректної ідентифікації.

- Простота використання та інтеграції: ML Kit має зручний API та документацію, які допомагають швидко та легко інтегрувати його в ваш додаток. Це значно спрощує процес розробки, оскільки ви можете швидко почати використовувати функціонал ML Kit без значних зусиль.
- Розширені можливості: Окрім базового функціоналу сканування кодів, ML Kit має розширені можливості, такі як розпізнавання тексту, обробка зображень, виявлення об'єктів та інші. Це відкриває для вас широкий спектр можливостей для розширення функціоналу вашого додатку та забезпечення його конкурентоспроможності.
- Швидкість роботи: ML Kit оптимізований для швидкої обробки зображень та даних, що дозволяє забезпечити швидку реакцію додатку на сканування кодів. Це особливо важливо для забезпечення приємного користувацького досвіду та підвищення продуктивності додатку.
- Постійне оновлення та підтримка: ML Kit активно розвивається та оновлюється Google, що забезпечує вашому додатку доступ до передових технологій машинного навчання та найновіших досягнень у цій області. Це дозволяє вашому додатку залишатися сучасним та конкурентоспроможним на ринку.

Отже, використання ML Kit у нашому додатку для сканування кодів дозволить забезпечити кожному користувачу високу точність, швидкість та надійність процесу сканування, що є важливими аспектами успішного функціонування вашого додатку.

2.2.2 Підбір інструмента для інтеграції з хмарним середовищем

iCloud - це хмарна платформа від Apple, яка надає користувачам можливість зберігати, синхронізувати та резервувати їх дані на серверах Apple через Інтернет. За допомогою iCloud користувачі можуть отримати доступ до своїх даних з будь-якого пристрою, підключеного до їх облікового запису iCloud.

Ось деякі переваги використання синхронізації з iCloud для вашого додатка:

- Зручність і доступність: iCloud надає користувачам можливість отримати доступ до своїх даних з будь-якого пристрою, підключеного до їх облікового запису iCloud. Це означає, що користувачі можуть легко синхронізувати дані між своїми пристроями, наприклад, iPhone та iPad, і мати постійний доступ до оновленої інформації.
- Надійність і безпека: iCloud забезпечує надійне зберігання даних на серверах Apple з використанням сучасних методів шифрування та захисту. Це забезпечує захист конфіденційної інформації користувачів та гарантує безперебійний доступ до даних.
- Автоматична синхронізація: iCloud надає можливість автоматичної синхронізації даних між різними пристроями, що дозволяє користувачам зручно працювати зі своїми даними без необхідності ручної синхронізації.
- Широкий функціонал: iCloud пропонує широкий спектр функцій, включаючи зберігання фотографій, відео, контактів, календарів, нотаток, документів та іншого. Це робить його ідеальним вибором для синхронізації різноманітних типів даних у вашому додатку.
- Інтеграція зі сучасними технологіями: iCloud інтегрується з іншими сервісами та технологіями Apple, такими як Core Data, CloudKit та

іншими. Це робить його потужним інструментом для розробників, які хочуть створити додатки з розширеним функціоналом синхронізації та зберігання даних у хмарі.

Отже, синхронізація з iCloud є найкращим рішенням для додатку, оскільки вона забезпечує зручність, безпеку, надійність та широкий функціонал для синхронізації та зберігання даних користувачів.

2.2.3 Підбір інструмента для розробки історії пошуку

Історія пошуку є однією із найважливіших функцій та переваг нашого додатку, адже кожен користувач зможе побачити просканові ним коди та повторно використати його ще раз. Існує чимало інструментів, для її успішної реалізації. Одним із них є нативна локальна база iOS – CoreData.

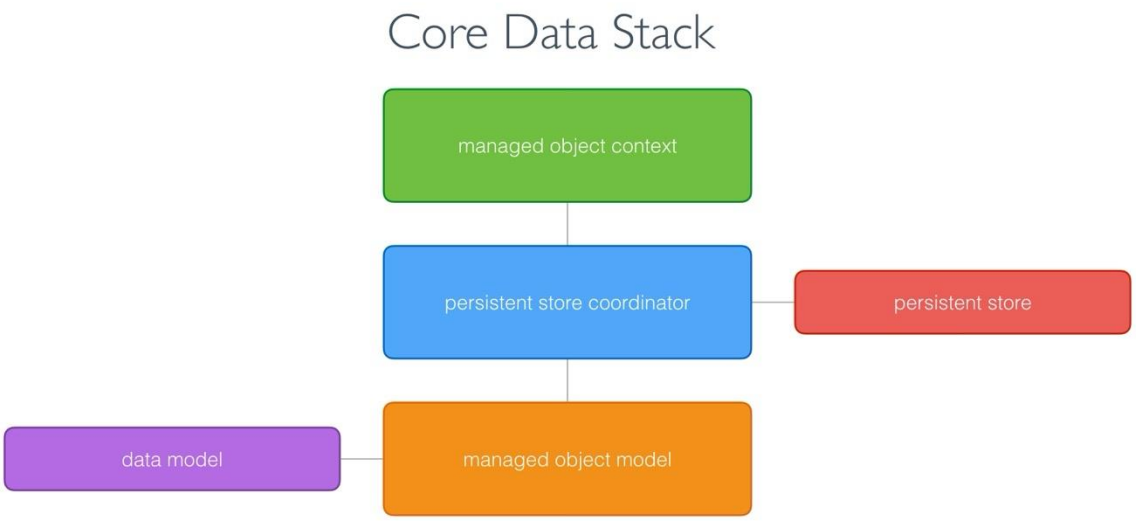


Рисунок 3.6 – «Схема роботи “Core Data”»

Інтеграція Core Data для розробки історії пошуку у вашому додатку є вибором, який принесе багато переваг та зручностей для вас як розробника, а також для користувачів вашого додатку.Коли мова йде про створення додатків, які мають зберігати та відтворювати деяку інформацію, ефективне зберігання даних - ключовий аспект. Core Data вирішує це завдання, дозволяючи зберігати складну ієрархію даних, які можуть включати різні типи об'єктів та їх відносини між собою. Він забезпечує вам можливість легко організувати ваші дані в модель, яка відповідає основним принципам структурованості та ефективності.

Ще однією перевагою використання Core Data є його легкість інтеграції з існуючим кодом. Ви можете швидко створити модель даних та налаштувати зв'язки між об'єктами без значних зусиль. Це дозволяє вам швидко розпочати розробку функціоналу історії пошуку без додаткових складнощів. Однією з ключових переваг Core Data є його можливість автоматично синхронізувати дані з iCloud. Це означає, що користувачі можуть легко синхронізувати свою історію пошуку між різними пристроями, забезпечуючи постійний доступ до оновленої інформації з будь-якого пристрою, на якому вони користуються вашим додатком. Крім того, ви можете легко використовувати функціонал Core Data для відновлення даних, що є важливим аспектом для історії пошуку. Ви можете створити резервні копії даних та відновити їх у разі випадкового видалення або втрати, що забезпечить стабільність та надійність вашого додатку.

Отже, інтеграція Core Data для розробки історії пошуку у нашому додатку - це не просто зручне рішення, але і стратегічний крок для забезпечення ефективного та надійного зберігання даних, а також для забезпечення високої якості користувацького досвіду вашим користувачам.

2.4 Висновки

Так, вибір архітектури MVVM, інтеграція штучного інтелекту через ML Kit, а також історія пошуку та синхронізація з хмарою через iCloud дійсно відіграють важливу роль у розробці додатку і надають йому унікальність та конкурентні переваги.

Архітектура MVVM (Model-View-ViewModel) є відмінним вибором для розробки мобільних додатків, оскільки вона дозволяє чітко відокремити бізнес-логіку додатку від його користувацького інтерфейсу. Це полегшує тестування, розширення та підтримку додатку в майбутньому.

Щодо інтеграції ML Kit, це надає вашому додатку здатність швидко та надійно сканувати коди зображень. Це особливо корисно для функціоналу, який потребує розпізнавання QR-кодів, штрих-кодів та інших типів кодів.

Історія пошуку та синхронізація з хмарою через iCloud - це важливий елемент вашого додатку, який надає користувачам зручність та доступність до їх даних на будь-якому пристрої, що підтримується. Це дозволяє користувачам легко переходити між пристроями та забезпечує стабільну та безперервну роботу додатку.

Загалом, комбінація цих рішень дозволяє створити додаток, який не лише забезпечує потреби користувачів, але і вирізняється на тлі конкурентів завдяки своїм унікальним функціям та перевагам.

Отже, було розгялнуто велику кількість різноманітних архітектур та обґрунтовано вибір архітектури MVVM. Також було підібрано найкращий інструмент штучного інтелекту для швидкого сканування. Не менш важливу роль, відіграватиме історія пошуку та синхронізація з хмарою, адже саме це надасть нашому додатку унікальність та перевагу над іншими додатками.

3 РОЗРОБКА І ТЕСТУВАННЯ ДОДАТКУ ТА РОЗГЛЯД ПОДАЛЬШИХ ПЕРСПЕКТИВ РОЗВИТКУ

3.1 Програмна реалізація додатку

3.1.1 Налаштування середовища для розробки

Налаштування середовища розробки в Xcode для iOS-додатків
Для розробки додатків для платформи iOS вибір середовища розробки є критично важливим кроком. Одним із найпотужніших інструментів для цієї задачі є Xcode — офіційна інтегрована середа розробки (IDE) від Apple. Xcode забезпечує розробників усіма необхідними інструментами для створення, тестування та налагодження додатків для iPhone, iPad, Mac, Apple Watch та Apple TV.

Xcode є ідеальним вибором для розробки iOS-додатків завдяки його глибокій інтеграції з екосистемою Apple та підтримці сучасних технологій розробки. Він пропонує багатий набір інструментів, таких як SwiftUI для швидкого прототипування інтерфейсу користувача, та інструменти для аналізу продуктивності. Xcode забезпечує потужний налагоджувальник (debugger) та симулятори, що дозволяють тестувати додатки на віртуальних пристроях з різними версіями iOS. Крім того, підтримка Swift — мови програмування, розробленої Apple для iOS та macOS, робить Xcode незамінним інструментом для розробників, які бажають створювати продуктивні та безпечні додатки.

Щоб розпочати роботу з Xcode, необхідно мати Mac-комп'ютер з операційною системою macOS 11 (Big Sur) або новішою. Xcode можна завантажити з App Store, що є зручним, оскільки всі оновлення також доступні через цей магазин. Після завантаження слід відкрити Xcode і дозволити йому встановити необхідні компоненти, такі як інструменти

командного рядка (Command Line Tools). Ці інструменти дозволяють компілювати і розгортувати додатки з командного рядка, що особливо корисно для досвідчених розробників.

Після встановлення Xcode важливо налаштувати середовище для ефективної розробки. Першим кроком є вхід в систему за допомогою Apple ID через Xcode. Це дає змогу використовувати сертифікати для розробки, які необхідні для тестування додатків на реальних пристроях. Наступним кроком є створення нового проекту. Xcode пропонує різноманітні шаблони для початку розробки, включаючи шаблони для простих додатків, ігор та інтеграцій з іншими платформами Apple. Вибравши шаблон "App" у розділі iOS, можна вказати ім'я проекту, організаційний ідентифікатор та вибрати мову програмування Swift.

Створивши проект, слід налаштувати цільову платформу та мінімальну версію iOS, з якою додаток буде сумісний. Xcode дозволяє обрати симулятори для тестування додатків на віртуальних пристроях, що є корисним для перевірки роботи на різних моделях iPhone та iPad без необхідності мати фізичні пристрої. Проте, для більш детального тестування та перевірки реальної продуктивності, важливо також мати можливість розгортати додаток на реальному пристрої. Підключивши iPhone або iPad до Mac за допомогою USB, можна вибрати його як цільовий пристрій для налагодження.

Для управління залежностями Xcode пропонує інтеграцію з CocoaPods та Swift Package Manager (SPM). CocoaPods є популярним менеджером залежностей, який дозволяє легко додавати сторонні бібліотеки до проекту. Встановити CocoaPods можна через командний рядок, після чого створити файл Podfile у директорії проекту і додати необхідні бібліотеки. Swift Package Manager є альтернативою, інтегрованою безпосередньо в Xcode, і забезпечує зручний спосіб додавання та

управління бібліотеками без необхідності створення додаткових файлів конфігурації.

Для контролю версій та спільної роботи над проектом використовується система Git. Ініціалізація локального репозиторію Git у директорії проекту дозволяє відстежувати зміни коду і зберігати його історію. Додавання віддаленого репозиторію, наприклад, на GitHub або GitLab, забезпечує можливість співпраці з іншими розробниками та зберігання резервних копій проекту.

Xcode пропонує потужні інструменти для тестування додатків. Використовуючи XCTest, можна створювати модульні тести, що автоматично перевіряють коректність роботи коду. Ці тести допомагають швидко виявляти помилки та перевіряти, що нові зміни не порушують роботу додатку. Для налагодження додатків Xcode надає вбудований дебагер, який дозволяє відстежувати виконання коду в реальному часі, переглядати значення змінних і використовувати точки зупину (breakpoints) для детального аналізу.

Налаштування Xcode як середовища розробки для iOS-додатків є важливим етапом для створення якісного програмного забезпечення. Xcode забезпечує всі необхідні інструменти та ресурси для ефективної розробки, тестування та налагодження додатків. Завдяки його глибокій інтеграції з екосистемою Apple, підтримці сучасних технологій та зручним інструментам для управління залежностями і версіями коду, Xcode є найкращим вибором для розробників, що працюють на платформі iOS.

3.1.2 Підключення MLKit

ML Kit від Google — це потужна платформа, яка надає широкий спектр інструментів для розпізнавання та обробки даних, таких як текст, обличчя, об'єкти, а також можливості для машинного навчання без необхідності

глибоких знань у цій галузі. Інтеграція ML Kit в iOS-додаток дозволяє розробникам легко додати функції штучного інтелекту, які значно підвищують функціональність та зручність користування. У цьому розділі ми розглянемо, як підключити ML Kit до вашого iOS-додатку, використовуючи Xcode.

ML Kit робить складні технології доступними навіть для розробників-початківців, забезпечуючи простий у використанні інтерфейс та готові моделі машинного навчання. Наприклад, для реалізації функції розпізнавання тексту або обличчя в додатку не потрібно створювати та навчати власні моделі. ML Kit пропонує набір попередньо навчених моделей, які можна легко інтегрувати у ваш проект, скорочуючи час розробки та спрощуючи процес впровадження AI-рішень.

Перед тим як розпочати інтеграцію ML Kit, необхідно підготувати Xcode та проект до роботи з бібліотеками сторонніх розробників. Зокрема, ми скористаємося CocoaPods — менеджером залежностей для iOS, який дозволяє легко додавати бібліотеки, такі як ML Kit, до нашого проекту.

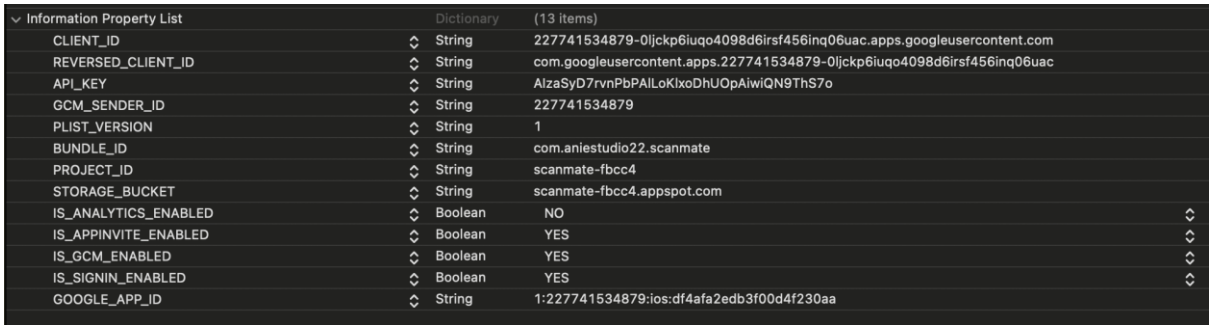
Перш за все, потрібно переконатися, що у нас встановлено CocoaPods. Якщо ні, його можна встановити за допомогою команди в Terminal:

Podfile

```
platform :ios, '16.0'
use_frameworks!

project 'ScanMate'

target 'ScanMate' do
#GooglePods
pod 'GoogleMLKit/BarcodeScanning', '3.2.0'

#SwiftLint
pod 'SwiftLint', '0.50.3'

#Firebase
pod 'Firebase/Crashlytics'
pod 'Firebase/Analytics'
pod 'Firebase/Firestore'

#CustomQrCodeGeneration
pod 'EFQRCode', '6.2.1'

#RevenueCat
pod 'RevenueCat', '4.15.0'

end
```

Рисунок 3.1 – «Встановлення CocoaPods»

ML Kit є частиною платформи Firebase, тому нам потрібно налаштувати Firebase для вашого проекту. Це включає створення облікового запису Firebase та налаштування проекту Firebase. Перейдіть на консоль Firebase, створіть новий проект та додайте до нього вашу iOS-аплікацію, завантаживши GoogleService-Info.plist файл.



Information Property List	Dictionary	(13 items)
CLIENT_ID	String	227741534879-0ljckp6iuqo4098d6irsf456inq06uac.apps.googleusercontent.com
REVERSED_CLIENT_ID	String	com.googleusercontent.apps.227741534879-0ljckp6iuqo4098d6irsf456inq06uac
API_KEY	String	AIzaSyD7rvnPbPAlLoKlxoDhUOpAiwiQN9ThS7o
GCM_SENDER_ID	String	227741534879
PLIST_VERSION	String	1
BUNDLE_ID	String	com.aniestudio22.scanmate
PROJECT_ID	String	scanmate-fbcc4
STORAGE_BUCKET	String	scanmate-fbcc4.appspot.com
IS_ANALYTICS_ENABLED	Boolean	NO
IS_APPINVITE_ENABLED	Boolean	YES
IS_GCM_ENABLED	Boolean	YES
IS_SIGNIN_ENABLED	Boolean	YES
GOOGLE_APP_ID	String	1:227741534879:ios:df4afa2edb3f00d4f230aa

Рисунок 3.2 – «Google Service Info.plist файл»

Після налаштування Firebase та підключення ML Kit до нашого проекту, можна починати використовувати його функції. Наприклад, щоб розпізнати код, імпортуємо необхідні модулі ML Kit та налаштуємо обробку зображення.

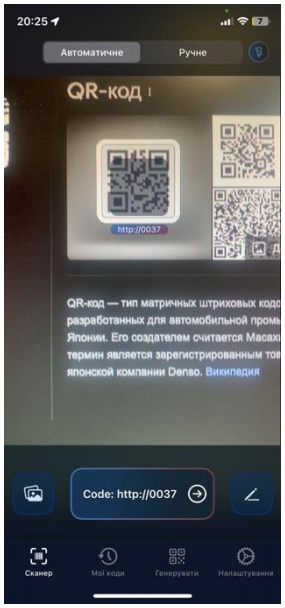


Рисунок 3.3 – «Знімок екрана сканування»

Знімок екрана демонструє, що додаток розпізнає коди, отже, підключення MLKit було вірним.

3.1.3 Розробка функції генерації

Apple надає нам можливість використати нативну бібліотеку CoreImage, яка надасть усі інструменти для генерації кодів. CoreImage є потужною фреймворком Apple, що забезпечує інструменти для обробки та аналізу зображень. Введена в iOS ще з версії 5.0, CoreImage дозволяє розробникам створювати та застосовувати різноманітні фільтри для обробки зображень, включаючи ефекти, корекцію кольору та, що важливо для нашого випадку, генерацію QR-кодів. Фреймворк широко використовується у фотографії, графічному дизайні та додатках для редагування зображень, завдяки своїй ефективності та гнучкості.

CoreImage використовує об'єкти, звані CIFilter, для застосування ефектів до зображень. CIFilter може приймати одне або більше зображень як вхідні дані і обробляти їх відповідно до заданих параметрів, щоб створити нове зображення як результат. CoreImage підтримує більше 100 різних фільтрів, що дозволяє реалізовувати різноманітні завдання обробки зображень, такі як розмиття, посилення кольору, викривлення та навіть детекція обличчя. Кожен фільтр має певний набір вхідних параметрів, які можна налаштовувати, щоб досягти бажаного ефекту.

Однією з особливостей CoreImage є його архітектура, яка використовує принципи відкладеної обробки (lazy evaluation). Це означає, що фільтри не застосовуються негайно, коли ви їх налаштовуєте. Натомість, CoreImage будує граф обробки зображення, який виконується лише тоді, коли ви фактично запитуєте вихідне зображення. Це дозволяє ефективно керувати обробкою зображень, мінімізуючи витрати ресурсів і забезпечуючи високу продуктивність.

CoreImage не тільки відома своїми фільтрами для покращення та корекції зображень, але й пропонує інструменти для генерації спеціалізованих зображень, таких як QR-коди. За допомогою CIFilter.qrCodeGenerator() можна легко створити QR-код з будь-якого текстового вмісту. Цей фільтр перетворює вхідні дані (стрічку тексту) у зображення QR-коду, яке можна додатково налаштовувати та масштабувати.

Процес генерації QR-коду через CoreImage включає створення CIFilter.qrCodeGenerator(), встановлення вхідного повідомлення у форматі даних ASCII та масштабування результату для отримання зображення в потрібному розмірі. Результат — це високоякісне зображення QR-коду, яке може бути легко інтегроване в користувацький інтерфейс додатку.

Використання CoreImage у вашому додатку відкриває широкі можливості для обробки та аналізу зображень. Ви можете створювати фільтри для покращення фотографій, застосовувати ефекти реального часу в додатках для камери або створювати генератори QR-кодів, які можуть бути використані для широкого спектру застосувань, таких як маркетинг, управління даними або інтерактивні послуги. Завдяки своїй потужності та гнучкості, CoreImage є важливим інструментом для розробників iOS, які прагнуть додати функціональні можливості обробки зображень до своїх додатків.

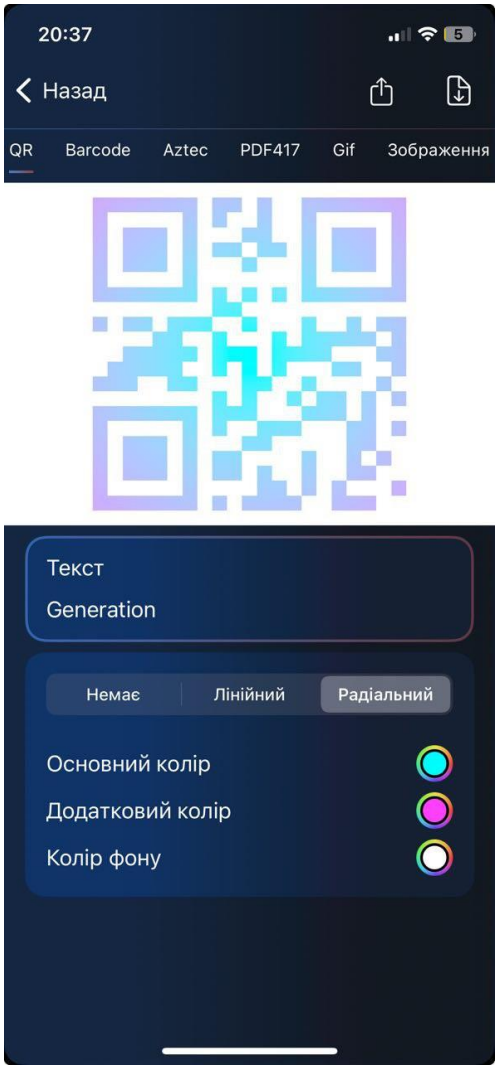


Рисунок 3.4 – «Демонстрація роботи генерації»

На рисунку видно, що додаток вдало генерує код, отже, підключення CoreImage пройшло успішно.

3.1.4 Розробка функції історії пошуку

Як говорилось вище, для історії пошуку використовуватимемо CoreData. Для цього потрібно створити файл ScanMate.xcdatamodeld.
У цьому файлі необхідно описати усі сутності.

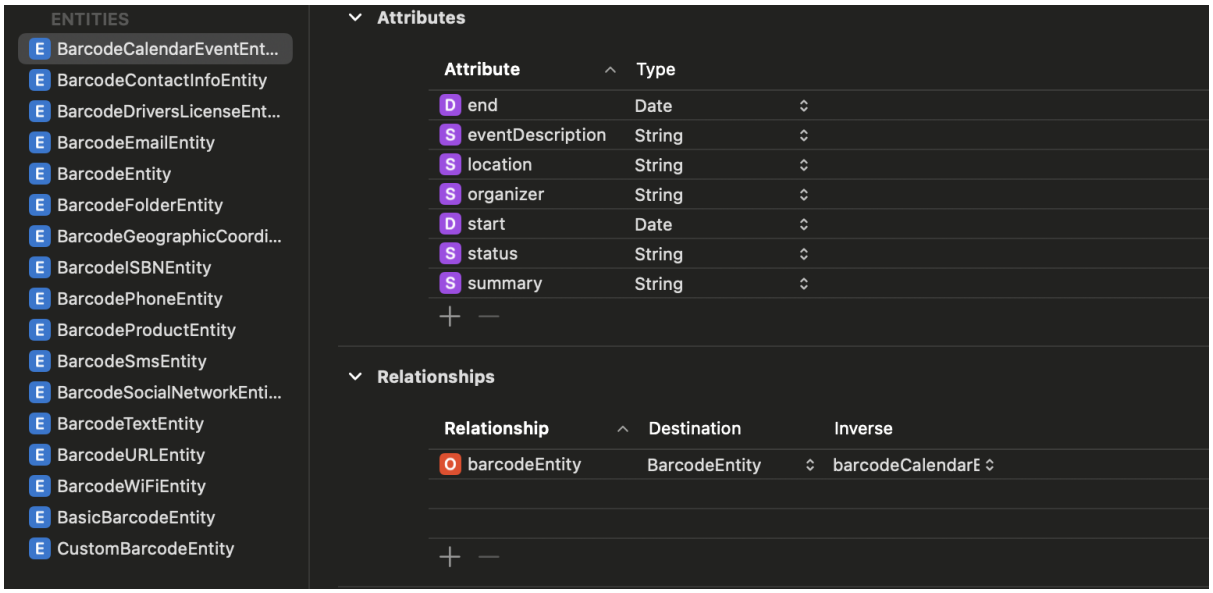


Рисунок 3.5 – «Сутності, необхідні для історії пошуку»

Тепер напишемо клас, який буде додавати коди у локальне сховище.

```
import FirebaseFirestore
import Combine

final class FireStoreRemoteRepositoryImpl: FireStoreRemoteRepository {

    typealias Identifier = String

    private lazy var fireStore = Firestore.firestore()

    func fetchPromoCode(_ codeBody: String) -> AnyPublisher<PromoCode?, Never> {
        return Future<PromoCode?, Never> { [weak self] promise in
            guard let self = self else { return }

            self.fireStore.collection("promoCodes").whereField("codeBody", isEqualTo: codeBody).getDocuments { snapshot, error in
                if error != nil {
                    promise(.success(nil))
                } else {
                    guard
                        let document = snapshot?.documents.first,
                        let promoCode = PromoCode(dictionary: document.data())
                    else {
                        promise(.success(nil))
                        return
                    }
                    promise(.success(promoCode))
                }
            }
        }.eraseToAnyPublisher()
    }

    func deletePromoCode(_ codeBody: String) {
        fireStore.collection("promoCodes").whereField("codeBody", isEqualTo: codeBody).getDocuments { [weak self] snapshot, _ in
            guard let self,
                  let document = snapshot?.documents.first
            else { return }
            self.fireStore.collection("promoCodes").document(document.documentID).delete { _ in }
        }
    }

}
```

Рисунок 3.6 – «Клас “FireStoreRemoteRepository”»

Спробуємо зберегти кілька кодів до історії пошуку, щоб перевірити, чи працює вона

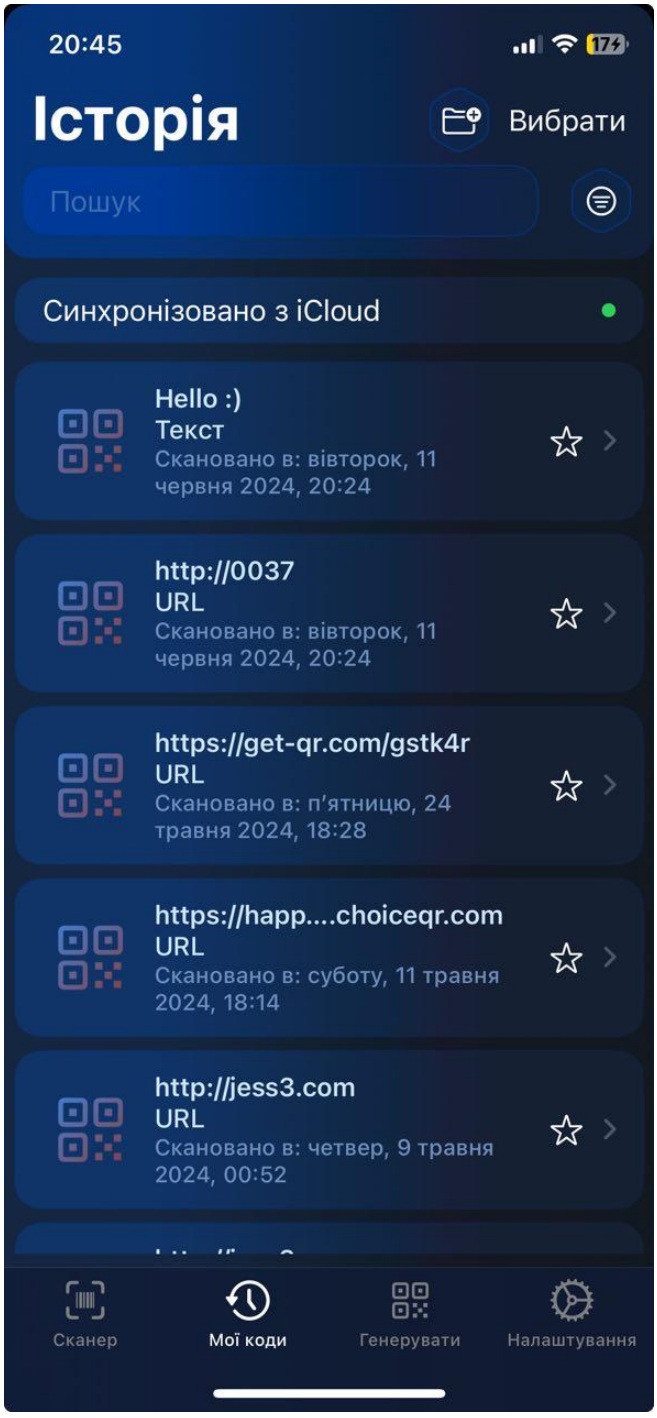


Рисунок 3.7 – «Демонстрація роботи історії пошуку»

3.1.5 Розробка функції налаштувань

Налаштування є важливими для будь якої програми, адже додаток має бути зручним для кожного. Користувачеві необхідно надати можливіть:

- Увімкнути/вимкнути синхронізацію з iCloud
- Увімкнути/вимкнути групове сканування
- Увімкнути/вимкнути автоматичне сканування
- Увімкнути/вимкнути автоматичне збереження
- Увімкнути/вимкнути вібрацію
- Увімкнути/вимкнути автоматичний спалах
- Очистити історію пошуку

```
var body: some View {
    GeometryReader { screen in
        ZStack {
            BackgroundImageView()

            List {
                if !viewModel.isProVersionUnlocked {
                    constructSectionView(with: .subBanner(viewModels: viewModel.constructSubscriptionBannerCellViewModel()))
                        .listRowInsets(EdgeInsets())
                        .listRowBackground(Color.clear)
                }
                constructSectionView(with: .iCloudSync(viewModels: viewModel.constructiCloudSyncCellViewModel()),
                                     sectionTitle: "storage",
                                     footerTitle: NSLocalizedString(Constants.iCloudSyncFooterTitleKey, comment: ""),
                                     footerNote: NSLocalizedString(Constants.iCloudSyncFooterNoteTitleKey, comment: ""))
                constructSectionView(with: .general(viewModels: viewModel.constructGeneralCellViewModels()), sectionTitle: "general")
                constructSectionView(with: .feedback(viewModels: viewModel.constructFeedbackCellViewModels()), sectionTitle: "support_and_feedback")
                constructSectionView(with: .delete(viewModels: viewModel.constructDeleteCellViewModels()), sectionTitle: "data_managemant")
                constructSectionView(with: .privacy(viewModels: viewModel.constructPrivacyCellViewModels()), sectionTitle: "terms_of_use" )
                constructSectionView(with: .promoCode(viewModels: viewModel.constructPromoCodeCellViewModels()), sectionTitle: "promocodes",
                                     footerTitle: NSLocalizedString("promocode_cell_footer_key", comment: ""))
            }
        }
        .overlay {
            if shouldShowLoader {
                ZStack {
                    Rectangle()
                        .foregroundColor(.clear)
                        .frame(width: screen.size.width * 0.2,
                               height: screen.size.width * 0.2)
                        .overlay(.ultraThinMaterial)
                        .cornerRadius(20)

                    VStack {
                        ProgressView()
                            .tint(.white)
                            .scaleEffect(2)
                            .progressViewStyle(CircularProgressViewStyle())
                    }
                }
```

Рисунок 3.8 – «Проектування екрану налаштувань»

Отже, коли основний функціонал додатку реалізовано, можна перейти до тестування.

3.2 Тестування додатку

Найкращим вибором платформи для тестування є TestFlight – нативна платформа від Apple, яка підтримує зворотній зв’язок та має легке налаштування.

TestFlight є офіційною платформою Apple для бета-тестування iOS-додатків, забезпечуючи розробників ефективним і зручним засобом для випробовування своїх продуктів перед їх офіційним релізом у App Store. За допомогою TestFlight можна легко поширювати бета-версії додатків серед тестерів, збирати зворотний зв'язок, відстежувати помилки та покращувати продуктивність додатка. Це дозволяє не тільки виявляти проблеми та покращувати користувацький досвід, але й готувати додаток до стабільного та успішного запуску. TestFlight надає багатий набір функцій, які роблять процес бета-тестування інтуїтивним і ефективним. Однією з головних переваг платформи є можливість швидкого розповсюдження бета-версій додатків серед широкої аудиторії. Розробники можуть запрошувати як внутрішніх тестерів (членів команди), так і зовнішніх користувачів, що дозволяє отримати цінний зворотний зв'язок від реальних користувачів з різними пристроями та конфігураціями. Ця функція є критично важливою для ідентифікації та усунення багатьох проблем, які можуть виникати тільки в умовах реального використання. Ще однією значущою перевагою TestFlight є вбудована функція збору зворотного зв'язку. Тестери можуть надсилати коментарі та звіти про помилки безпосередньо з додатка, що спрощує процес обробки зворотного зв'язку і дозволяє розробникам швидко реагувати на проблеми. Окрім цього, TestFlight автоматично збирає дані про аварії додатків і моніторить їх продуктивність, що допомагає розробникам аналізувати та оптимізувати додаток. Це може бути особливо корисним для виявлення та виправлення критичних помилок перед запуском. Щоб забезпечити зручність для тестерів, TestFlight підтримує автоматичні

оновлення бета-версій додатків. Коли нова версія стає доступною, тестери отримують повідомлення і можуть легко встановити оновлення. Це значно полегшує процес тестування нових функцій або виправлень, забезпечуючи безперервний зворотний зв'язок протягом усього періоду бета-тестування.

Інтеграція TestFlight у процес розробки додатків надає численні переваги, які роблять платформу незамінною для ефективного бета-тестування. Однією з ключових переваг є широке охоплення тестерів, яке підтримує до 10,000 зовнішніх тестерів на додаток. Це дозволяє залучити велику кількість користувачів для тестування та отримати більш репрезентативний зворотний зв'язок про додаток.

Інтеграція з екосистемою Apple є ще однією важливою перевагою. TestFlight легко взаємодіє з іншими інструментами Apple, такими як Xcode і App Store Connect, що значно спрощує процес завантаження та управління бета-версіями додатків. Ця інтеграція дозволяє безперешкодно завантажувати нові збірки, керувати тестерами та аналізувати результати тестування з однієї платформи.

Для тестерів TestFlight забезпечує зручний спосіб встановлення та тестування додатків. Тестери можуть легко встановлювати бета-версії через додаток TestFlight на своїх пристроях, що підвищує їх залучення до процесу тестування. Крім того, можливість надсилання зворотного зв'язку безпосередньо з додатку робить процес комунікації простим та ефективним.

Безпека та конфіденційність також є ключовими аспектами, які забезпечує TestFlight. Платформа гарантує, що тільки авторизовані користувачі отримають доступ до бета-версій додатків, що особливо важливо для захисту конфіденційних даних і нових функцій, які ще не були офіційно випущені. Це дає розробникам впевненість у тому, що їхній продукт буде захищений на всіх етапах бета-тестування.

Налаштування TestFlight починається з завантаження додатку на платформу App Store Connect. Це включає створення архіву додатка у Xcode

та його завантаження в App Store Connect, де можна додати інформацію про нову бета-версію, включаючи описи змін та інструкції для тестерів. Після цього можна додати тестерів — як внутрішніх, так і зовнішніх. Зовнішні тестери повинні пройти додаткову перевірку Apple перед тим, як отримати доступ до бета-версії.

Після додавання тестерів можна розіслати їм запрошення через електронну пошту або посилання, що дозволяє їм легко встановити додаток через додаток TestFlight на своїх пристроях. Тестери будуть отримувати сповіщення про нові версії додатка та зможуть автоматично оновлювати його, що значно спрощує процес тестування нових функцій або виправлень. Таким чином, TestFlight забезпечує ефективний і зручний спосіб проведення бета-тестування, що дозволяє розробникам підготувати свої додатки до успішного запуску.

TestFlight є незамінним інструментом для розробників iOS, які прагнуть забезпечити високу якість своїх додатків перед офіційним випуском. Завдяки своєму широкому функціоналу, інтеграції з екосистемою Apple, зручності використання для тестерів та забезпеченню безпеки, TestFlight допомагає розробникам виявляти та виправляти проблеми, отримувати цінний зворотний зв'язок і покращувати загальний досвід користування додатком. Використання TestFlight у процесі розробки гарантує, що ваш додаток буде готовий до успішного запуску в App Store.

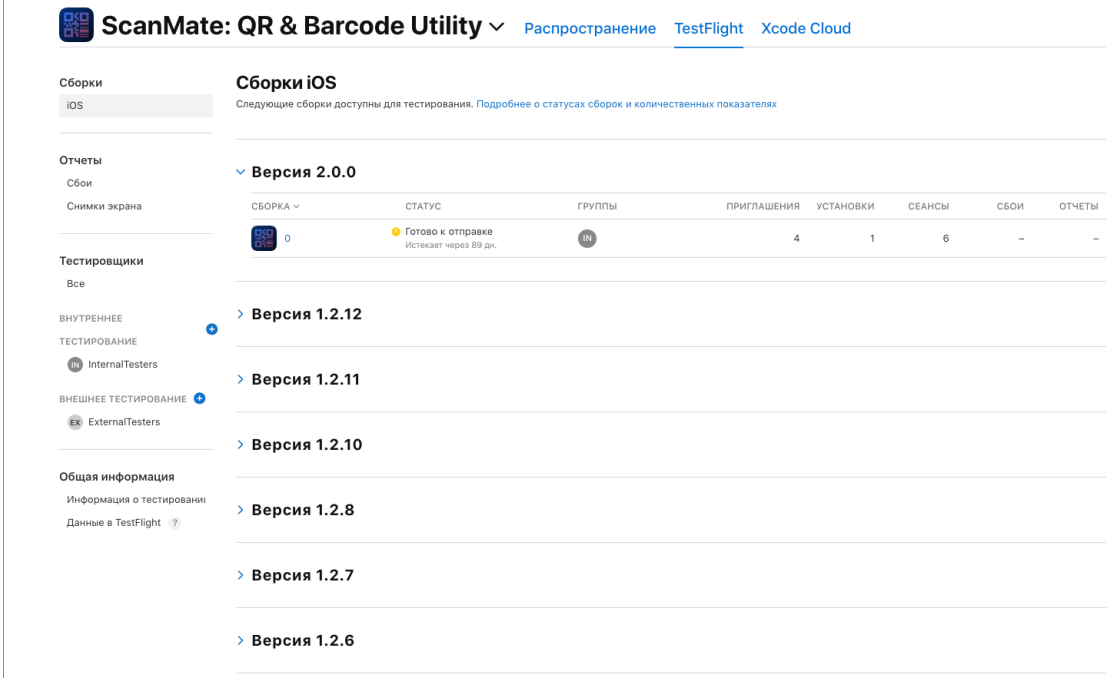


Рисунок 3.9 – «Платформа AppStoreConnect»

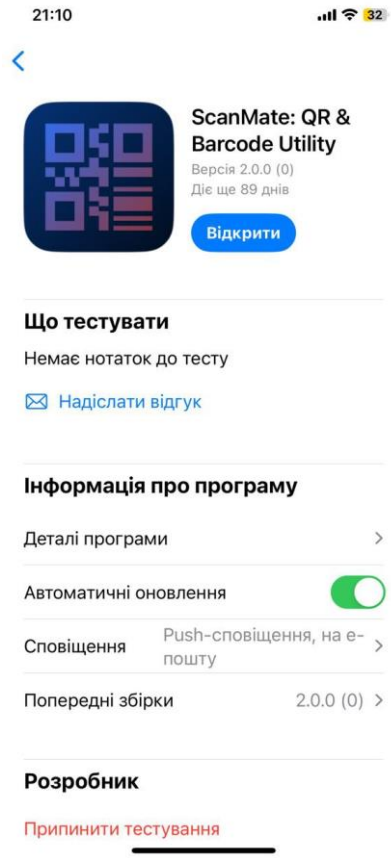


Рисунок 3.10 – «Тестова версія додатку на TestFlight»

3.3 Публікація додатку в AppStore

Після проведення ретельного бета-тестування ваш додаток через TestFlight, отримання зворотного зв’язку від тестерів і усунення виявлених помилок, ви, нарешті, наближаєтесь до одного з найважливіших етапів у процесі розробки – публікації додатку в App Store. Це важливий момент, який вимагає обдуманого підходу та уваги до деталей, щоб забезпечити успішний запуск. Завершення тестування та аналіз результатів Перш ніж перейти до публікації, важливо переконатися, що всі основні завдання тестування завершені. Завдяки TestFlight, ви змогли провести ефективне бета-тестування вашого додатку, під час якого тестери допомогли виявити потенційні проблеми та надіслати цінний зворотний зв’язок. Ви переглянули всі звіти про аварії, оцінки продуктивності та коментарі від тестерів, що дозволило виявити слабкі місця додатку. Завдяки цьому, усі критичні баги були виправлені, продуктивність оптимізована, а користувацький досвід вдосконалений. Переконайтесь, що всі зміни перевірені, та додаток працює стабільно на різних пристроях та версіях iOS.

На цьому етапі корисно провести останній контрольний огляд, щоб переконатися, що всі необхідні функції працюють належним чином. Це включає перевірку навігації, працездатності всіх кнопок і функцій, а також відповідність додатку очікуванням користувачів.

Після того, як ви переконалися, що додаток готовий до публікації, настає час підготувати його для розміщення в App Store. Спочатку необхідно створити обліковий запис розробника Apple, якщо ви ще цього не зробили. Наступним кроком є заповнення всіх необхідних даних про додаток у App Store Connect. Це включає основну інформацію про додаток, таку як ім'я, опис, ключові слова, категорії та контактну інформацію. Окрім цього, вам знадобиться підготувати і завантажити скріншоти та іконку додатку, які будуть відображатися у App Store. Важливо, щоб усі ці елементи були

якісними та відповідали вимогам Apple, оскільки вони допоможуть залучити увагу користувачів.

Також важливо визначити стратегію ціноутворення для вашого додатку. Ви можете обрати модель безкоштовного додатку, додатку з підпискою або встановити фіксовану ціну. Вибір правильної моделі залежить від вашої аудиторії та цінності, яку ваш додаток приносить користувачам. Крім того, якщо у вашому додатку є покупки всередині додатку (in-app purchases), переконайтеся, що вони правильно налаштовані та описані.

Коли всі дані заповнені і додаток повністю готовий, ви можете надіслати його на рецензію в Apple. Цей процес включає завантаження додатку на сервери Apple і очікування перевірки його відповідності політикам App Store. Зазвичай, перевірка триває кілька днів, але може зайняти більше часу, якщо Apple виявить необхідність додаткового огляду.

Перед надсиланням переконайтесь, що ваш додаток відповідає всім правилам і рекомендаціям Apple. Це включає відповідність додатку політикам конфіденційності, забезпечення коректної роботи на всіх підтримуваних пристроях і відсутність неприйнятного контенту. У процесі рецензії Apple може запитати додаткову інформацію або внести корективи, тому будьте готові до можливих запитів.

Після успішного проходження рецензії ваш додаток буде готовий до публікації. Ви можете вибрати, чи публікувати його негайно або запланувати випуск на певну дату. Важливо продумати маркетингову стратегію, щоб забезпечити максимальне охоплення аудиторії під час запуску. Розгляньте можливість використання соціальних мереж, розсилки новин та інших каналів комунікації для просування вашого додатку.

Випуск додатку в App Store є ключовим кроком у процесі розробки, який відкриває нові можливості для взаємодії з користувачами та розширення вашої аудиторії. Після запуску важливо продовжувати підтримувати

додаток, випускати оновлення, враховувати відгуки користувачів і постійно працювати над покращенням його якості.

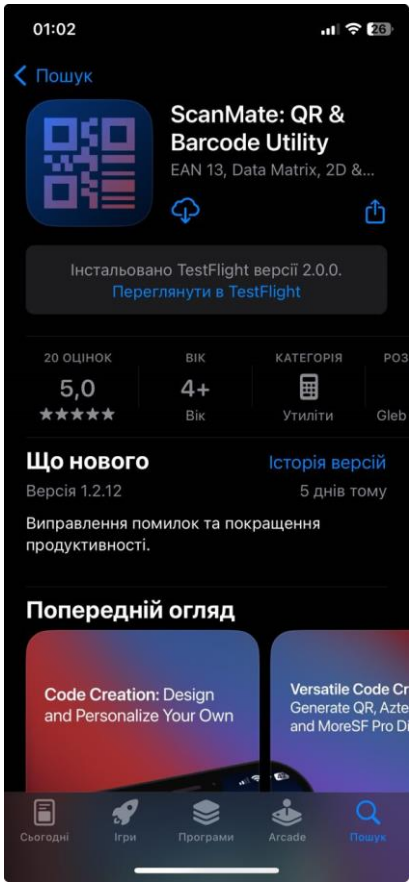


Рисунок 3.11 – «Опублікований додаток у AppStore»

3.4 Перспективи подальшого розвитку

Після успішного запуску додатку настає новий етап, який включає в себе не тільки підтримку його стабільної роботи, але й активний розвиток і вдосконалення функціональності. З метою збереження конкурентоспроможності та постійного підвищення задоволеності користувачів, важливо впроваджувати нові стратегії та підходи до оптимізації продукту. У цьому контексті особливе місце займають A/B

тестування та маркетинговий аналіз, які можуть суттєво вплинути на успіх вашого додатку в довгостроковій перспективі.

A/B тестування є потужним інструментом для аналізу та оптимізації додатку. Воно дозволяє тестувати різні варіанти інтерфейсу, функцій або контенту, щоб визначити, який з них працює найкраще. Ідея полягає у тому, що користувачам випадковим чином показуються різні версії додатку, після чого аналізуються їхні дії та реакції. Наприклад, можна тестувати різні варіанти дизайну кнопок, кольорові схеми або розташування елементів на екрані, щоб зрозуміти, який варіант найбільше сприяє збільшенню конверсій або покращенню користувацького досвіду.

Впровадження A/B тестування у ваш додаток дозволить вам робити більш обґрунтовані рішення щодо змін і оновлень. Це допомагає уникнути ризику введення змін, які можуть негативно вплинути на користувачів, і натомість базувати ваші оновлення на реальних даних і поведінці користувачів. A/B тестування також може бути корисним для тестування нових функцій або контенту перед їх повним впровадженням. Це дозволяє отримати зворотний зв'язок і виправити будь-які проблеми ще на ранньому етапі.

Щоб забезпечити стійке зростання вашого додатку, необхідно постійно проводити маркетинговий аналіз та оновлювати рекламні кампанії. Ринок мобільних додатків є надзвичайно конкурентним, і без активного маркетингового підходу ваш додаток може загубитися серед тисяч інших пропозицій. Маркетинговий аналіз дозволяє краще зрозуміти вашу цільову аудиторію, її потреби та поведінку, а також оцінити ефективність ваших поточних маркетингових зусиль.

Роблячи маркетинговий аналіз, важливо використовувати різноманітні джерела даних, такі як аналітика додатків, соціальні мережі та ринкові дослідження. Це допоможе вам визначити, які аспекти вашого додатку привертають найбільше уваги, які функції користуються популярністю і які

частини вашої стратегії потребують покращення. На основі цих даних ви можете коригувати свої маркетингові кампанії, спрямовуючи їх на найбільш перспективні сегменти ринку і підвищуючи ефективність своїх рекламних зусиль.

Регулярне оновлення реклами також є ключовим аспектом успішної маркетингової стратегії. Це включає адаптацію вашого рекламного контенту до змін у ринку та поведінці користувачів, а також використання нових форматів і каналів для залучення аудиторії. Наприклад, ви можете використовувати відеорекламу, рекламу в соціальних мережах або партнерства з впливовими особами, щоб підвищити видимість вашого додатку. Важливо також тестувати різні рекламні стратегії та креативи через A/B тестування, щоб знайти найбільш ефективні підходи для залучення та утримання користувачів.

З метою підтримки динамічного розвитку додатку, важливо поєднувати технічні вдосконалення з активними маркетинговими зусиллями. Інтеграція A/B тестування та проведення регулярного маркетингового аналізу дозволять вам ефективно реагувати на потреби користувачів і ринкові зміни, забезпечуючи постійне покращення користувацького досвіду та зростання вашої аудиторії. Ці підходи сприяють не тільки збереженню лояльності існуючих користувачів, але й допомагають привернути нових, що є ключовим фактором для успішного розвитку вашого додатку у довгостроковій перспективі.

3.5 Висновки

У даному розділі було виконано програмну реалізацію додатку універсального сканера. Підібрана архітектура MVVM дала можливіть успішно розширювати функціонал додатку за рахунок своєї гнучкості. Після розробки, додаток успішно пройшов тестування на платформі

TestFlight, де можна було побачити його реальну швидкість та простоту у використанні.

Було опубліковано додаток в AppStore і тепер кожен бажаючий, може завантажити його з відкритого доступу. Також було розглянуто перспективи подальшого розвитку, а саме – інтеграцію AB тестування, що дозволить аналізувати додаток та визначити, які саме елементи додатку є найбільш популярними.

ВИСНОВКИ

Отже, в даній роботі було розглянуто важливість розвитку технологій сканування кодів та їх інтеграцію в сучасні мобільні додатки як ключовий етап у вдосконаленні інформаційних систем. Починаючи з винаходу штрих-коду у 1948 році і до появи QR-кодів, ці технології пройшли великий шлях, спрямований на поліпшення доступу та обробки інформації. Сьогодні мобільні додатки для сканування та генерування кодів стали невід'ємною частиною багатьох аспектів бізнесу та повсякденного життя.

У моїй роботі було розроблено та успішно реалізувано універсальний сканер кодів з декількома ключовими функціями. Перш за все, ми забезпечили підтримку різних типів кодів, що робить наш додаток універсальним та зручним для користувачів. Крім того, ми додали функціонал генерації кодів, що дозволяє користувачам створювати власні коди для обміну інформацією та використання в бізнесі. Інтеграція з хмарними сервісами, зокрема з iCloud, дозволяє зберігати і синхронізувати дані користувачів, забезпечуючи їм постійний доступ до історії та налаштувань навіть на різних пристроях.

Було вибрано архітектуру MVVM для реалізації нашого додатку, що дозволило чітко відокремити бізнес-логіку від користувацького інтерфейсу і спростило його розширення та підтримку. Інтеграція ML Kit надала нам можливість швидко та надійно сканувати різні типи кодів, що зробило наш додаток більш універсальним та корисним для користувачів. Синхронізація з хмарними сервісами через iCloud надала зручність та доступність до даних на будь-якому пристрої, що підтримується, підвищуючи таким чином користувацький досвід.

Наша робота завершилася успішним тестуванням додатку і його публікацією в App Store. Ми також розглянули можливості подальшого

розвитку, такі як інтеграція AB тестування для аналізу користувацької взаємодії та популярності різних функцій.

Отже, у результаті цієї роботи ми продемонстрували важливість та переваги розвитку технологій сканування кодів та їх інтеграцію в сучасні мобільні додатки. Наш додаток є прекрасним прикладом використання новітніх технологій для створення зручних та корисних інструментів для користувачів .

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кмитюк Д. С., Арсенюк І. Р. Обґрунтування доцільності розробки додатку універсального сканера/генератора кодів з історією пошуку та синхронізацією через iCloud. URL: http://www.konferenciaonline.org.ua/ua/article/id-1814/ (дата звернення 08.06.2024)
2. Класифікація та типи кодів. URL: https://www.scandit.com/resources/guides/types-of-barcodes-choosing-the-right-barcode/ (дата звернення 08.06.2024)
3. Документація середовища Core Data. URL: https://developer.apple.com/documentation/coredata/ (дата звернення 08.06.2024)
4. Як працювати з Core Data. URL: https://dou.ua/forums/topic/38832/ (дата звернення 08.06.2024)
5. iOS Architecture - A Comprehensive Guide. URL: https://intellipaat.com/blog/tutorial/ios-tutorial/ios-architecture/#:~:text=The%20architecture%20of%20IOS%20is,of%20well%2Ddefined%20system (дата звернення 08.06.2024)
6. Migration for iOS. URL: https://developers.google.com/ml-kit/migration/ios?hl (дата звернення 08.06.2024)
7. iCloud Drive setup. URL: https://support.apple.com/ru-ru/guide/icloud/mm203b05aec8/icloud (дата звернення 08.06.2024)

8. App Store Connect Help. URL:
https://developer.apple.com/help/app-store-connect/
(дата звернення 08.06.2024)
9. Beta testing made simple with TestFlight. URL:
https://developer.apple.com/testflight/
(дата звернення 08.06.2024)
10. App Magic. URL:
https://appmagic.rocks/ (дата звернення 08.06.2024)
(дата звернення 31.05.2024)
11. Firebase – що таке та як використовувати? URL:
https://firebase.google.com/docs/ios/setup?hl
(дата звернення 02.06.2024)
12. Xcode – Apple Developer Documentation. URL:
https://developer.apple.com/documentation/xcode
(дата звернення 05.06.2024)
13. Аналітика додатків Firebase. URL:
https://firebase.google.com/docs/crashlytics?hl
(дата звернення 78.06.2024)
14. App Store Optimisation Guide. URL:
https://www.semrush.com/blog/app-store-optimization/
(дата звернення 06.06.2024)

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка універсального додатку сканера/генератора кодів з історією пошуку та синхронізацією через iCloud

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,5% Схожість 2,5%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Кмитюк Д.С.

Керівник роботи Арсенюк І.Р

ДОДАТОК Б ІНСТРУКЦІЯ КОРИСТУВАЧА

Під час запуску, користувач потрапляє на вкладку сканування. Користувачу доступна навігаційна панель, де він можна обрати, на який екран користувач бажає потрапити.

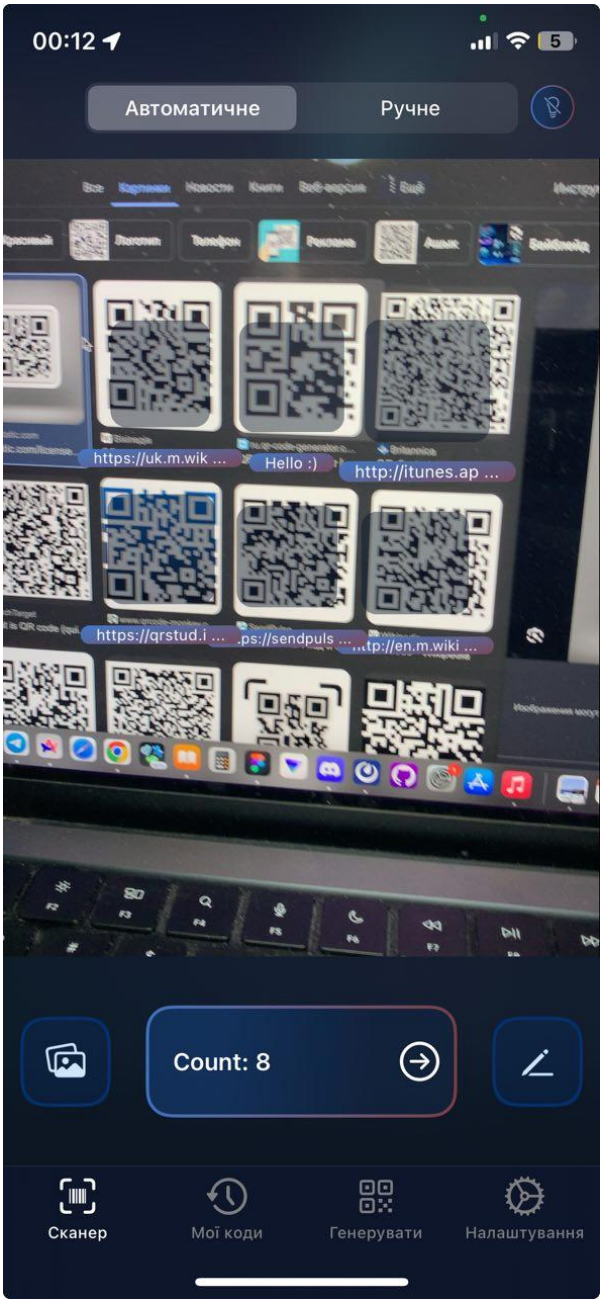


Рисунок Б.1 – «Демонстрація функції сканування»

Наступна вкладка – історія пошуку, де користвач поже переглянути усі відскановані, або згенеровані фото. Доступний фільтр та можливість відмітити будь який код, як вибраний.

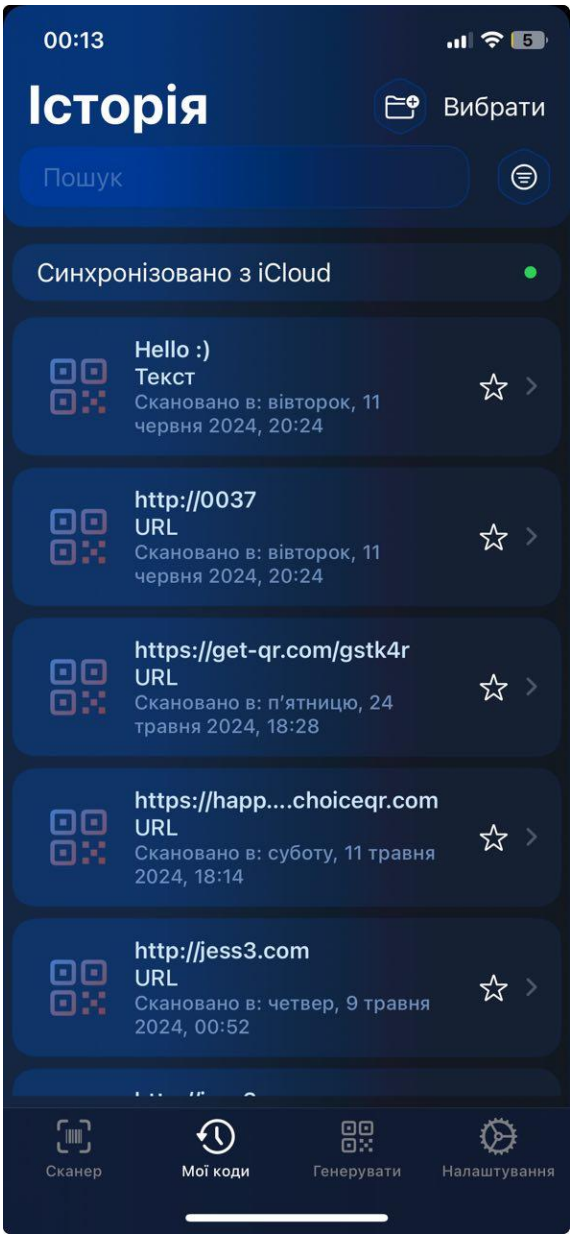


Рисунок Б2 «Демонстрація історії пошуку»

Доступний перехід на детальний екран, де можна переглянути детальну інформацію про код.

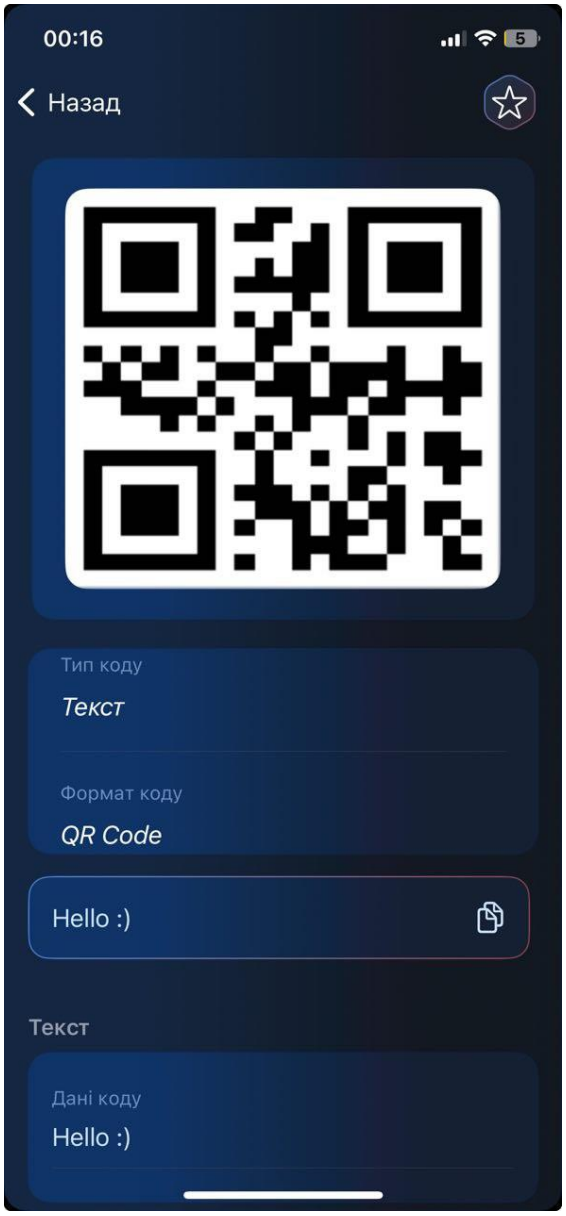


Рисунок Б3 – «Демонстрація історії пошуку»

Ще один із ключових екранів – екран генерації, де користувачеві дається можливість обрати тип інформації, яку він хоче зашифрувати, у вигляді коду

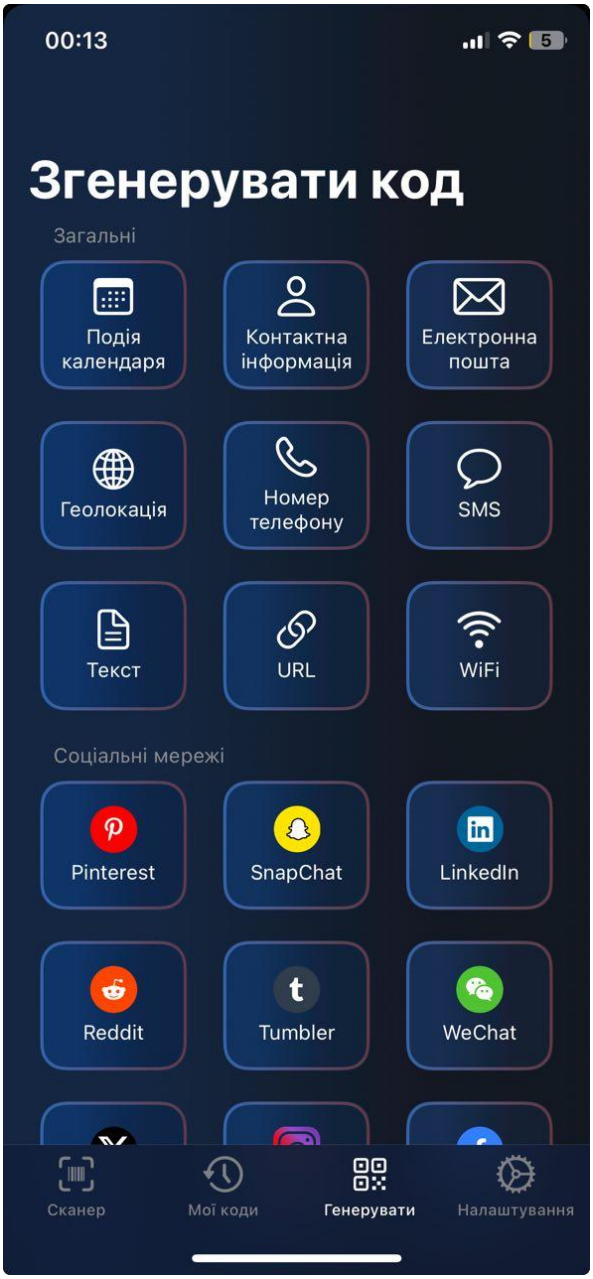


Рисунок Б3 – «Екарн генерації»

При генеруванні, користувач може бачити, як генерується його код у режимі реального часу. Також можна обрати, який із типів кодів буде згенеровано. Крім цього, можна відправити код, у вигляді повідомлення.

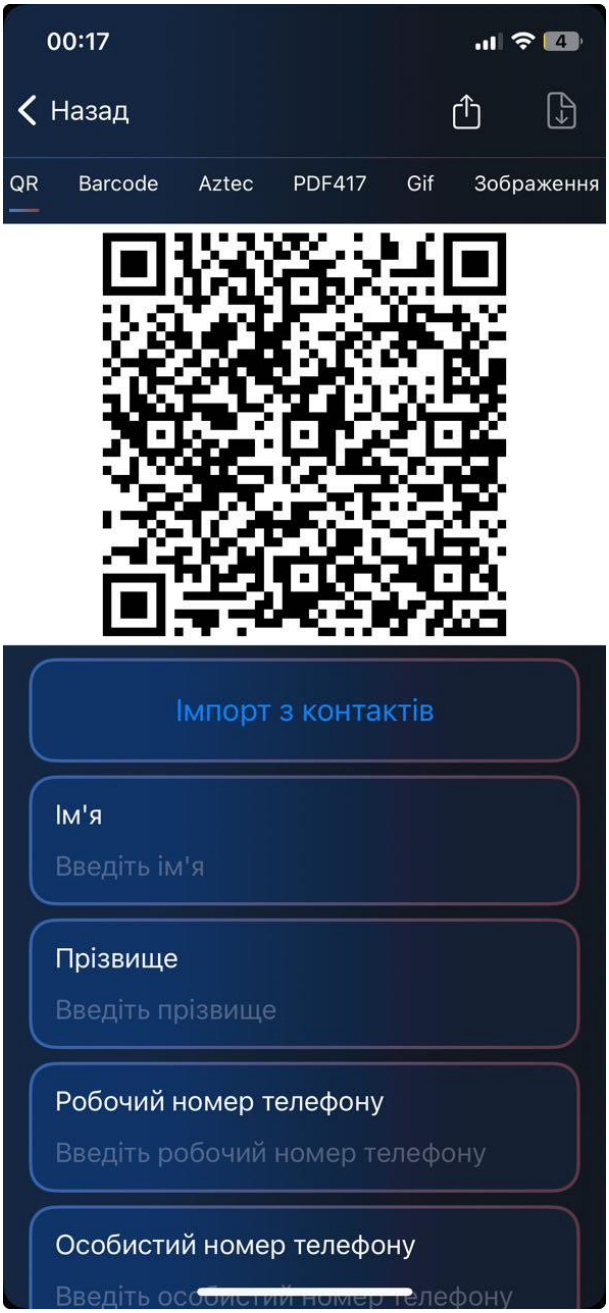


Рисунок Б5 – «Генерація коду»

Останній, із ключових екранів – екран налаштування, де користувач може кастомізувати додаток під свої потреби.

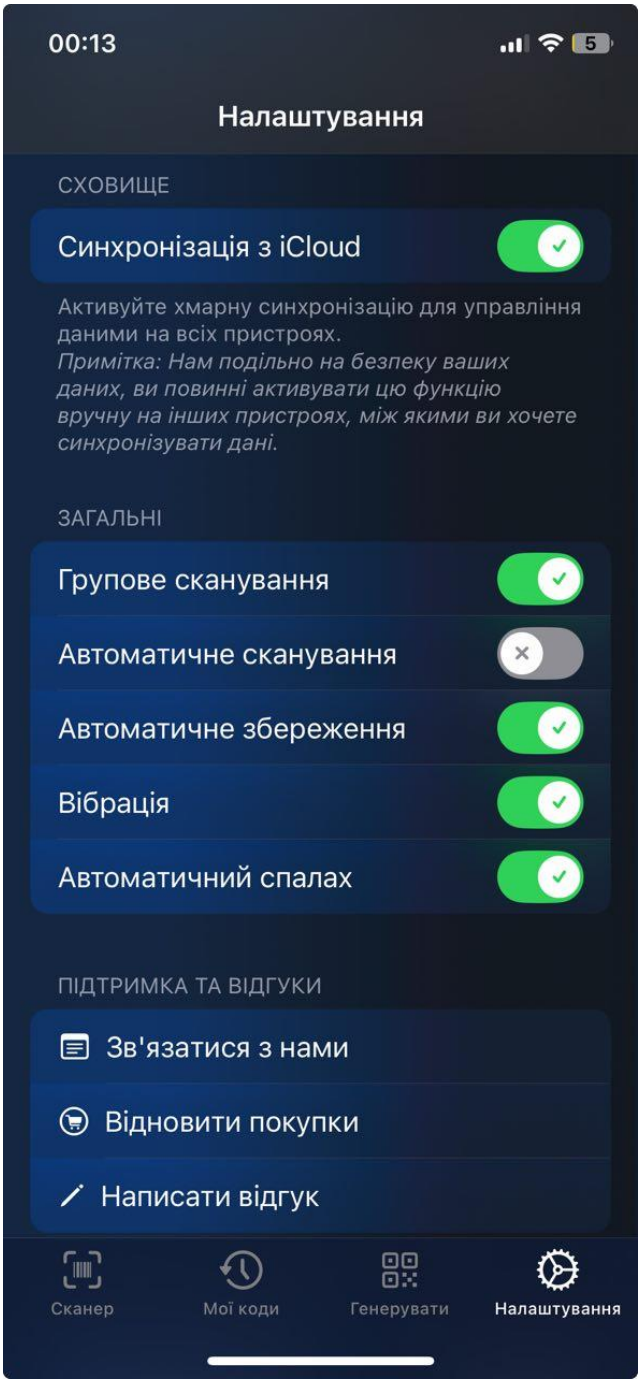


Рисунок Б6 – «Екарн налаштувань»

ДОДАТОК В ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
import SwiftUI

struct RootView: View {

    @State private var selectedTab: Int = 0
    @State private var shouldShowUpdatesScreen = false

    @StateObject var tabViewState = TabViewState()
    @StateObject var viewModel: RootViewModel

    @FocusState var isInputActive: Bool

    var body: some View {
        GeometryReader { screen in
            NavigationStack {
                ZStack {
                    TabView(selection: $selectedTab) {
                        ScannerView(viewModel:                    .init(container:
viewModel.fetchContainer(), screenHeight: screen.size.height))
                        .tabItem {
                            Label(NSLocalizedString("scanner_tab", comment: ""),
                                systemImage: "barcode.viewfinder"
                            )
                        }
                        .tag(0)

                    NavigationView {
```

```
                HistoryView(
                    viewModel: .init(container: viewModel.fetchContainer(), isRoot: true),
                    screen: screen,
                    screenHeight: screen.size.height
                )
            }
            .tabItem {
                Label(
                    NSLocalizedString("my_codes_tab", comment: ""),
                    systemImage: "clock.arrow.circlepath"
                )
            }
            .tag(1)
            .navigationViewStyle(.stack)
            .toolbar(tabViewState.isVisibleTabView ? .visible : .hidden, for: .tabBar)

            NavigationView {
                CodeDataTypePickerView(screen: screen, viewModel: .init(container: viewModel.fetchContainer()))
            }
            .tabItem {
                Label(
                    NSLocalizedString("generator_tab", comment: ""),
                    systemImage: "qrcode"
                )
            }
            .tag(2)
```

```
                .navigationViewStyle(.stack)

                NavigationView {
                    SettingsView(viewModel:                    .init(container:
viewModel.fetchContainer()))
                }
                .tabItem {
                    Label(
                        NSLocalizedString("settings_tab", comment: ""),
                        systemImage: "gear"
                    )
                }
                .tag(3)
                .navigationViewStyle(.stack)
            }
            .tint(.white)
            .accentColor(.white)
            .truncationMode(.middle)
        }
    }
    .tint(.white)
    .environmentObject(tabViewState)
    .focused($isInputActive)
    .keyboardType(.asciiCapable)
    .disableAutocorrection(true)
}
.ignoresSafeArea(.keyboard)
.ignoresSafeArea(.all)
.onAppear {
```

```
            setupTabBarUI()
            setupToolBarUI()
        }
    }

    private func setupTabBarUI() {
        let appearance = UITabBarAppearance()
        appearance.backgroundEffect = UIBlurEffect(style:
.systemUltraThinMaterial)
        appearance.backgroundColor = UIColor(red: 0.004, green: 0.063, blue:
0.200, alpha: 0.6)
        UITabBar.appearance().standardAppearance = appearance
        UITabBar.appearance().scrollEdgeAppearance = appearance
    }

    private func setupToolBarUI() {
        let appearance = UIToolbarAppearance()
        appearance.backgroundEffect = UIBlurEffect(style:
.systemUltraThinMaterial)
        appearance.backgroundColor = UIColor(red: 0.004, green: 0.063, blue:
0.200, alpha: 0.6)
        UIToolbar.appearance().standardAppearance = appearance
        UIToolbar.appearance().scrollEdgeAppearance = appearance
    }

}
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

«РОЗРОБКА ДОДАТКУ УНІВЕРСАЛЬНОГО СКАНЕРА/ГЕНЕРАТОРА КОДІВ З ІСТОРІЄЮ ТА СИНХРОНІЗАЦІЮ ЧЕРЕЗ ICLOUD»

Виконав: студент 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Кмитюк Д. С.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

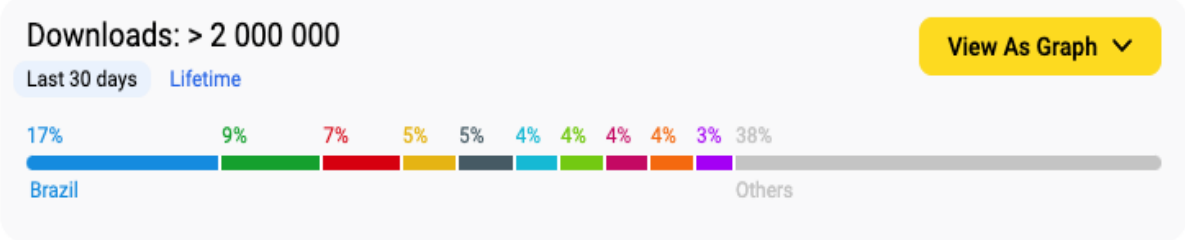


Рисунок Г.1 – «Кількість завантажуваннь додатку “QR & Barcode Reader”»

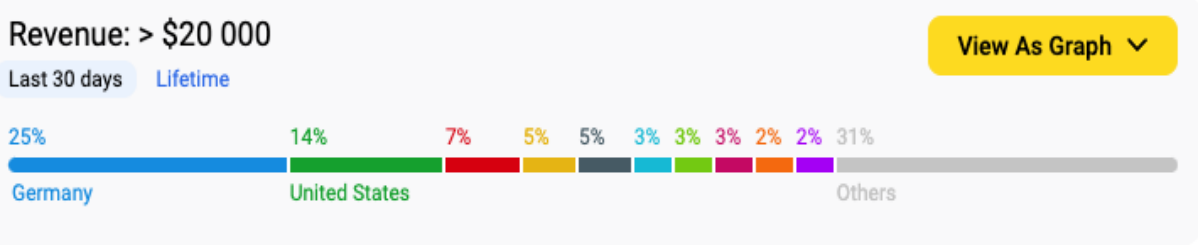


Рисунок Г.2 – «Кількість коштів, які заробив додаток “QR & Barcode Reader”»

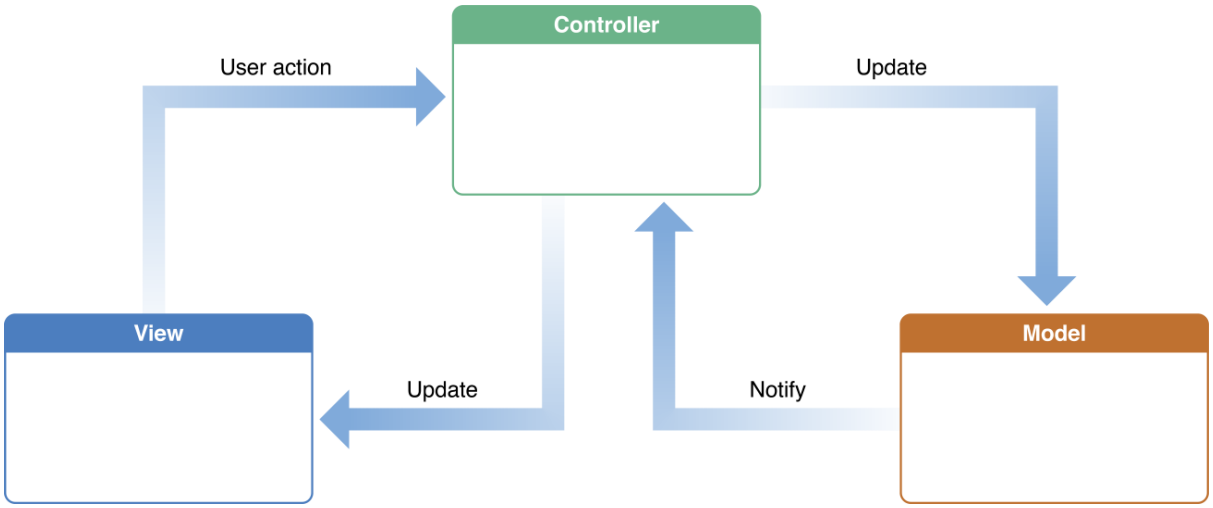


Рисунок Г.3 – «Схема роботи архітектури MVC»

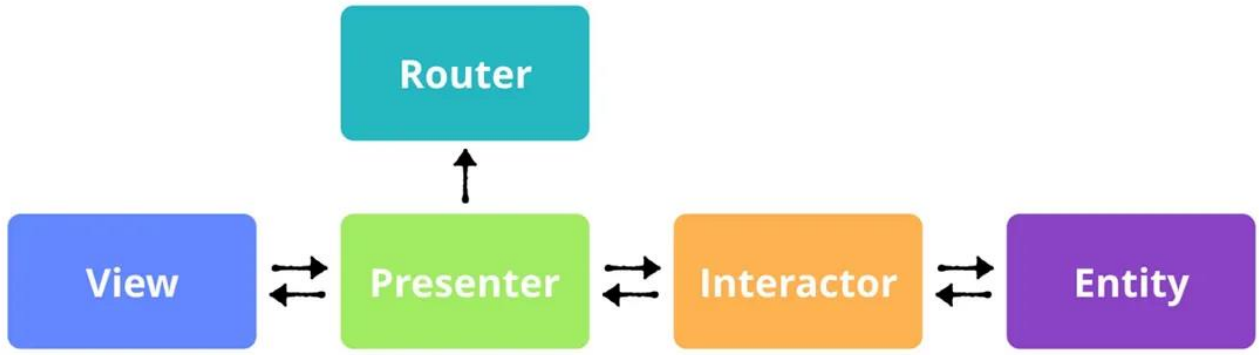


Рисунок Г.4 – «Схема роботи архітектури VIPER»

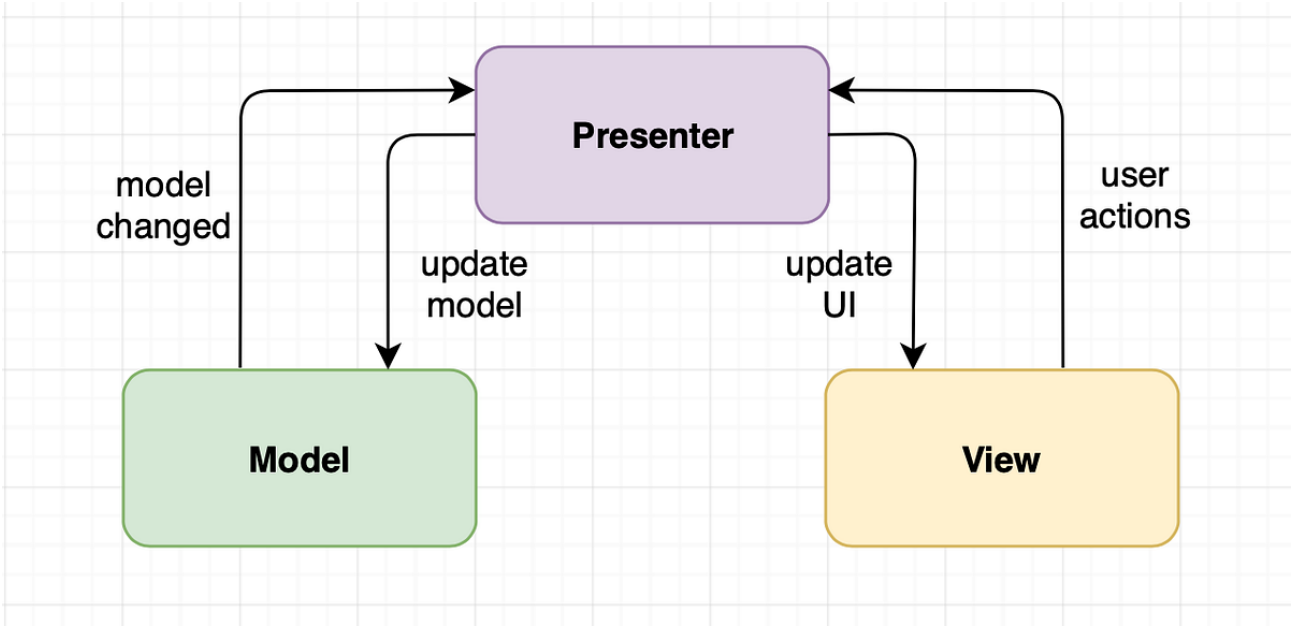


Рисунок Г.5 – «Схема роботи архітектури MVP»

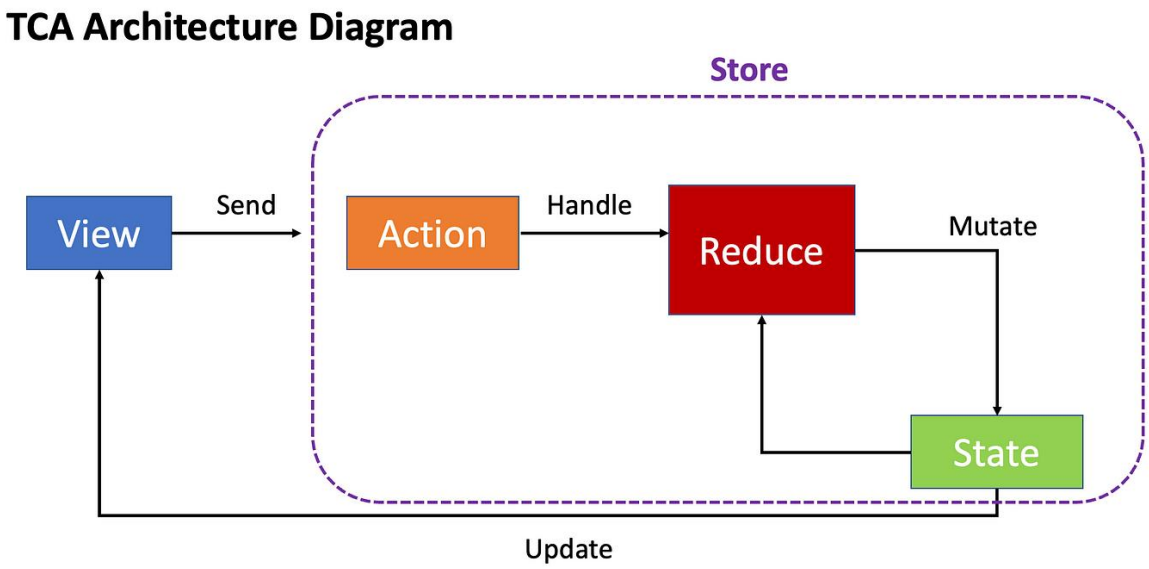


Рисунок Г.4 – «Схема роботи архітектури TCA»

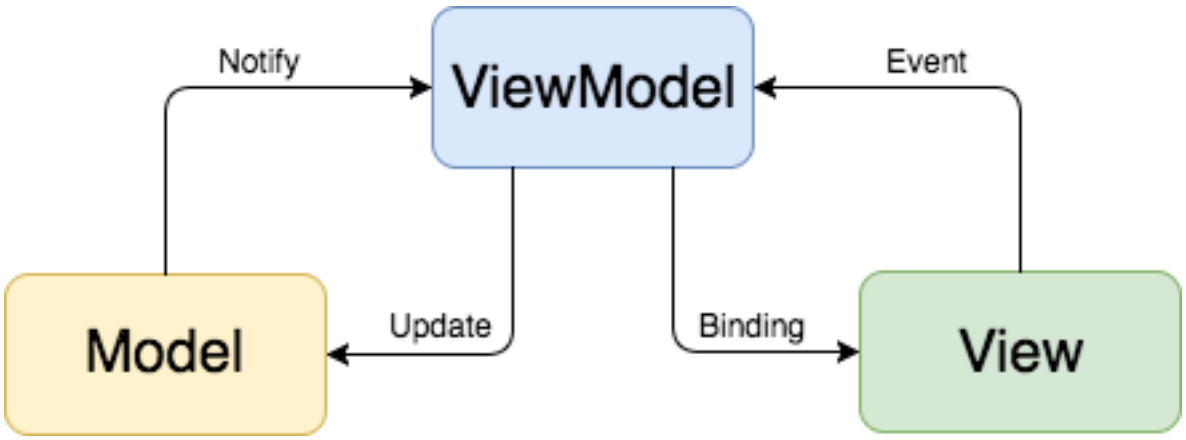


Рисунок Г.4 – «Схема роботи архітектури MVVM»

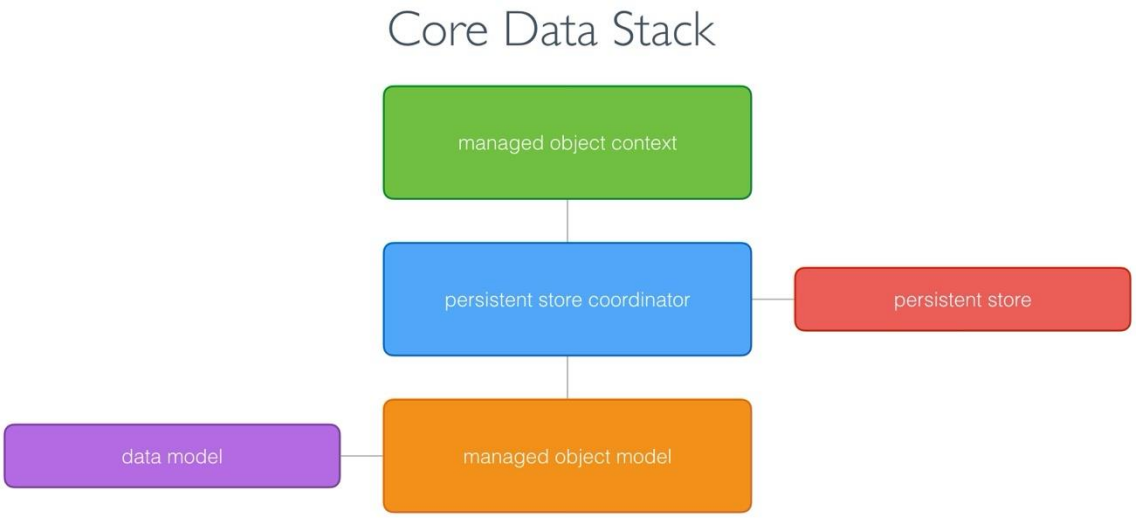


Рисунок Г.5 – «Схема CoreData»

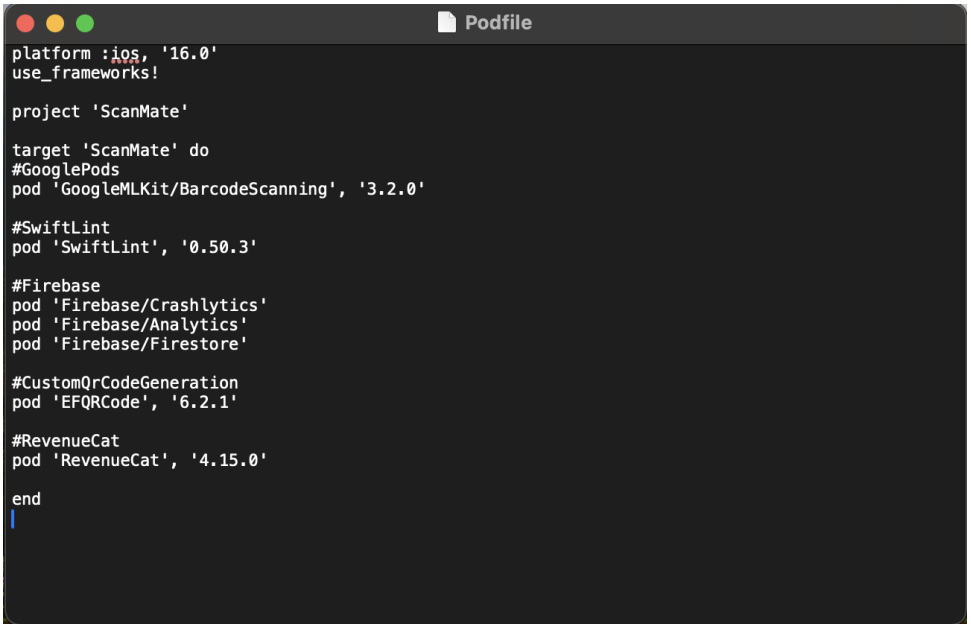


Рисунок Г.6 – «Встановлення podfile у додатку»

Information Property List	Dictionary	(13 items)
CLIENT_ID	String	227741534879-0ljckp6iuqo4098d6irsf456inq06uac.apps.googleusercontent.com
REVERSED_CLIENT_ID	String	com.googleusercontent.apps.227741534879-0ljckp6iuqo4098d6irsf456inq06uac
API_KEY	String	AIzaSyD7rvnPbPAlLoKlxoDhUOpAiwiQN9ThS7o
GCM_SENDER_ID	String	227741534879
PLIST_VERSION	String	1
BUNDLE_ID	String	com.aniestudio22.scanmate
PROJECT_ID	String	scanmate-fbcc4
STORAGE_BUCKET	String	scanmate-fbcc4.appspot.com
IS_ANALYTICS_ENABLED	Boolean	NO
IS_APPINVITE_ENABLED	Boolean	YES
IS_GCM_ENABLED	Boolean	YES
IS_SIGNIN_ENABLED	Boolean	YES
GOOGLE_APP_ID	String	1:227741534879:ios:df4afa2edb3f00d4f230aa

Рисунок Г.7 – «Встановлення MLKit у додатку»

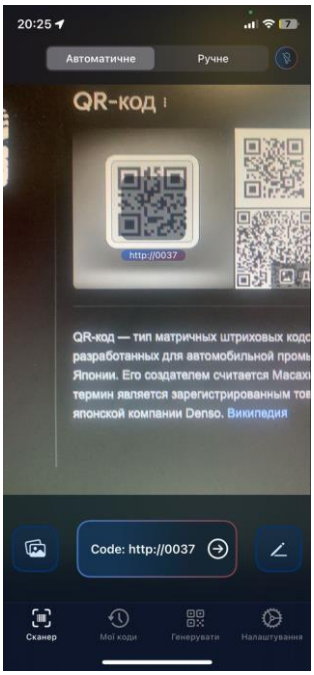


Рисунок Г.8 – «Перевірка роботи MLKit у додатку»

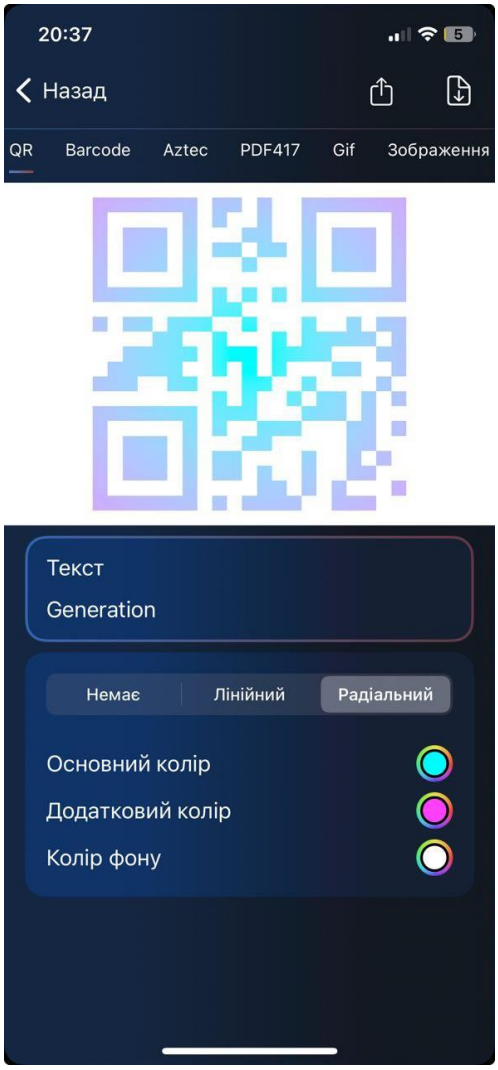


Рисунок Г.9 – «Перевірка роботи функції генерації»

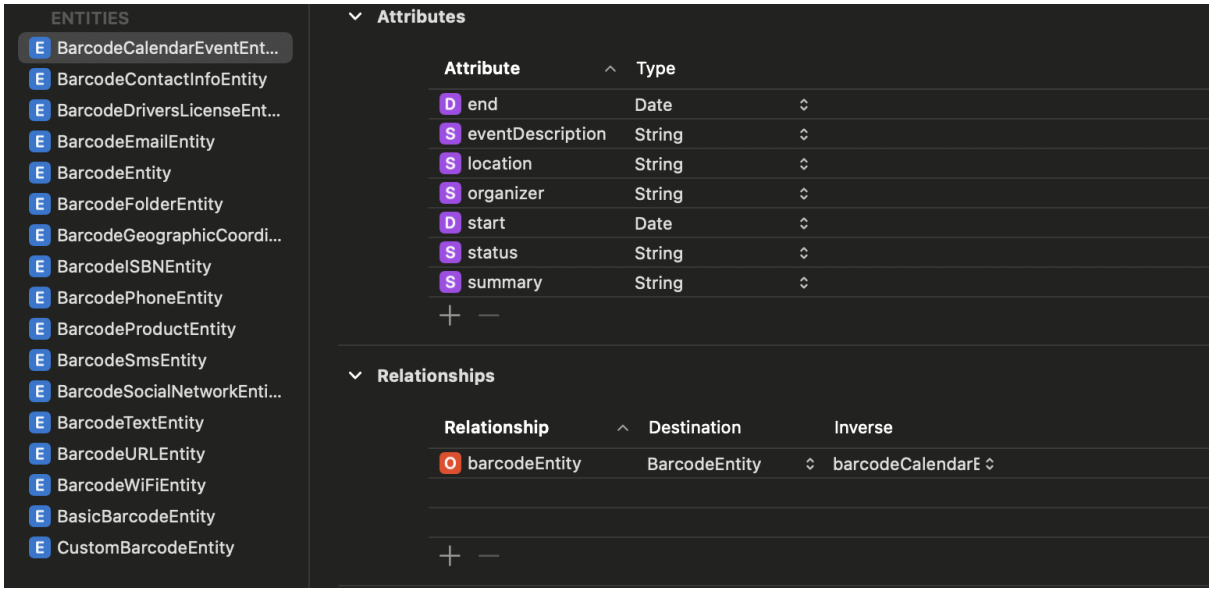


Рисунок Г.10 – «Демонстрація сутностей у CoreData»

```
import FirebaseFirestore
import Combine

final class FireStoreRemoteRepositoryImpl: FireStoreRemoteRepository {

    typealias Identifier = String

    private lazy var fireStore = Firestore.firestore()

    func fetchPromoCode(_ codeBody: String) -> AnyPublisher<PromoCode?, Never> {
        return Future<PromoCode?, Never> { [weak self] promise in
            guard let self = self else { return }

            self.fireStore.collection("promoCodes").whereField("codeBody", isEqualTo: codeBody).getDocuments { snapshot, error in
                if error != nil {
                    promise(.success(nil))
                } else {
                    guard
                        let document = snapshot?.documents.first,
                        let promoCode = PromoCode(dictionary: document.data())
                    else {
                        promise(.success(nil))
                        return
                    }
                    promise(.success(promoCode))
                }
            }
        }.eraseToAnyPublisher()
    }

    func deletePromoCode(_ codeBody: String) {
        fireStore.collection("promoCodes").whereField("codeBody", isEqualTo: codeBody).getDocuments { [weak self] snapshot, _ in
            guard let self,
                  let document = snapshot?.documents.first
            else { return }
            self.fireStore.collection("promoCodes").document(document.documentID).delete { _ in }
        }
    }

}
```

Рисунок Г.11 – «Демонстрація класу, який відповідає за історію пошуку»

```swift
var body: some View {
    GeometryReader { screen in
        ZStack {
            BackgroundImageView()

            List {
                if !viewModel.isProVersionUnlocked {
                    constructSectionView(with: .subBanner(viewModels: viewModel.constructSubscriptionBannerCellViewModel()))
                        .listRowInsets(EdgeInsets())
                        .listRowBackground(Color.clear)
                }
                constructSectionView(with: .iCloudSync(viewModels: viewModel.constructiCloudSyncCellViewModel()),
                                     sectionTitle: "storage",
                                     footerTitle: NSLocalizedString(Constants.iCloudSyncFooterTitleKey, comment: ""),
                                     footerNote: NSLocalizedString(Constants.iCloudSyncFooterNoteTitleKey, comment: ""))
                constructSectionView(with: .general(viewModels: viewModel.constructGeneralCellViewModels()), sectionTitle: "general")
                constructSectionView(with: .feedback(viewModels: viewModel.constructFeedbackCellViewModels()), sectionTitle: "support_and_feedback")
                constructSectionView(with: .delete(viewModels: viewModel.constructDeleteCellViewModels()), sectionTitle: "data_managemant")
                constructSectionView(with: .privacy(viewModels: viewModel.constructPrivacyCellViewModels()), sectionTitle: "terms_of_use" )
                constructSectionView(with: .promoCode(viewModels: viewModel.constructPromoCodeCellViewModels()), sectionTitle: "promocodes",
                                     footerTitle: NSLocalizedString("promocode_cell_footer_key", comment: ""))
            }
        }
        .overlay {
            if shouldShowLoader {
                ZStack {
                    Rectangle()
                        .foregroundColor(.clear)
                        .frame(width: screen.size.width * 0.2,
                               height: screen.size.width * 0.2)
                        .overlay(.ultraThinMaterial)
                        .cornerRadius(20)

                    VStack {
                        ProgressView()
                            .tint(.white)
                            .scaleEffect(2)
                            .progressViewStyle(CircularProgressViewStyle())
                    }
                }
```

Рисунок Г.12 – «Проектування екрану налаштувань»

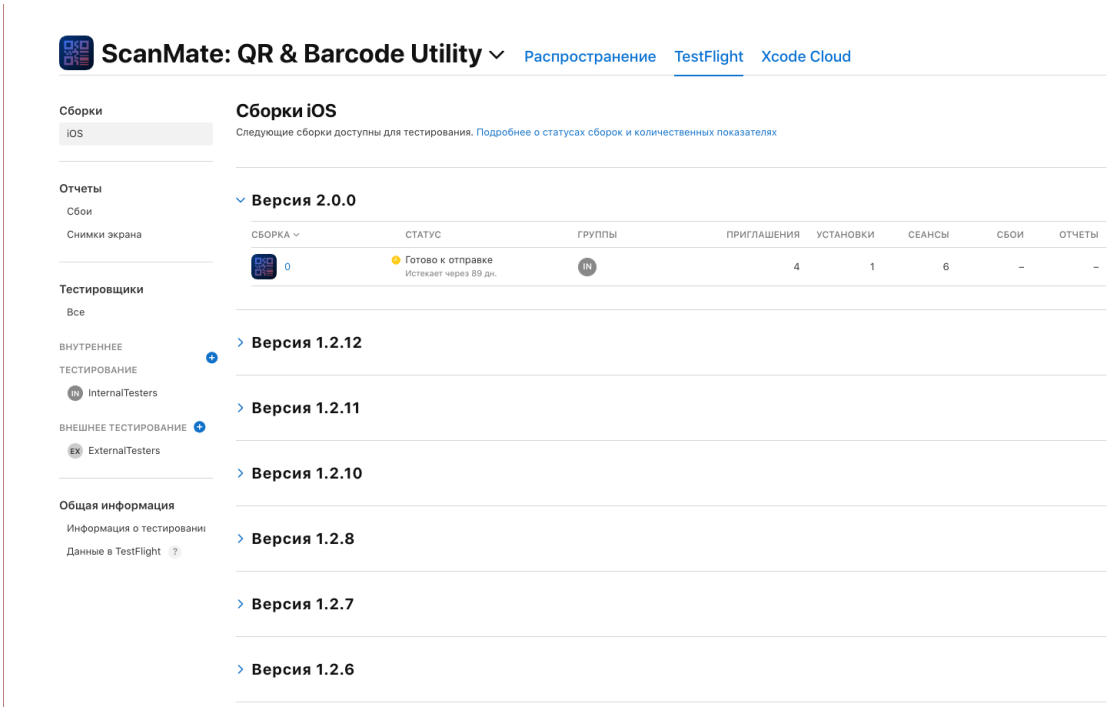


Рисунок Г.13 – «Розміщення додатку на App Store Connect»

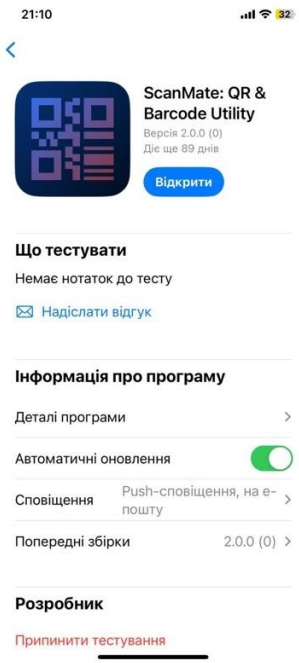


Рисунок Г.14 – «Розміщення додатку на платформі Test Flight»

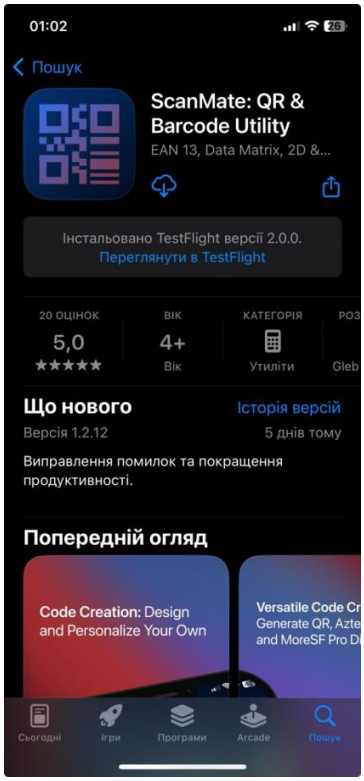


Рисунок Г.15 – «Зображення додатку у App Store»