ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для сканування та аналізу штрих-кодів товарів з аналізом прихильностей споживачів і оцінкою корисності мовою Swift»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро ШЕВЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Дмитро ШЕВЧЕНКО

Керівник: Вячеслав САВІЦЬКИЙ
асистент кафедри ІПЗ

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шевченку Дмитру Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для сканування та аналізу штрих-кодів товарів з аналізом прихильностей споживачів і оцінкою корисності мовою Swift»
керівник кваліфікаційної роботи ассистент кафедри ІПЗ Вячеслав САВІЦЬКИЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки мобільного застосунку, опис інструментів для розробки мобільного застосунку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1. Пошук та аналіз існуючих застосунків для сканування штрих кодів для мобільних пристроїв.
2. Проектування архітектури мобільного застосунку для сканування штрих кодів.
3. Пошук та аналіз методів розробки мобільного застосунку.

4. Програмна реалізація та опис функціонування застосунку для сканування штрих кодів товарів.
5. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Архітектура системи.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих застосунків на ринку для сканування штрих кодів	14.03-19.03.2024	
4	Проектування застосунку для сканування штрих кодів з можливістю порівняння цін	20.03-02.04.2024	
5	Програмна реалізація застосунку	03.04-21.04.2024	
6	Тестування застосунку	22.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Дмитро ШЕВЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Вячеслав САВІЦЬКИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _51_ стор., _18_ рис., _7_ джерел.

Мета роботи – спрощення процесу аналізу штрих кодів та порівняння цін з різних продуктових магазинів

Об'єкт дослідження – процес сканування та аналізу штрих кодів товарів з аналізом прихильностей споживачів і оцінкою користності

Предмет дослідження – мобільний застосунок для сканування та аналізу штрих-кодів товарів з аналізом прихильностей споживачів і оцінкою корисності.

Короткий зміст роботи: В роботі проаналізовано алгоритми та методи для розробки мобільного застосунку в контексті задачі розробки мобільного застосунку для сканування штрих кодів. Проаналізовано інструменти від Apple для розробки мобыльного застосунку такі як: SwiftUI, CarBode, Lottie-ios, SwiftMessages, SwiftyStoreKit. Розроблено архітектуру застосунку та програмно реалізовані ключові функціональні можливості, зокрема: сканування штрих коду як з фтовару так і з вже готового фото, відображення інформації про товар, відображення цін на товар, можливість скаладит список покупок, можливість залишати та переглядати відгуки до товару . Проведено функціональне та модульне тестування додатку. В роботі використано програмні продукти для створеня дизайну, анімації переходут між екранами, завантаження даних.

Сферою використання застосунку є спрощення отримання інформації про товар за допомогою сканування штрих коду в продуктових магазинах.

КЛЮЧОВІ СЛОВА: iOS, Swift, мобільний застосунок.

ЗМІСТ

ВСТУП

Ще буквально декілька років тому, до повномаштабного вторгнення росії, наша економіка активна розвивалася, і люди спокійно могли дозволити купляти будь-яку продукцію і особливо не турбувалися яка її вартість, але війна перекреслила увесь прогрес української економіки за 2016-2021 роки.

Ні для кого не секрет що за останні роки після повномасштабного російського вторгнення рівень бідності в Україні зріс і з кожним днем все більше громадян нашої країни змушені задумуватися над тим, як зекономити власні кошти.

Продукти харчування – це один із основних джерел витрат будь якої людини. Людина може відмовитися від розваг, подорожей, нової техніки або брендового одягу, але від їжі людина не зможе відмовитися ніколи, це не лише наше здоров’я, а й життєво важливе джерело енергії. Кожного дня ми продумуємо раціон їжі на завтрашній день із чого і формуємо наш список покупок і думаємо де саме зробимо закупку.

Продуктові ринки вже давно почали відходити в минуле, їх стали витісняти великі супермаркети в яких ми можемо знайти все і відразу, тому порівняння цін на продукти харчування із супермаркетів є більш актуальною. Все більше людей з кожним днем задає собі питання, де б можна було купити той чи інший товар за більш сприятливою ціною? Звичайно якщо ми говоримо про покупку пляшки води або однієї шоколадки то очікувати на якусь значну економію не варто, але й забувати про те що часті покупки на малу суму можуть перетворитися в результаті на величезну.

Наприклад, ще в недалекому минулому, щоб просто дізнатися де і коли можна придбати товар за найнижчою ціною, люди треба було сходити в

декілька магазинів, знайти товар який цікавить, записати десь собі ціну, і потім порівняти їх. Це все вимагало багато часу та не малих зусиль. Однак з появою нових технологій цей процес значно спрощується. В кожного великого супермаркета, вже є свій додаток в якому ви з легкістю можете знайти та побачити ціну будь якого товару, але щоб порівняти ціни з різних магазинів, вам треба встановити всі ці додатки і при порівнянні ціни в різних додатках, вам потрібно буде шукати товар знову.

Тому виникає питання про розробку системи порівняння цін продуктів з різних супермаркетів для зменшення витрат при закупівлі зі зручним та зрозумілим інтерфейсом

1. АНАЛІЗ ТА ХАРАКТЕРИСТИКА МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Останні тенденції розвитку мобільних додатків

У сучасному цифровому світі мобільні додатки стали невід'ємною частиною повсякденного життя. Близько 87% користувачів смартфонів проводять свій час у мобільних застосунках, включаючи соціальні мережі, розважальні програми, а також інструменти для підвищення продуктивності та бізнес-застосунки. Мобільні додатки відіграють ключову роль у нашій взаємодії з технологіями. Розгляньмо, що таке мобільний додаток і які існують його типи та види.

Мобільний додаток, або мобільна програма, – це програмне забезпечення, спеціально розроблене для використання на невеликих бездротових пристроях, таких як смартфони та планшети, на відміну від програм для настільних комп'ютерів або ноутбуків.

Мобільні додатки класифікуються за тим, чи є вони веб-застосунками, нативними додатками, створеними для конкретної платформи, або гібридними додатками, які поєднують елементи як нативних, так і веб-додатків.

Оскільки ринок мобільних додатків продовжує зростати, а потреби користувачів змінюються, знання останніх тенденцій та можливостей мобільної розробки дозволить розробникам адаптуватися до нових вимог клієнтів, залишатися конкурентоспроможними на ринку та створювати сучасні, інноваційні рішення. Розглянемо основні тренди мобільної розробки у 2023 році.

Технологія 5G

Дослідження e-marketer показують, що кількість людей, підключених до мережі 5G, щорічно зростає, і до 2024 року вона досягне 1,5 мільйона користувачів. Зовсім скоро можна очікувати значних змін у комунікаційній сфері.

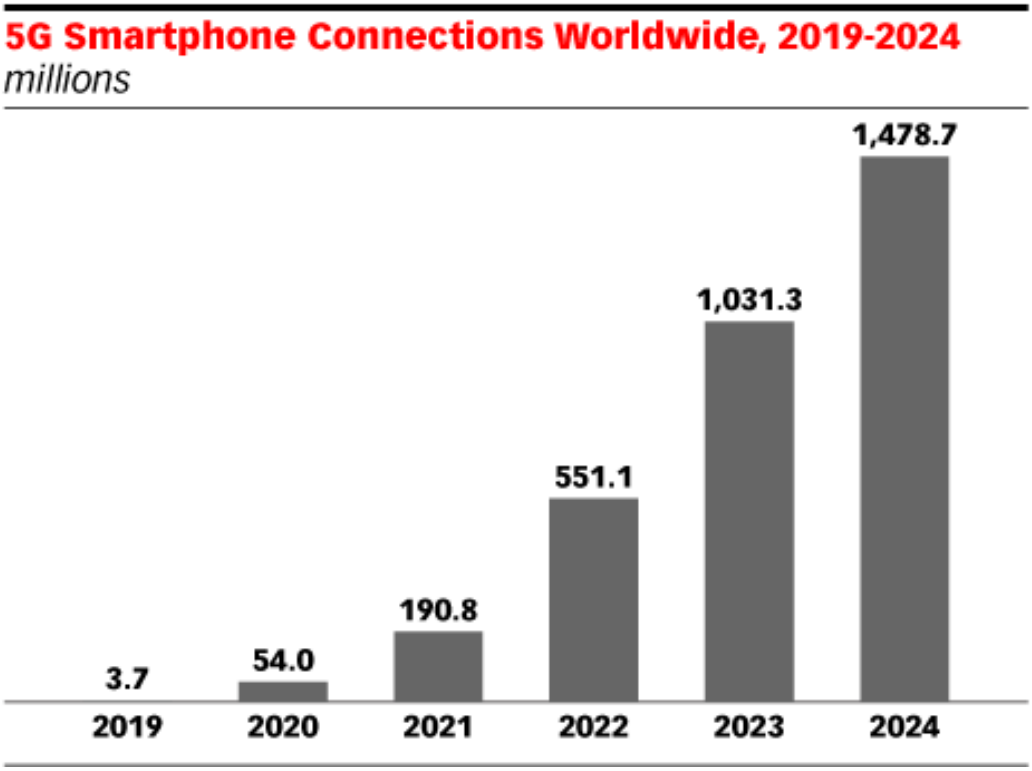


Рис. 1.1 Кількість підключень до 5G

Серед переваг цієї технології:

- 10 мілісекунд прискорення використовуються для завантаження вмісту, який у 100 разів швидше, ніж 4G.
- Затримка зменшена з 50 мілісекунд, пов'язаних із технологією 4G, до 1 мілісекунди,
- Вища роздільна здатність і краща продуктивність, що корисно для програм потокового відео.
- Швидша та ефективніша передача даних між пристроями.
- Збільшує кількість користувачів, пристроїв і послуг,
- Завдяки швидкій обробці даних під час ідентифікації мобільні платежі є більш надійними та ефективними.

Що вигідно розробникам? Вони матимуть можливість доповнювати програми додатковими функціями, додавати більше функцій без шкоди для продуктивності. Наприклад, створення ефекту занурення в AR і VR, покращення телевізійних передач у прямому ефірі та інші заходи.

Технологія інтернет речей та хмарні сервіси

IoT (Інтернет речей) – це машини, які підключені до Інтернету та спілкуються одна з одною. Інтернет речей сприяє значному збільшенню життя споживачів.

Так, за допомогою мобільного додатку ви можете дистанційно змінювати температуру системи кондиціонування чи термостата в резиденції, увімкнути або вимкнути сигналізацію. Також до IoT входять системи розумного дому, роботизовані пилососи тощо, усі вони ініціюються та програмуються за допомогою мобільних додатків.

Розробка мобільних додатків, яка враховує IoT на різних типах пристроїв, популярна в охороні здоров'я, роздрібній торгівлі та інших галузях. Підприємства використовують мобільні програми для IoT, щоб стежити за продуктивністю машин або віддалених місць. Як наслідок, попит на послуги з розробки додатків зростає, оскільки компанії намагаються створити надійне плавне з'єднання між інтелектуальними пристроями та програмами.

Хмара та Інтернет речей нерозривно пов'язані. Якщо IoT створює великий обсяг даних, то необхідна хмарна служба для транспортування, зберігання та керування ними з віддаленого місця. Крім того, коли інформація зберігається в хмарі, компанії мають миттєвий доступ до великих наборів даних у будь-який час і на будь-якому пристрої.

IoT — це популярна платформа веб-розробки, яка розвивається страшною швидкістю. IT-експерти повинні зосередитися на розумінні тонкощів розробки додатків із використанням цієї технології, щоб бути в курсі останніх тенденцій.

Носимі пристрої

Розумні годинники, трекери та браслети для фітнесу стають все більш популярними. Це не дивно, адже їхній потенціал величезний. За допомогою носимих технологій сьогодні це вже дозволяє отримувати дзвінки та повідомлення, контролювати артеріальний тиск, частоту сердечного ритму, рівень концентрації в крові кисню та емоційний стан, створювати нагадування про події, здійснювати

платежі безконтактно та навіть виявляти ДТП. Це далеко не всі можливості цих пристроїв.

Носимі технології будуть широко використовуватися в майбутньому, і, без сумніву, наступні роки будуть визначними в цій галузі. Не в останню чергу через асоціацію з передовими технологіями (AR/VR, NFC, 5G). Це викличе необхідність розробки мобільних додатків для носимих технологій.

Це надає численні можливості для розробників, які слідують тенденції мобільної розробки. Створюючи нові програми, вони повинні зосередити свою увагу на створенні зручних, адаптивних та інтерактивних продуктів, які задовольнять зростаючі вимоги та очікування користувачів.

Машинне навчання та Штучний інтелект

Штучний інтелект робить комп'ютери більш інтелектуальними, що зрештою підвищує їх ефективність і покращує взаємодію з користувачем. Найпоширенішими цілями ШІ (штучного інтелекту) є розпізнавання облич, зображень, мови, письма та створення фільтрів.

Чат-боти, віртуальні радники, безпілотні автомобілі та дрони, фільтри захисту від спаму в електронній пошті, інтелектуальний пошук Google, створення цифрових зображень із текстових описів природною мовою — це далеко не повний перелік сфер, у яких використовується ШІ.

За допомогою машинного навчання (ML, також відомого як машинне навчання) полегшується завдання віртуальних помічників, алгоритмів, які передбачають результати, та інших технологічних інновацій. Програмне забезпечення оцінює дані та визначає закономірності, а потім приймає рішення, які підвищать якість його власної роботи.

Наприклад, інженери Netflix створили просунуті алгоритми на основі ML, що допомогло краще задовольнити запити користувачів платформи. З цього моменту кожному споживачеві надається унікальний ідентифікатор, а потім програмне забезпечення надає йому вміст і результати, які стосуються його запланованого пошуку, а не попередні запити, які зазвичай надсилаються (див. нижче).

Оскільки штучний інтелект надає додаткам майже безмежні можливості, обсяг використання лише зростатиме. Завдяки співпраці ML допомагає розробникам та інженерам доповнювати системи ШІ, тому технологія машинного навчання навряд чи зникне найближчим часом.

У майбутньому важливо зосередитися на таких аспектах, як використання природної мови (зокрема, у чат-ботах), розпізнавання зображень для покращення контакту з користувачами, налаштування каналу, створення чат-ботів, які розуміють історію та вподобання своїх клієнтів , а також виявлення потенційних небезпек (шахрайство, розкриття даних).

Мобільна комерція

Мобільна торгівля або комерція – це практика здійснення покупок в Інтернеті через мобільний додаток. Його впровадження сприяло скороченню часу, витраченого на купівлю товарів у порівнянні з купівлею товарів через веб-сайт, крім того, компанії та бренди використовують мобільні додатки для збільшення своїх доходів (Statista стверджує, що у 2021 році понад 70% продажів у -сфера комерції викликана мобільними пристроями).

Крім того, зростає популярність мобільних гаманців і можливості оплачувати покупки прямо в додатку, не вводячи щоразу дані картки. Це спрощує онлайн-платежі, скорочує час, необхідний для оплати онлайн, і робить його більш зручним. Згодом інтеграція гаманця стане звичайним компонентом програмного додатку, який виконує транзакції.

Оскільки кількість людей, які використовують програми для купівлі речей, зростає, мобільна комерція в майбутньому стане популярною формою мобільного розвитку.

Beacon technology

Технологія маяків (beacon) була вперше популяризована в 2013 році, і, ймовірно, вона стане одним із найпопулярніших напрямків мобільної розробки в майбутньому. Beacon використовує портативні мобільні передавачі (маяки) для передачі радіосигналу на інші пристрої поблизу (мобільні телефони з програмами).

Ці пристрої розміщені в магазинах, авіакомпаніях, лікарнях та інших місцях. Коли телефон клієнта знаходиться в зоні дії маяка, пристрій надсилатиме сповіщення про майбутні розпродажі чи поточні пропозиції, персоналізовані пропозиції, карту для внутрішньої навігації тощо.

Для компаній технологія маяків дозволяє досліджувати вподобання та моделі поведінки клієнтів (наприклад, кількість часу, яку клієнт проводить у певному відділі магазину). По суті, це безконтактний маркетинг, метою якого є підвищення задоволеності клієнтів і стимулювання продажів.

1.2 Аналіз ринку мобільних додатків

Стадія зростання ринку мобільних додатків є середньою, а темпи зростання ринку прискорюються. Індустрія мобільних додатків є фрагментованою, з великою кількістю розробників мобільних додатків. Розробники мобільних додатків використовують передові технології, такі як штучний інтелект (ШІ) та Інтернет речей (IoT). ШІ в мобільних додатках може використовуватися для підтримки внутрішніх функцій, таких як розпізнавання облич. Крім того, мобільний додаток, інтегрований з IoT, може надавати користувачам цінну інформацію. Наприклад, мобільні додатки на основі Інтернету речей можуть надавати виробничим компаніям інформацію про машини в режимі реального часу.

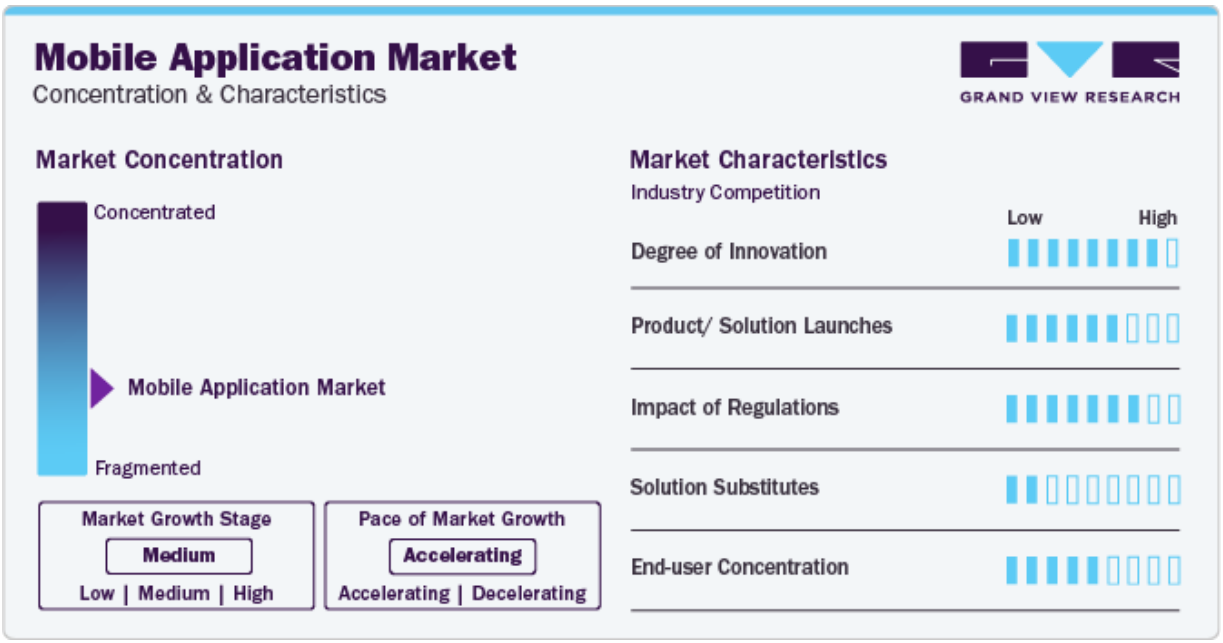


Рис. 1.2 Характеристика ринку моб. застосунків

Гравці ринку інвестують у розробку мобільних додатків, що є гарною ознакою для зростання ринку. Наприклад, у червні 2023 року французька компанія Ubisoft Entertainment оголосила про запуск додатку для навчання музиці Rocksmith+. Мобільний додаток став доступним на платформах Android від Google та iOS від Apple Inc. Додаток надає доступ до понад 6 000 пісень і дозволяє користувачам навчатися грі на гітарі з будь-якого місця і в будь-який час.

Концентрація і характеристики ринку

Безпека мобільних додатків є важливим фактором, і мобільні додатки вимагають інформування користувачів про практику збору даних, включаючи використання, збір та зберігання даних. Нормативні акти, що впливають на додатки, вимагають від розробників додатків створювати політику конфіденційності та інформувати користувачів про те, як додаток збирає, зберігає та використовує їхні дані. Крім того, розробники додатків повинні інформувати користувачів про їхні права на конфіденційність і пояснювати, які саме персональні дані, такі як ім'я користувача, пароль, ім'я та прізвище, адреса та інші, збираються у них.

Загроза появи замінників мобільних додатків є низькою, оскільки прямих замінників не існує. Більше того, зростаюче використання мобільних додатків для різних цілей, таких як онлайн-покупки, замовлення їжі та цифрові платежі, підвищує попит на мобільні додатки. Очікується, що зі зростанням проникнення інтернету та смартфонів потреба в мобільних додатках зростатиме. Простота використання мобільних додатків, ймовірно, підтримуватиме попит на них серед користувачів.

Люди по всьому світу використовують мобільні додатки для різних цілей, включаючи розваги, ігри та соціальні мережі. Мобільні додатки задовольняють усі типи користувачів, а різні категорії додатків обслуговують користувачів залежно від демографічних факторів, таких як вік і стать. Наприклад, спортивні додатки частіше використовують користувачі-чоловіки, а ігрові додатки частіше використовує молоде покоління.

Статистика програм

Сегмент ігрових додатків на ринку мобільних додатків мав найбільшу частку доходу - понад 34,0% у 2023 році. Крім того, очікується, що сегмент збереже своє домінування протягом прогнозованого періоду завдяки безпрецедентному зростанню кількості геймерів і, як наслідок, кількості мобільних ігрових додатків у таких країнах, як Китай та Індія. Крім того, трьома основними платформами операційних систем (ОС), що використовуються для створення мобільних ігор, є iOS, Android і Windows. У 2021 році на ОС Android припав значний обсяг завантажень ігор, тоді як iOS згенерувала більший дохід завдяки платній природі ігор.

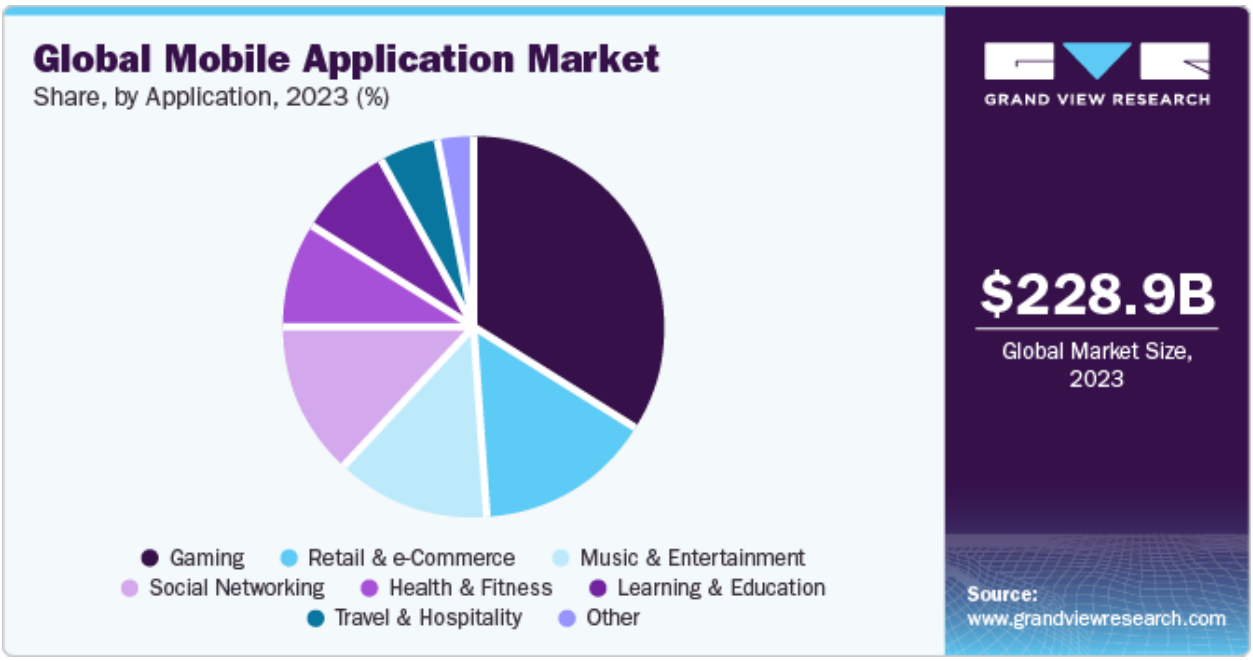


Рис. 1.3 Розподіл ринку мобільних застосунків за сферами

Прогнозується, що сегмент музичних і розважальних додатків продемонструє значне середньорічне зростання на рівні близько 14,9% у період з 2024 по 2030 рік. Сегмент включає в себе музичні та розважальні додатки, такі як Netflix, HBO NOW, Tinder, Spotify, YouTube, Amazon Prime і Hulu та інші. Використання музичних і розважальних додатків значно збільшилося завдяки зростаючій популярності потокових сервісів. Додатки для прямих трансляцій, такі як Netflix, YouTube та Instagram, дозволяють користувачам транслювати відео в прямому ефірі для широкої аудиторії, а також залучати її на щоденній основі.

1.3 Варіанти створення мобільних додатків

Розробка мобільних додатків вважається однією з найбільш інноваційних і швидкозростаючих галузей IT. З кожним роком кількість користувачів смартфонів зростає, що, як наслідок, зумовлює зростання попиту на мобільні додатки.

Під час розробки програмного додатку розробникам доводиться вибирати між розробкою окремо для Android та iOS або створенням гібридного додатка, який

є кросплатформним. Кожен із цих підходів має переваги та недоліки. Розглянемо кожен із цих підходів більш детально.

Нативні додатки

Нативні застосунки — це програми, які спеціально створені для певної мобільної платформи (iOS або Android) з використанням рідних мов програмування та засобів розробки для цієї платформи. Процес розробки додатків для iOS зазвичай більш вимогливий до розробників. Зокрема, особи повинні мати комп'ютер із операційною системою macOS. Звичайно, ви можете встановити ОС hackintosh без міток, але це не завжди життєздатний варіант, і його неможливо встановити на всіх комп'ютерах. З програмами для Android це зазвичай простіше, але є й унікальні аспекти.

Перевагами нативних додатків є:

- Простота

Ці програми простіше освоїти та мають більш інтуїтивно зрозумілий досвід, оскільки вони дотримуються правил дизайну та інтерфейсу платформи. Користувачі одразу розуміють, як використовувати ці програми, оскільки вони схожі на інші програми на їх персональному пристрої. Нативні програми також мають вищий ступінь сумісності з різними розмірами екрану та орієнтацією дисплея.

- Швидкість роботи

Ці програми є ефективними та стабільними, оскільки вони розроблені з урахуванням особливостей їхньої операційної системи та використовують нативні мови програмування, такі як Swift або Java для Android або iOS. Нативні програми також можуть зберігати дані на самому пристрої, це зменшує потребу в підключенні до Інтернету та покращує продуктивність.

- Безпека

Також нативні програми безпечніші, оскільки вони підлягають суворим нормам щодо якості та відповідності перед випуском в офіційних магазинах, таких як App Store або Google Play. Вони також можуть використовувати вбудовані в систему механізми безпеки, такі як біометрична автентифікація, шифрування даних і SSL-карти.

- Краща інтеграція

Ці програми можна легко поєднувати з іншими компонентами пристрою, наприклад камерою, мікрофоном, GPS, контактами, календарем, сповіщеннями тощо. Це полегшує використання нативних програм, забезпечуючи більш комплексний і зручний досвід, а також використовує максимально використовувати можливості пристрою.

Недоліками нативних додатків є:

- Витрати на розробку

Створення нативних програм зазвичай потребує більше часу, ресурсів і грошей, оскільки їх потрібно розробляти окремо для кожної платформи, а також спеціальну команду розробників, які розуміють нативну мову. Для публікації такого типу застосунку може знадобитися багато часу для перевірки його, зокрема, в App Store..

- Складніша підтримка та оновлення

Нативні програми часто оновлюються, щоб залишатися функціональними та сумісними з новими операційними системами та пристроями. Нові оновлення нативних програм залежать від схвалення магазинів програм і відмови користувачів. Нативнії програми також складніше підтримувати та виправляти неполадки через їхню кількість версій і варіантів.

- Кросплатформеність

Нативні програми не можуть працювати на платформах, відмінних від призначених для них. Якщо розробник хоче зробити свою програму доступною для більшої аудиторії, він повинен створити нову версію програми для іншої

платформи, це потребує часу та грошей. Нативним програмам також важко використовувати такі веб-технології, як SSO, SEO або базова онлайн-статистика.

Гібридні додатки

Гібридні програми – це комбінація веб- і нативних програм. Зокрема, мається на увазі їхні кросплатформні можливості та доступ до функцій смартфона. Такі програми можна завантажити лише з таких торгових майданчиків, як Google Play і App Store. При цьому вони можуть вибрати автономне оновлення інформації, а для їх роботи потрібне підключення до Інтернету. Без останнього функціональність мережі просто не працюватиме.

У багатьох компаніях найпоширенішим варіантом є розробка гібридних програм. Це можна пояснити тим, що гібридні програми здатні поєднувати переваги нативних програм із технічною відповідністю, яку забезпечують новітні мережеві технології. Однак, на відміну від рідних, створювати гібриди на порядок дешевше і швидше. Спорідненість гібридних програм із веб-програмами, у свою чергу, приносить плоди у вигляді того, як їх можна легко та швидко адаптувати. Це означає, що розробникам не потрібно повторно публікувати програму в магазині, щоб усунути помилки з попередніх версій, як вони роблять із рідними програмами.

Розробка гібридних додатків виглядає багатообіцяючою, оскільки передбачає створення на двох платформах одночасно. Таким чином, це позбавляє від клопоту щодо розробки програм окремо для кожної операційної системи.

Перевагами гібридних додатків є:

- Кросплатформеність

Гібридні програми можуть працювати на кількох платформах, включаючи iOS, Android, WIndows, а також веб- та десктопні версії. Це полегшує розширення аудиторії користувачів, зберігаючи однакову взаємодію з користувачами на різних пристроях.

- Час розробки

Гібридні програми мають менший час розробки, оскільки вони використовують єдину базу вихідного коду для всіх платформ. Розробникам не потрібно розуміти нативні мови програмування, і вони можуть використовувати такі інструменти та веб-технології як HTML, CSS і JavaScript. Також гібридні додатки також можуть використовувати такі фреймворки, як React Native, Flutter, Ionic тощо, що зменшує складність розробки та впровадження нової системи.

- Вартість розробки

Гібридні програми дешевші, ніж нативні програми, тому що вони потребують менше зобов'язань перед розробниками та менше ресурсів. Вони також знижують вартість тестування та підтримки завдяки наявності єдиного базового коду для всіх платформ. Крім того, гібридні застосунки не перевіряються всебічно на предмет розміщення та додавання нових функцій у магазинах програм, як нативні програми.

- Простота оновлень

Гібридні програми легко оновлювати, оскільки їх єдині вимоги — це зміна одного базового коду, а не зміна для кожної платформи. Оновлення гібридних програм може відбуватися автоматично через Інтернет і не залежить від схвалення магазину програм.

Недоліками гібридних додатків є:

- Обмежена продуктивність

Гібридні програми мають нижчу швидкість і менш стабільні, ніж нативні програми, оскільки вони працюють у контейнері, схожому на веб-браузер, який імітує рідне середовище. Інші гібридні додатки також залежать від надійного підключення до Інтернету як основного джерела коду, тому що вони беруть більшу частину коду з сервера

- Обмежені можливості

Гібридні програми мають менший доступ до функцій і ресурсів пристрою, ніж нативніі програми, оскільки вони використовують веб-технології, які не підтримують усі можливості операційної системи та компонентів пристрою. Крім того, гібридні застосунки можуть використовувати спеціальні надбудови, які

дозволяють їм використовувати деякі функції пристрою, але це може призвести до проблем із сумісністю та безпекою.

- Складність підтримки попередніх версій

Гібридні програми складніше підтримувати, оскільки вони використовують веб-технології, несумісні зі старішими версіями ОС і пристроями. Крім того, гібридні програмні продукти можуть мати проблеми зі стабільністю та продуктивністю на старих пристроях, оскільки їм потрібно більше ресурсів для функціонування.

- Недостатня оптимізація

Гібридні програми не повністю вміють враховувати особливості кожної платформи, оскільки вони мають однаковий інтерфейс і дизайн на всіх платформах. Вони також мають обмежений доступ до нативних технологій, таких як анімація, графіка, жести тощо. Гібриди можуть бути незручними для користувачів, які знайомі з нативним дизайном та інтерфейсом своєї платформи.

1.4 Мови програмування для розробки на iOS

Зазвичай iOS-розробник працює на двох мовах програмування: Objective-C та Swift

Objective-C

Objective-C - це об'єктно-орієнтована мова програмування, яка поєднує в собі можливості мови Сі з динамічною системою об'єктів та синтаксисом Smalltalk. Вона була розроблена у 1980-х роках компанією NeXT і наразі є однією з основних мов програмування для розробки програмного забезпечення для платформ Apple, таких як macOS та iOS.

Історія Objective-C сягає коріння до мови програмування C, яка була розроблена у 1972 році Деннісом Рітчі. У 1980-х роках компанія NeXT, заснована Стівом Джобсом після його відходу з Apple, вирішила розробити свою власну мову

програмування, яка би була ефективною та зручною для розробки програмного забезпечення для їхніх комп'ютерів.

Objective-C отримала значний розквіт завдяки успіху компанії Apple з випуском Macintosh та подальшим створенням платформи NeXTSTEP, яка включала Objective-C як основну мову програмування. Після придбання NeXT компанією Apple у 1996 році, Objective-C стала ключовою мовою для розробки програмного забезпечення для комп'ютерів Mac та пізніше для мобільних пристроїв iOS.

Objective-C є динамічною мовою програмування, що означає, що багато рішень приймаються під час виконання програми, а не під час компіляції. Вона підтримує об'єктно-орієнтований підхід до програмування, зокрема спадкування класів, поліморфізм та інкапсуляцію.

Objective-C використовується для розробки різноманітних програмних продуктів, включаючи десктопні та мобільні додатки, ігри, веб-служби та багато іншого. Вона є важливою мовою для розробників, що працюють у середовищі Apple, і продовжує залишатися популярною серед спільноти розробників завдяки своїй потужності та гнучкості.

Swift

Створення мови Swift було ініційовано Крісом Латнером спільно з кількома іншими розробниками. Ідеї Swift взяті з таких мов програмування, як «Objective-C, Rust, Haskell, Ruby, Python та багатьох інших мов у списку». 2 червня 2014 року на Всесвітній конференції розробників було офіційно представлено програму на основі Swift. Бета-версія мови була випущена на веб-сайті компанії як зареєстрованого розробника, але Apple не обіцяла, що представлена версія буде сумісна з фінальним випуском.

Безкоштовний ресурс із 500 сторінками, де обговорюється використання Swift, також був випущений на сервісі iBooks. Swift 1.0 було випущено 9 вересня 2014 року разом із версією «Gold Master» Xcode 6.0 для iOS. Swift 1.1 було випущено 22 жовтня 2014 року разом із Xcode 6.1. Swift 1.2 було випущено 8

квітня 2015 року з Xcode 6.3. Swift 2.0 було оголошено під час WWDC 2015. Оновлення до Swift 3.0 і додаткові відомості про нього було оголошено 3 грудня 2015 року.

Swift — це мова програмування, яка компілює кілька парадигм в одну, розроблену Apple, щоб співіснувати з Objective C і мати більш стійку кодову базу. Swift було вперше продемонстровано на WWDC 2014. Мова створена за допомогою компілятора LLVM, який включено до шостої бета-версії Xcode. Безкоштовний посібник із програмування у Swift доступний у магазині iBooks.

Компілятор Swift є похідним від безкоштовного проекту на основі LLVM, який більш детально описано в наступному розділі. Swift використовує найкращі функції мов C і Objective-C, тому синтаксис знайомий розробникам, які вже їх освоїли, але він також відрізняється використанням автоматичного розподілу пам’яті та переповнення змінних і масивів, що підвищує стабільність. і безпечність коду. Крім того, програми Swift перетворюються на машинний код, що забезпечує високу продуктивність. Apple стверджує, що Swift має перевагу в швидкості над Objective-C: вони кажуть, що код Swift в 1,3 рази швидший. Замість збирача сміття Objective-C Swift використовує функції підрахунку посилань на об’єкти, а також оптимізацію, представлену LLVM, наприклад автовекторизацію.

Мова також сприяє численним сучасним методам програмування, включаючи закриття, узагальнене програмування, лямбда-вирази, кортежі та словники, швидкі операції над колекціями та елементи функціонального програмування. Основним призначенням Swift є створення програм для Mac, iPhone, Android і Apple Watch. Ці програми розроблено з використанням наборів інструментів Cocoa та Cocoa Touch. Крім того, Swift надає об’єктну модель, сумісну з мовою Objective-C. Рідний код Swift можна поєднувати з C і Objective-C в одному проекті. Swift тісно інтегрований у власне середовище Xcode для розробки, але його також можна викликати з командного рядка, що дає змогу використовувати його на платформах, відмінних від macOS, таких як Linux..

1.5 Середовище для розробки на iOS

Xcode є ключовим інструментом для розробників, що працюють у середовищі Apple, незалежно від того, чи розробляють вони програми для iOS, macOS, watchOS або tvOS. Це інтегроване середовище розробки, яке об'єднує в собі низку потужних інструментів, що полегшують розробку програмного забезпечення для цих платформ.

Перша версія Xcode, зазвичай відома як Xcode 1.0, була випущена у вересні 2003 року разом з оновленим набором інструментів для розробки програмного забезпечення для Mac OS X, включаючи інтегроване середовище розробки, компілятори та відлагоджувальний інтерфейс.

З тих пір Xcode пройшов значний шлях розвитку та ряд оновлень, що додали нові функції та покращення. Наприклад, в 2008 році з випуском iPhone SDK, Xcode почав підтримувати розробку програм для iOS, що відкрило шлях для створення додатків для iPhone та iPad.

Значним кроком для Xcode стала випуск Swift, нової мови програмування, яка була представлена на конференції розробників WWDC 2014. Swift став альтернативою мові Objective-C і зробив розробку програм для платформ Apple більш простою та ефективною.

Xcode також постійно оновлюється, додаючи нові функції та покращення для полегшення роботи розробників. Від початкових версій до сучасних, Xcode залишається ключовим інструментом для створення програмного забезпечення для всіх платформ Apple, забезпечуючи розробникам потужні інструменти для створення найкращих програм.

Основні функції Xcode

- Інтегроване середовище розробки (IDE): Xcode є повнофункціональним інтегрованим середовищем розробки для платформ Apple, таких як macOS, iOS, iPadOS, watchOS та tvOS. Це включає в себе інструменти для написання,

відлагодження та тестування коду, роботи з інтерфейсами користувача, а також управління версіями.

- Відлагодження коду: Xcode надає засоби для відлагодження коду, що дозволяють розробникам відстежувати виконання програми, встановлювати точки зупинки, переглядати значення змінних та виконувати інші операції для виявлення та виправлення помилок.
- Інтерфейс для роботи з інтерфейсами користувача: Xcode має вбудовані інструменти для створення і редагування інтерфейсів користувача за допомогою Storyboards та Interface Builder. Це дозволяє розробникам швидко створювати та налагоджувати UI своїх додатків.
- Інструменти для тестування: Xcode містить інструменти для проведення різних видів тестів, таких як юніт-тести, функціональні тести та взаємодійові тести. Це дозволяє розробникам впевнитися в якості свого коду та виявити помилки на ранніх етапах розробки.
- Інтеграція зі службами розробки: Xcode підтримує інтеграцію з різними сервісами розробки, такими як GitHub, Bitbucket та GitLab. Це дозволяє розробникам працювати зі своїми проектами та спільно робити з ними.
- Управління версіями: Xcode має вбудовані засоби для управління версіями коду через інтерфейс Git. Це дозволяє розробникам вести історію змін коду та працювати з різними гілками проекту.
- Підтримка мов програмування: Xcode підтримує різні мови програмування, такі як Swift, Objective-C, C++ та інші. Це дозволяє розробникам вибирати мову, яка найкраще відповідає їхнім потребам та вимогам проекту.
- Інтеграція з іншими інструментами: Xcode може бути інтегрований з різними зовнішніми інструментами для розробки та тестування програмного забезпечення, такими як CocoaPods, Carthage, Firebase та інші. Це дозволяє розробникам використовувати різноманітні інструменти для покращення їхнього робочого процесу.

1.6 Задачі для кваліфікаційного проєкту

1. Проаналізувати існуючі застосунки для сканування штрих кодів для мобільних пристроїв
2. Розробити архітектуру мобільного застосунку з визначенням нефункціональних вимог
3. Розробити та продумати дизайн застосунку для сканування штрих кодів
4. Дослідити актуальні та сучасні методи розробки мобільних застосунків від Apple
5. Розробити мобільний застосунок для сканування штрих коду
6. Провести тестування та адаптацію мобільного застосунку під різні моделі Iphone
7. Функціональність:
8. Можливість сканувати штрих код товару як з товару, так із вже готової фотографії
9. Ввід штрих коду вручну
10. Перегляд інформації про товар з таблицею порівняння цін
11. Перегляд історії пошуку товарів з можливістю повторного їх перегляду
12. Складання списку покупок

2 КОНЦЕПЦІЯ

2.1 Розробка концепції мобільного додатку

Жанр.

Даний мобільний застосунок можна віднести до жанру Утиліті, так як він буде слугувати лише інструментом для надання інформації про ціни.

Вік.

Вікове обмеження в 13+ років є досить умовним, так як при створені аккаунту Apple ID, є пункт про вік, в якому вказано що зареєструватися можна з дозвулу батьків чи опікунів дитини з тринадцятирічного віку або при досягнені 18 років.

Цільова аудиторія.

Цільовою аудиторією застосунку є платоспроможні покупці продуктів, які мають на меті зекономити власні кошті при закупівлі продуктів харчування.

Короткий опис застосунку.

Під час завантаження на екрані відображається смуга прогресу. Після завантаження застосунку, користувач потрапляє на головну сторінку додатку.

Опис головних функцій додатку

Після того як користувач потрапив на головну сторінку застосунку, він зможе скористатися наступними функціями:

- Сканер – При запуски цієї функції, відкривається камера, за рахунок якої можна зчитати штрих-код товару
- Пошук – Тут користувач може ввести номер штрих-коду вручну
- Галерея – Після того як користувач натисне на галерею, він зможе вибрати будь-яку із своїх фотографій які він робив або зберігав раніше, щоб відсканувати штрих-код
- Налаштування – перейшовши в цей пункт людина яка користується додатком зможе вибрати необхідні їй налаштування та оформити

преміум підписку в яку буде входити аналіз прихильності товарів за допомогою штучного інтелекту

- Історія пошуку – в цьому пункті людина зможе переглянути все що вона шукала раніше, та при необхідності знову переглянути усю інформацію про товар, перейшовши на карту товару

Оновлення.

Так як інформація про ціни отримується за рахунок API з бази данних то випускати оновлення кожного разу нам не потрібно, але якщо нам потрібно буде змінити якийсь функціонал чи дизайн, для цього потрібне буде повноцінне оновлення, так для роботи додатку не використовується сервер

2.2 Завдання мобільного застосунку

Головними завданнями проекту є:

- Спрощення аналізу цін товарів
- Пошук товару за допомогою штрих коду товару
- Дослідження розробки застосунку використовуючи технології розробки Apple
-

Спрощення аналізу цін товарів

Ціни на товари за останній час суттєво збільшилися, і багато українців прагнуть якось зекономити свої кошти. І одним із способом економії є аналіз цін. Більшість мережевих супермаркетів та магазинів вже мають свої застосунки в яких будь-хто може знайти ціну на кожен товар, але якщо потрібно саме порівняти ціни на товари, то треба зробити багато дій. Для цього в проекті і реалізовується функціонал який відразу в формі таблиці відображає ціни на товар з різних магазинів, завдяки чому проаналізувати ціни на товар стає більш простою та легкою задачею.

Пошук товару за допомогою штрих коду товару

Пошук товару за допомогою штрих коду є досить важливою складовою при покупці товарів. Так як в сучасному світі кількість різноманітних товарів лише зростає, такий спосіб отримання інформації про товар полегшує процес пошуку інформації про товари.

Однією з головних переваг цього функціоналу є підвищення прозорості та довіри до товару. Покупці можуть переконатися, що інформація про товар є точною і актуальною, що сприяє більшій впевненості у своєму виборі. Крім того, можливість отримати відгуки інших користувачів допомагає уникнути невдалих покупок і зменшує ризик розчарування.

Дослідження розробки застосунку використовуючи технології розробки Apple

Застосунок для дипломної роботи був розроблений за допомогою саме нативних інструментів та можливостей для розробки мобільних застосунків від компанії Apple. Під час написання коду використовуються фреймоворки та бібліотеки які пришвидшують написання коду.

3 ОСОБЛИВОСТІ ТА МЕТОДИ РОЗРОБКИ

3.1 Архітектура iOS

Архітектура програмного забезпечення - це спосіб організації програми або обчислювальної системи, який визначає абстракцію її елементів на певній стадії розробки. Система може мати кілька рівнів абстракції і пройти через декілька фаз роботи, кожна з яких може мати свою власну архітектуру.

Дослідження архітектури програмного забезпечення спрямоване на визначення оптимального розбиття системи на частини, їх взаємодію, передачу інформації між ними, а також на розвиток окремих компонентів та способів формального або неформального опису цих відносин.

Архітектура має бути побудована так, щоб найкращим чином відповідати вимогам системи, яка створюється, відповідно до принципу "форма слідує функції".

Архітектура iPhone os складається з декількох різних програмних шарів, кожен з яких надає програмні фреймворки для розробки додатків, які працюють поверх операційної системи. Розглянемо ці рівні:

- Cocoa Touch Layer

Цей рівень знаходиться на вершині стека iPhone os і містить фреймворки, які найчастіше використовуються розробниками додатків для iPhone.

- Media Service Layer

Цей рівень надає ОС аудіо, відео, анімацію та графічні можливості. Медіа-шар включає в себе ряд фреймворків, які можуть бути використані при розробці додатків для iPhone.

- Core Services Layer

Цей шар забезпечує більшу частину фундаменту, на якому побудовані вищезазначені шари

- Core OS Layer

Це нижній рівень стеку iPhone os, який знаходиться безпосередньо на апаратному забезпеченні пристрою. Цей рівень надає різноманітні послуги, включаючи низькорівневу мережу, доступ до зовнішніх аксесуарів та звичайні базові послуги операційної системи, такі як управління пам'яттю, файловою системою та потоками.

3.1.1 MVC

MVC - це патерн, який розділяє додаток на три основні компоненти: модель (Model), представлення (View) і контролер (Controller). Кожен з цих компонентів має свої обов'язки.

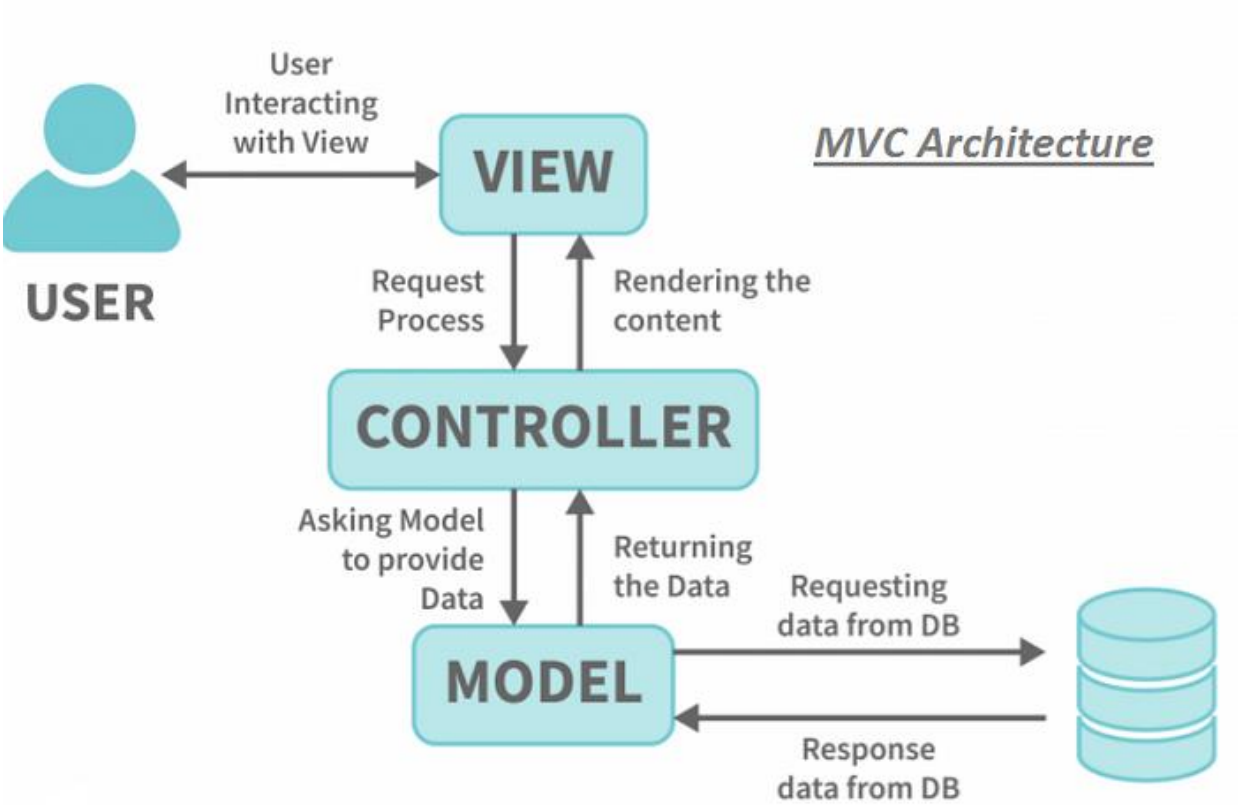


Рис. 3.1 Архітектура MVC

Модель

Модель відповідає за дані та бізнес-логіку додатку. Вона містить дані, які використовує додаток, і надає методи доступу до цих даних та маніпулювання ними. Модель також відповідає за виконання будь-яких обчислень або іншої обробки, необхідної для генерації цих даних. У розробці iOS модель часто реалізується у вигляді набору об'єктів даних або рівня даних.

Вигляд (View)

Представлення (View) відповідає за представлення користувацького інтерфейсу програми. Він визначає, як дані з моделі будуть представлені користувачеві, наприклад, за допомогою міток, текстових полів або зображень. Представлення також відповідає за обробку даних, введених користувачем, таких як натискання кнопок або жести.

Контролер (Controller)

Контролер виступає посередником між моделлю та представленням. Він отримує користувацьке введення від Представлення, взаємодіє з моделлю для виконання будь-яких необхідних операцій і оновлює Представлення, щоб відобразити зміни в моделі. У розробці iOS Контролер часто реалізується як контролер представлення.

Чому MVC важливий?

MVC надає ряд переваг для розробки додатків.

Розділяючи додаток на ці три окремі компоненти, MVC сприяє модульності, що полегшує тестування та підтримку додатку.

Розподіл обов'язків також дозволяє розробникам працювати над певними частинами програми, не впливаючи на інші частини. Це полегшує додавання нових функцій або модифікацію існуючих, не порушуючи роботу інших частин програми.

MVC також сприяє багаторазовому використанню. Оскільки компоненти розділені і мають конкретні обов'язки, їх можна повторно використовувати в інших частинах програми або в інших додатках. Це може заощадити час розробки та зменшити складність програми.

3.1.2 MVP

Model-View-Presenter (MVP) - це похідний патерн MVC, який також розділяє додаток на три компоненти, але робить це дещо по-іншому.

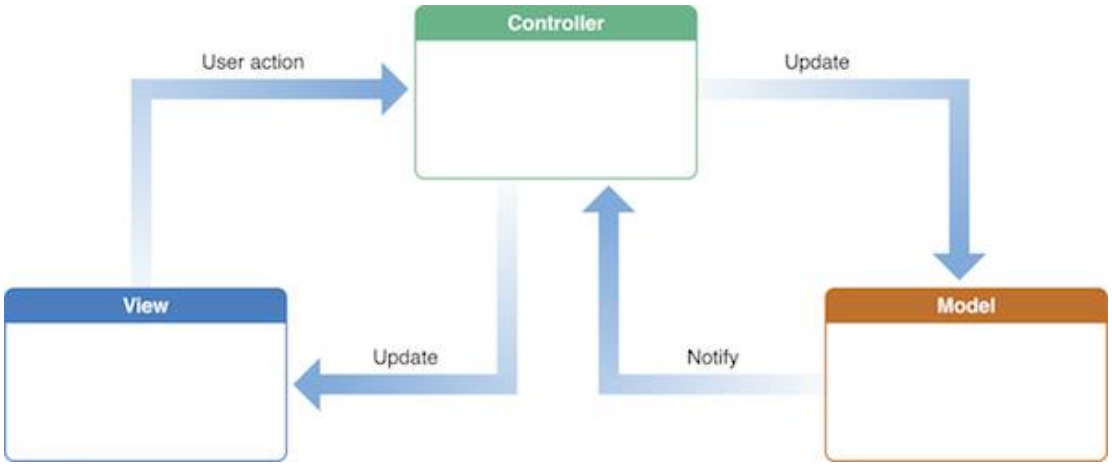


Рис.3.2 Архітектура MVP

Модель

Представляє дані та бізнес-логіку додатку. Це те ж саме, що і в MVC.

Вигляд (View)

Відображає дані (модель) і сповіщає доповідача про дії користувача.

Презентатор(Presenter)

Діє як посередник між представленням і моделлю. Він отримує дані з моделі та форматує їх для представлення.

Основні відмінності

Основна відмінність між MVC і MVP полягає в тому, як вони обробляють вхідні дані користувача і потік управління.

- У MVC контролер обробляє вхідні дані користувача, маніпулюючи моделлю та оновлюючи представлення у відповідь. Представлення є пасивним, тобто воно не робить нічого, окрім відображення того, що йому вказує контролер.
- У MVP представлення відіграє активну роль в обробці користувацького вводу, яку воно делегує доповідачу. Потім презентатор оновлює представлення новими даними, які він отримує і

форматує з моделі. Такий підхід відокремлює Представлення від Моделі, що полегшує тестування і обслуговування.

3.1.3 MVVM

Model-View-ViewModel (MVVM) - це патерн проектування, який широко використовується в розробці додатків для iOS для створення чистого, зручного для обслуговування і тестування коду. MVVM відокремлює користувацький інтерфейс додатку (View) від базових даних (Model) і вводить проміжний компонент ViewModel для управління логікою представлення.

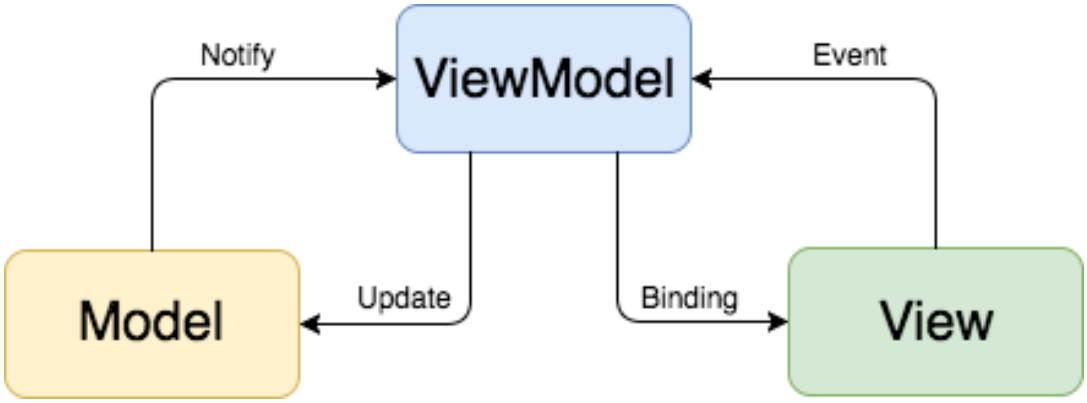


Рис. 3.3. Архітектура MVVM

Модель (Model)

Це місце, де відбуваються всі операції зі зміною даних. Тут ви розмістите логіку локального зберігання даних, мережу тощо. Також, якщо ви використовуєте доменний рівень у вашій архітектурі, це місце для UseCases.

Вигляд (View)

Все, що бачить користувач. Він повинен спостерігати за станом, який відображає ViewModel, і оновлювати себе на основі цього. Він також запускає функції у ViewModel на основі подій, таких як натискання кнопок, відкриття екрану тощо.

ViewModel

Проміжна ланка між View і Model. Вона не повинна знати про View і залежить тільки від Model.

Відповідає за логіку інтеракторів (логіку, специфічну для додатку) та експонування стану до View. Якщо взаємодія з користувачем вимагає бізнес-логіки або зміни/видобування даних, вона викликає необхідні функції з Model.

3.1.4 Viper

Viper - це патерн дизайну, який реалізує парадигму "розділення турбот". Як і MVP або MVC, він дотримується модульного підходу. Одна функція - один модуль. Для кожного модуля VIPER має п'ять різних класів з різними ролями. Жоден клас не виходить за рамки свого єдиного призначення.

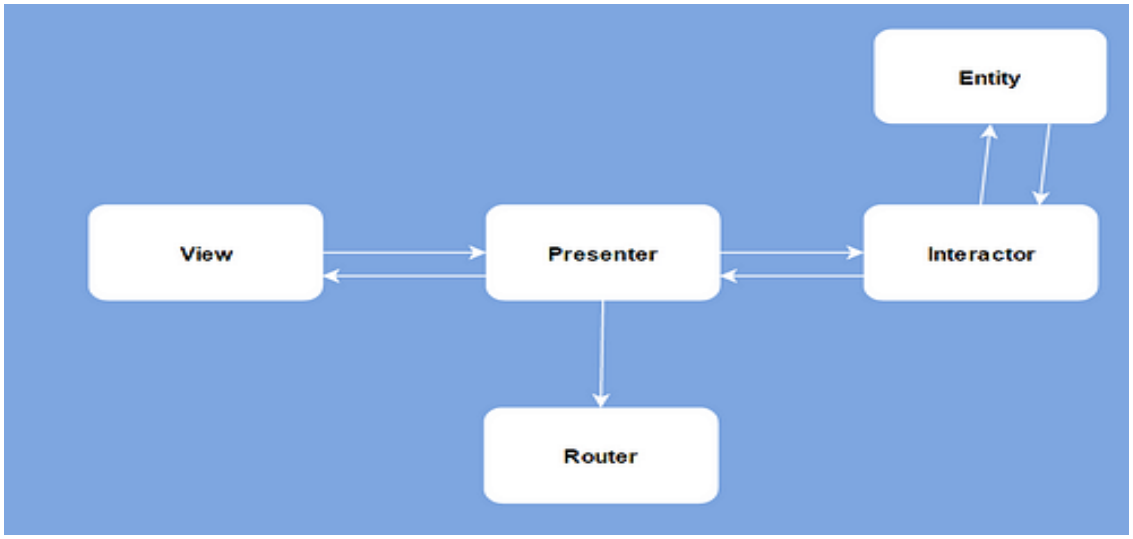


Рис. 3.4 Архітектура Viper

Вигляд (View)

Клас, який містить весь код для показу інтерфейсу програми користувачеві та отримання його відповідей. При отриманні відповіді View сповіщає про це ведучого.

Презентатор(Presenter)

Presenter: Ядро модуля. Він отримує відповідь користувача від View і працює відповідно до неї. Єдиний клас, який взаємодіє з усіма іншими компонентами. Викликає маршрутизатор для побудови каркасу, інтерактив для отримання даних (мережеві виклики або локальні виклики даних), вигляд для оновлення інтерфейсу.

Інтерактив (Interactor)

Відповідає за бізнес-логіку програми, наприклад, якщо бізнес-логіка залежить від здійснення мережевих дзвінків, то за це відповідає інтерактив.

Маршрутизатор (Router)

Виконує каркасний фреймінг. Вислуховує від доповідача, який екран показати, і виконує його.

Сутність (Entity)

Містить класи простої моделі, що використовуються інтерактором.

3.2 Принципи SOLID

SOLID - це принципи, які дають змогу вам писати чудовий код без зайвих зусиль.

П'ять принципів SOLID:

- Single Responsibility - Принцип єдиної відповідальності;
- Open-Closed - Принцип відкритості/закритості;
- Liskov Substitution - Принцип підстановки Барбари Лісков;
- Interface Segregation - Принцип поділу інтерфейсів;
- Dependency Inversion - Принцип інверсії залежностей.

Принцип єдиної відповідальності (Single-responsibility principle)

Поширеною помилкою при розробці є створення класів, які виконують кілька обов'язків. Кожен клас повинен мати єдине, чітко визначене призначення. Наприклад, уникайте створення класів, що відповідають за мережеві виклики, обробку функцій, передачу даних ViewController'ам тощо.

Принцип відкритості/закритості (Open-Closed)

Кожен клас, структура тощо має бути відкритим для розширення, але закритим для модифікації. Swift має потужні можливості, які відповідають цьому принципу, однією з яких є функція розширення. Наприклад, якщо ми хочемо додати функціональність до нативної структури String, яку надає Swift, але не маємо доступу до модифікації коду рядка безпосередньо, Swift пропонує нам

розширення, щоб додати цю функціональність. Працюючи з кастомними типами, ми маємо можливість змінювати код. Однак, важливо завжди пам'ятати про принцип відкритості/закритості, уникати типових помилок і бути обережними при роботі з кастомними типами. Протоколи також дотримуються принципу відкритості/закритості.

Принцип підстановки Барбари Ліскoв (Liskov Substitution)

Принцип підстановки Ліскова (LTSY) стверджує, що коли ми успадковуємо від базового класу, підклас не повинен змінювати поведінку функцій базового класу. Користувачі можуть використовувати як функціональність базового класу, так і дочірнього класу, який успадковує поведінку базового класу.

Принцип поділу інтерфейсів (Interface Segregation)

Принцип розділення інтерфейсів стверджує, що користувачі не повинні залежати від інтерфейсів або функціональності, які їм не потрібні. Він не рекомендує створювати надто складні або важкі для розуміння інтерфейси. Інтерфейс повинен бути простим і не містити складних елементів, які користувач не може зрозуміти або використати.

Принцип інверсії залежностей ((Dependency Inversion Principle)

Принцип інверсії залежностей (DIP) стверджує, що високорівневі модулі не повинні залежати від низькорівневих модулів. Натомість вони обидва повинні залежати від абстракцій. Іншими словами, ви повинні залежати від інтерфейсів, а не від реалізацій.

3.3 Управління пам’яттю ARC

Автоматичний підрахунок посилань (ARC) - це функція керування пам'яттю у Swift, яка автоматично керує розподілом та звільненням об'єктів у вашому коді. Це допомагає уникнути витоків пам'яті та ефективно керувати пам'яттю, не вимагаючи ручного керування пам'яттю.

Як працює ARC

Щоразу, коли ви створюєте новий екземпляр класу, ARC виділяє ділянку пам'яті для зберігання інформації про цей екземпляр. Ця пам'ять містить інформацію про тип екземпляра, а також значення всіх збережених властивостей, пов'язаних з цим екземпляром.

Крім того, коли екземпляр більше не потрібен, ARC звільняє пам'ять, яку використовував цей екземпляр, щоб її можна було використати для інших цілей. Це гарантує, що екземпляри класів не займатимуть місце у пам'яті, коли вони більше не потрібні.

Однак, якщо ARC вивільнить екземпляр, який все ще використовується, ви більше не зможете отримати доступ до властивостей цього екземпляра або викликати його методи. Дійсно, якщо ви спробуєте отримати доступ до цього екземпляра, ваша програма, скоріш за все, завершить роботу аварійно.

Щоб переконатися, що екземпляри не зникають, поки вони ще потрібні, ARC відстежує, скільки властивостей, констант і змінних посилаються на кожен екземпляр класу. ARC не вивільнятиме екземпляр доти, доки існує хоча б одне активне посилання на цей екземпляр.

Щоб зробити це можливим, щоразу, коли ви присвоюєте екземпляр класу властивості, константі або змінній, ця властивість, константа або змінна створює сильне посилання на екземпляр. Посилання називається "сильним", тому що воно міцно утримує екземпляр і не дозволяє йому бути дерозподіленим доти, доки існує це сильне посилання.

Сильні посилання

Сильне посилання міцно утримує об'єкт, тобто запобігає його вивільненню. Доки існує хоча б одне сильне посилання на об'єкт, він залишатиметься в пам'яті.

Слабкі та невласні посилання

Слабкі та невласні посилання використовуються для переривання циклів сильних посилань. Слабке посилання дозволяє об'єкту, на який воно посилається, стати нульовим, коли об'єкт, на який воно посилається, буде звільнений.

Неприналежне посилання передбачає, що об'єкт, на який посилається, завжди матиме дійсне значення.

Сильні цикли посилань

Сильний цикл посилань виникає, коли два або більше об'єктів мають сильні посилання один на одного. Це може запобігти вивільненню обох об'єктів, що може призвести до витоку пам'яті. Слабкі та невласні посилання можуть бути використані для розриву сильних циклів посилань.

Автоматичне керування пам'яттю

ARC автоматично вставляє код керування пам'яттю під час компіляції. Цей код гарантує, що об'єкти будуть звільнені, коли на них більше не буде посилань, що допомагає запобігти витоку пам'яті та підвищити продуктивність вашого коду.

Переваги використання ARC

- Ефективність: ARC допомагає підвищити ефективність вашого коду, автоматично вивільняючи об'єкти, коли на них більше не посилаються.
- Простота: ARC усуває необхідність ручного керування пам'яттю, що може зробити ваш код простішим і легшим в обслуговуванні.
- Надійність: ARC допомагає запобігти витокам пам'яті, які можуть спричинити збої та інші проблеми у вашому коді.

3.4 Життєвий цикл застосунку

Протягом свого життєвого циклу додаток для iOS проходить кілька станів. Додаток може бути активним і реагувати на потреби користувача, працювати у фоновому режимі або бути закритим користувачем чи операційною системою.

Коли додаток змінює свій стан, він надсилає сповіщення, і ми можемо реалізувати кастомну функціональність для різних станів у межах їхніх відповідних функцій.

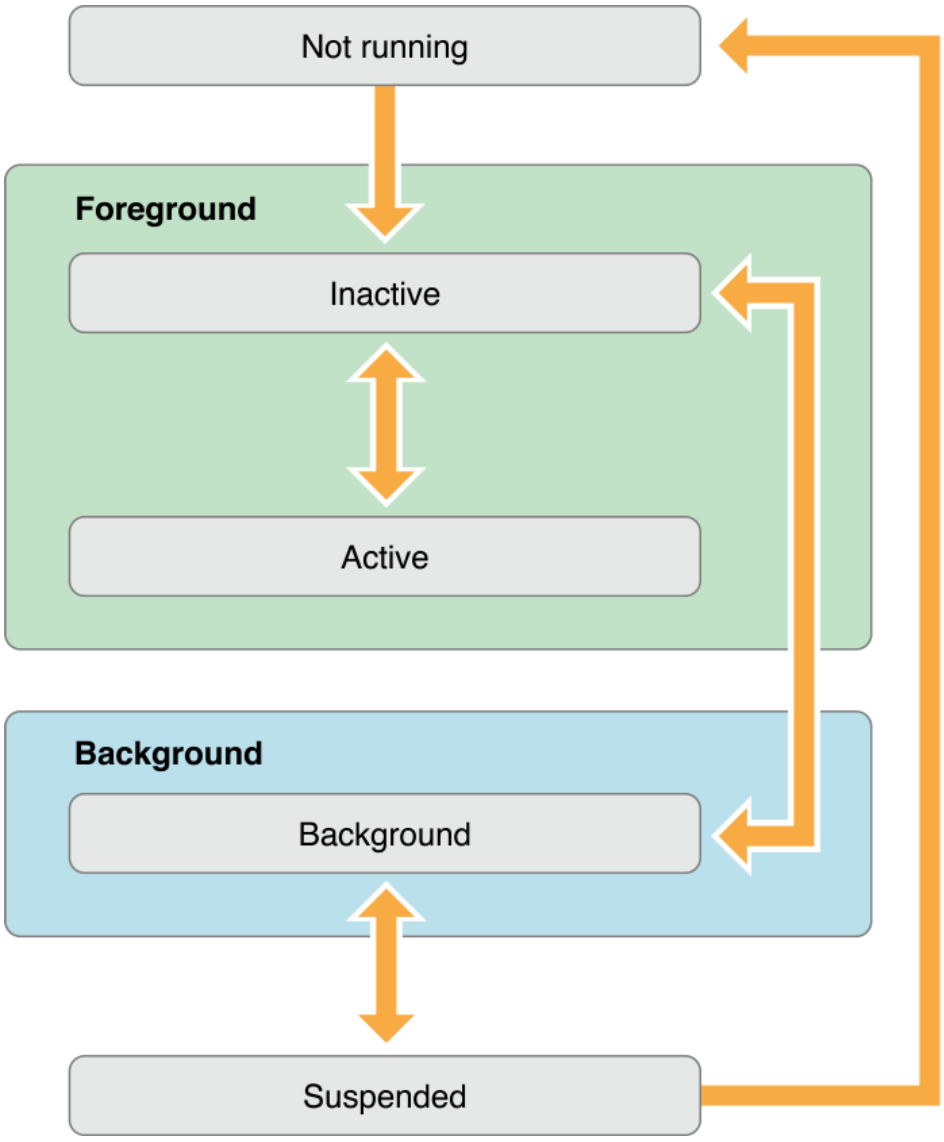


Рис. 3.5 Схема переходу застосунку у різні стани

Існує п'ять різних станів, в яких може перебувати програма:

- Не працює
- Неактивний
- Активний
- У фоновому режимі
- Призупинено

Неактивний

Додаток працює на передньому плані, але не отримує подій. Додаток для iOS може бути переведений в неактивний стан, наприклад, при отриманні дзвінка або

SMS-повідомлення. Користувач наразі не бере активної участі в роботі з нашим додатком. Ймовірно, додаток змінює стан.

Такий стан може бути і в деяких інших випадках, наприклад, коли ми запускаємо неактивний додаток, він спочатку переходить в цей стан, а потім з нього переходить в активний.

Активний

Програма працює на передньому плані і отримує події. Перехідний стан. І користувач активно використовує наш додаток.

Фоновий

Додаток працює у фоновому режимі та виконує код. Користувач зараз використовує інші додатки, але наш додаток продовжує працювати у фоновому режимі і може отримувати події та виконувати код. У цьому стані ми повинні виконувати короткі завдання і повертати управління назад до операційної системи.

Коли додаток збирається бути призупиненим, він також переходить у цей стан на невеликий проміжок часу. У цьому стані програма залишається у фоновому режимі і виконує код.

Призупинений

Програма знаходиться у фоновому режимі, але код не виконується. Додаток все ще знаходиться в пам'яті, але вона заморожена, і операційна система, ймовірно, видалить наш додаток в найближчому майбутньому, і додаток буде в непрацюючому стані.

3.5 SwiftUI

SwiftUI

SwiftUI - це сучасний декларативний фреймворк інтерфейсу користувача, представлений компанією Apple на WWDC 2019. Він побудований на основі мови програмування Swift і забезпечує потужний, але простий спосіб проектування користувацьких інтерфейсів для iOS, macOS, watchOS і tvOS. Декларативне програмування - це парадигма програмування, яка фокусується на описі того, як

повинен виглядати кінцевий результат, а не на покроковому описі того, як його досягти. У SwiftUI ви описуєте свій інтерфейс з точки зору його стану та компонування. Фреймворк автоматично оновлює ієрархію представлень і оптимізує продуктивність, коли стан додатку змінюється. SwiftUI має кілька ключових переваг:

- **Спрощений синтаксис**

Код SwiftUI часто коротший і легший для читання порівняно з UIKit, що робить його простішим для розуміння і підтримки розробниками.

- **Вбудовані функції**

SwiftUI включає вбудовану підтримку темного режиму, доступності, локалізації та інших сучасних елементів інтерфейсу, що спрощує їх інтеграцію у ваш додаток.

- **Попередній перегляд**

Функція попереднього перегляду в SwiftUI дозволяє розробникам бачити зміни в коді в реальному часі, не перероблюючи і не запускаючи додаток, що значно прискорює робочий процес розробки.

- **Багатоплатформенна підтримка**

SwiftUI дозволяє створювати інтерфейси для iOS, macOS, watchOS і tvOS з меншою кількістю специфічного для кожної платформи коду, забезпечуючи більш узгоджений зовнішній вигляд на різних пристроях і операційних системах.

Тим не менш, SwiftUI також має деякі обмеження, зокрема:

- **Сумісність**

SwiftUI вимагає iOS 13 або новішої версії, а це означає, що ви не зможете націлити свій додаток на старіші пристрої або операційні системи.

- **Зрілість**

Незважаючи на те, що SwiftUI постійно розвивається, він є відносно новим порівняно з UIKit, і все ще має меншу базу знань, менше ресурсів та певні обмеження фреймворку, які потрібно подолати.

UIKit

UIKit - це добре зарекомендований, обов'язковий фреймворк інтерфейсу користувача від Apple, який використовується для розробки додатків для iOS з моменту запуску iPhone. Він побудований з використанням мов програмування Objective-C та Swift і надає широкий спектр компонентів інтерфейсу, обробки подій та інструментів малювання для створення насичених та інтерактивних додатків для iOS.

Імперативне програмування - це процедурний підхід, коли ви пишете код, який крок за кроком описує, як створювати та керувати компонентами інтерфейсу і як вони взаємодіють з базовою системою. UIKit керує компонентами інтерфейсу за допомогою ієрархічної структури подання, обробляє взаємодію з користувачем і забезпечує зворотній зв'язок за допомогою анімації або модальних переходів подання. UIKit має кілька переваг:

- **Зрілість**

UIKit існує вже давно і має перевірену репутацію стабільності, продуктивності та надійності.

- **Вичерпні ресурси**

Завдяки своїй довгій історії, UIKit може похвалитися великою кількістю навчальних посібників, документації та сторонніх бібліотек, що полегшує пошук рішень і підтримку практично для будь-якої проблеми або вимоги.

- **Повний контроль над інтерфейсом користувача**

UIKit забезпечує тонкий контроль над елементами інтерфейсу та обробкою подій, що дозволяє розробникам створювати повністю кастомізовані та складні користувацькі інтерфейси.

- **Сумісність**

Оскільки UIKit вже багато років є основою розробки додатків для iOS, він сумісний з більш ранніми версіями iOS, що дозволяє розробникам орієнтуватися на ширшу аудиторію зі своїми додатками.

Але у використанні UIKit є і деякі недоліки:

- **Складність коду**

UIKit зазвичай вимагає більше коду для досягнення того ж результату, що і SwiftUI, що може вплинути на час розробки і, як наслідок, на підтримку коду.

- **Швидкість розробки**

Через більш багатослівний код та імперативний підхід, розробка UIKit може бути повільнішою порівняно зі SwiftUI, особливо для складних або кастомних користувацьких інтерфейсів.

4 КРОКИ РОЗРОБКИ ДОДАТКА

4.1 Екран «Головна»

Початковий екран - це екран, у який користувач потрапляє відразу після входу до додатку. Цей екран водночас є і головним меню усього додатку. На цьому екрані є можливість у користувача потрапити на екрани, які розділені на певні категорії для доступного знаходження інформації, яка цікавить користувача додатку.

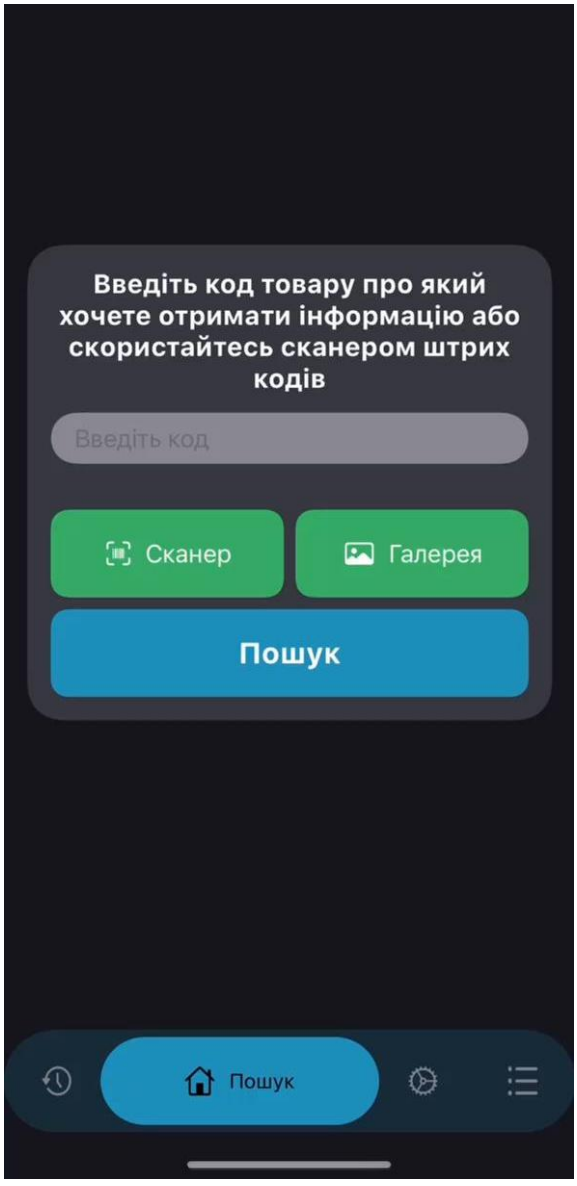


Рис. 4.1 головний екран застосунку

Екран створений з використанням кількох основних елементів. Основна частина екрану займає поле введення коду товару, під яким розташовані кнопки для сканування штрих-кодів, вибору зображень з галереї та кнопка пошуку. Таке розташування забезпечує зручний доступ до основних функцій додатку.

Рис. 4.2 Код головної сторінки застосунку

Дизайн головного екрану базується на використанні простих і зрозумілих елементів інтерфейсу, які дозволяють користувачеві легко і швидко орієнтуватися в додатку.

4.2 Екран «Історія»

Екран історії пошуку є важливою частиною додатку, що дозволяє користувачеві переглядати раніше здійснені пошуки. Цей екран надає швидкий доступ до результатів попередніх пошуків, що є зручним для повторного перегляду товарів без необхідності повторного введення даних.

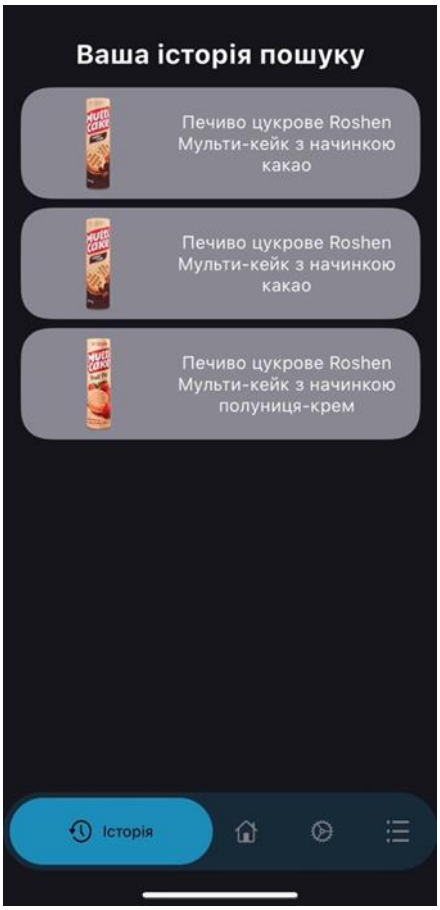


Рис. 4.3 Екран історії пошуку

Основна частина екрану містить список раніше знайдених товарів, кожен з яких представлений у вигляді окремої клітинки. Клітинки містять зображення товару, його назву та короткий опис, що дозволяє швидко знайти потрібний товар у списку.

Список результатів пошуку. У цьому списку відображаються всі товари, які були знайдені користувачем раніше. Кожен запис у списку містить зображення товару та його назву, що робить його легко впізнаваним.

Кнопка "Історія". Розташована внизу екрана, ця кнопка дозволяє швидко перейти до екрану історії пошуку з будь-якого іншого розділу додатку. Кнопка підсвічується, коли користувач знаходиться на цьому екрані, що допомагає орієнтуватися в додатку.

```
struct HistoryScreen: View {
    @ObservedObject var viewModel = HistoryViewModel()
    var body: some View {
        NavigationView {
            ZStack {
                Color(hex: "16161F")
                    .ignoresSafeArea()
                VStack {
                    Text("Ваша історія пошуку")
                        .foregroundColor(.white)
                        .font(.custom("SFProText-Bold", size: 24))

                    if viewModel.historyProducts.isEmpty {
                        Spacer()
                        Text("Ви ще не використовували пошук товарів")
                            .foregroundColor(.white)
                            .font(.custom("SFProText-Bold", size: 16))
                    } else {
                        ScrollView {
                            ForEach(viewModel.historyProducts, id: \.self) { product in
                                RoundedRectangle(cornerRadius: 25)
                                    .foregroundColor(Color(hex: "949398"))
                                    .frame(height: 100)
                                    .overlay {
                                        NavigationLink(destination:
ResultHistory(selectedProduct: product)) {
                                            HStack {
                                                Spacer()
                                                Image(product.photoName)
                                                    .resizable()
                                                    .scaledToFit()
                                                    .padding(5)
                                                Spacer()
                                                Text(product.name)
                                                    .foregroundColor(.white)
                                                    .font(.custom("SFProText-Regular", size:
16))
                                                Spacer()
                                            }
                                        }

                                    }
                            }
                        }
                    }
                    Spacer()
                }
                .padding()
            }
            .navigationBarHidden(true)
            .onAppear {
                viewModel.historyProducts = viewModel.loadProducts()
            }
        }
    }
}
```

Рис. 4.4 Код до екрану "Історія"

Екран історії пошуку розроблений з акцентом на зручність користувача. Логічне розташування елементів і використання візуальних підказок роблять цей екран інтуїтивно зрозумілим. Завдяки цьому користувач може легко повернутися до раніше знайдених товарів та переглянути їх деталі без необхідності повторного введення інформації.

4.3 Екран «Товару»

Екран інформації про товар є найважливою частиною додатку, що надає детальну інформацію про конкретний товар, включаючи його зображення, опис, ціну та рейтинг у різних магазинах. Цей екран забезпечує користувача всією необхідною інформацією для прийняття рішення про покупку.

Рис. 4.5 Екран інформації про товар

Основні елементи екрану:

- Зображення товару.

Велике зображення товару розташоване у верхній частині екрану, що дозволяє користувачеві легко впізнати товар.

- Назва товару.

Назва та короткий опис товару відображаються під зображенням, надаючи базову інформацію про продукт.

- Ціна та рейтинг.

У центральній частині екрану розташована таблиця з цінами та рейтингами товару в різних магазинах. Це дозволяє користувачеві швидко порівняти ціни та оцінити якість товару на основі відгуків інших користувачів. У таблиці наведено такі дані:

- Назва магазину (Сільпо, Фозі, Фора)
- Ціна товару в кожному з магазинів
- Рейтинг товару з зазначенням кількості оцінок

- Рекомендовані товари. Під таблицею з цінами та рейтингами розташована секція з рекомендованими товарами, які можуть зацікавити користувача. Це дозволяє легко знайти подібні або пов'язані товари, розширюючи вибір користувача.

```
struct Result: View {
    @ObservedObject var viewModel: SearchViewModel
    @AppStorage("fullVersion") private var fullVersion: Bool = false
    @AppStorage("userName") private var userName: String = ""

    @State private var randomNumbers: [String: Int] = [:]
    var body: some View {
        NavigationView {
            ZStack {
                Color(hex: "16161F")
                    .ignoresSafeArea()
                VStack {
                    HStack {
                        Button {
                            print("kasfcklasmlkamsclk")
                            withAnimation(.easeInOut(duration: 0.5)) {
                                viewModel.selectedProduct = Product(barcode: "", name: "",
photoName: "", prices: [:], ratings: [:], notes: [])
                            }

                        } label: {
                            Image(systemName: "arrowshape.turn.up.backward.fill")
                        }
                        Spacer()
                    }
                    .padding(.horizontal)
                    Image(viewModel.selectedProduct.photoName)
                        .resizable()
                    ScrollView {
                        VStack {
                            Text(viewModel.selectedProduct.name)
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-Bold", size: 24))
                            HStack {
                                VStack(alignment: .leading) {
                                    Text("Ціна:")
                                        .foregroundStyle(.white)
                                        .font(.custom("SFProText-Bold", size: 22))
                                    ForEach(viewModel.selectedProduct.prices.sorted(by: {
$0.key < $1.key }), id: \.key) { store, price in
                                        HStack {
                                            Button {
                                                viewModel.showStoresOnMap(store: store)
                                            } label: {
                                                Text(store)
                                                    .foregroundStyle(.white)
                                                    .font(.custom("SFProText-Regular", size:
20))
```

Рис. 4.6 Частина коду до екрану «Товар»

Дизайн цього екрану базується на сучасних принципах UI/UX, що забезпечує зручність користувачів. Всі елементи розташовані логічно та інтуїтивно зрозуміло, що робить процес отримання інформації про товар максимально простим і швидким.

4.4 Екран «Налаштування»

Екран налаштувань є ключовим компонентом додатку, який дозволяє користувачеві персоналізувати свій досвід використання. На цьому екрані користувач може змінити особисті дані, налаштувати звукові сповіщення, активувати преміум підписку та багато іншого.

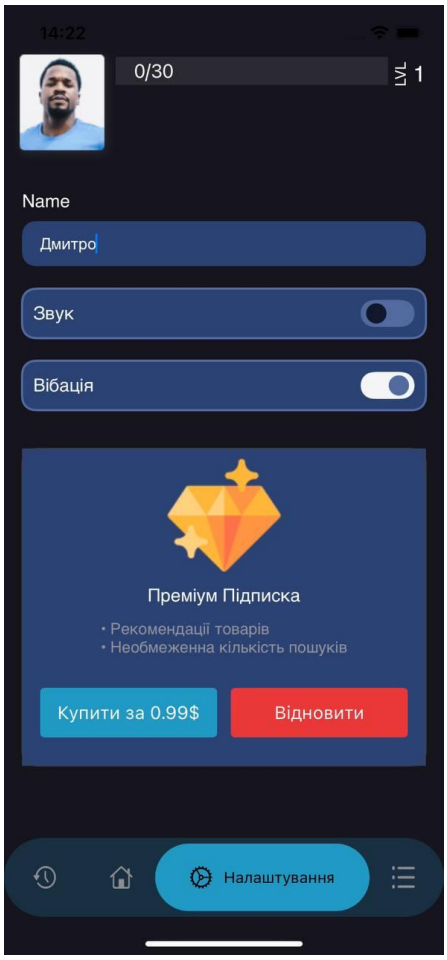


Рис. 4.7 Екран налаштувань.

Основні елементи екрану:

- Профіль користувача.

У верхній частині екрану знаходиться фото користувача та індикатор рівня, який відображає прогрес. Поле з іменем дозволяє змінювати ім'я користувача.

- Преміум підписка.

Секція для активації або поновлення преміум підписки. Преміум підписка надає доступ до додаткових функцій, таких як рекомендації товарів та необмежена кількість пошуків. Є кнопки для покупки (з позначкою ціни) та для поновлення підписки.

Завдяки цьому екрану додаток забезпечує можливість персоналізації користувацького досвіду, що підвищує загальне задоволення від користування додатком та робить його більш гнучким для різних категорій користувачів.

4.5 Екран «Список покупок»

Екран списку покупок є важливою частиною додатку, що дозволяє користувачам створювати та редагувати список необхідних для покупки товарів. Цей екран забезпечує зручний спосіб організації покупок і допомагає користувачам слідкувати за тим, що їм потрібно придбати.

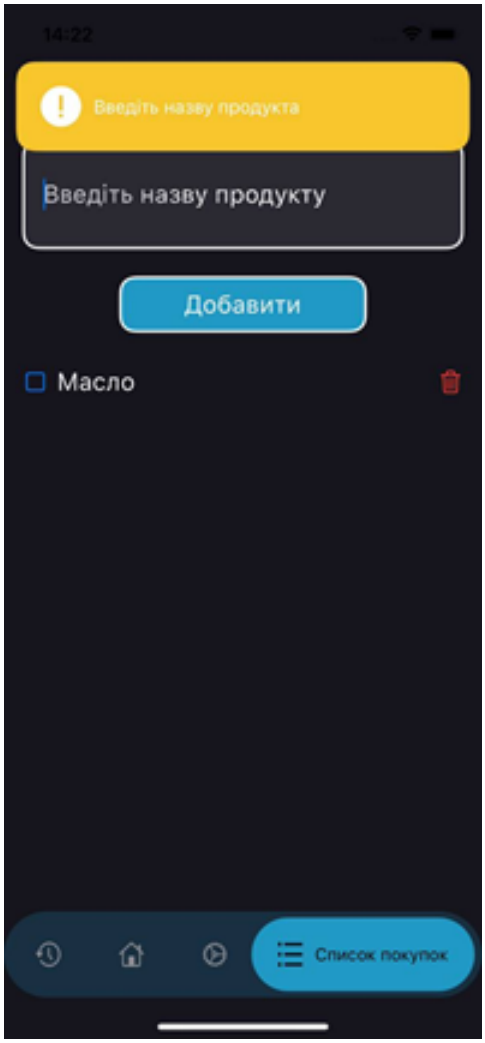


Рис. 4.8 Екран списку покупок

Основні елементи екрану:

- Поле введення назви продукту.

Розташоване у верхній частині екрану, це текстове поле дозволяє користувачеві ввести назву товару, який він хоче додати до списку покупок. Підказка "Введіть назву продукту" допомагає користувачеві зрозуміти, що потрібно зробити.

- Кнопка "Додати".

Після введення назви товару користувач може натиснути цю кнопку для додавання продукту до списку покупок. Кнопка розташована під полем введення і має виразний колір, що полегшує її знаходження.

- Список товарів.

Під кнопкою додавання розташований список вже доданих товарів. Кожен товар у списку відображається з можливістю позначення його як придбаного (чекбокс) та видалення (значок смітника). У прикладі в списку є товар "Масло".

- Кнопка "Список покупок".

Розташована внизу екрану, ця кнопка дозволяє швидко перейти на екран списку покупок з будь-якого іншого розділу додатку. Кнопка підсвічується, коли користувач знаходиться на цьому екрані, що допомагає орієнтуватися в додатку.

Завдяки цьому екрану, додаток забезпечує користувачам можливість швидко і легко створювати список необхідних продуктів, що значно спрощує процес планування покупок і допомагає не забути придбати потрібні товари.

5 ТЕСТУВАННЯ ТА АДАПТАЦІЯ ДОДАТКУ

Середовище розробки XCode має консоль журналу, інтегровану в набір інструментів, ця консоль сповіщає розробника в режимі реального часу. Збір компонентів проекту відбувається кожного разу як розробник натискає кнопку «Побудувати» в середовищі. Усі помилки та застережні повідомлення в коді відображаються розробнику. Це стає очевидним майже одразу після написання блоку коду, оскільки Xcode автоматично компілює код без помилок і надає конкретні поради. Крім того, контекст навколишнього середовища вказує на місце помилки, виділивши його певним кольором і якщо це синтаксична помилка, він може запропонувати вирішення проблеми, натиснувши кнопку «Виправити», яка з'явиться.

Якщо розробник пише синтаксично правильний код, але має запитання щодо логіки, є кілька способів зрозуміти помилку: print, break point, debugPrint, thread.

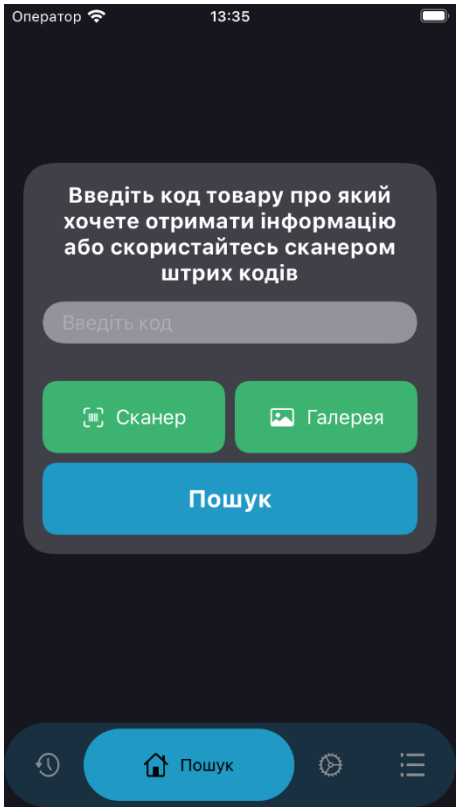
print - Коли компілятор досягає номера поля, де записане print, деяка інформація виводиться на консоль як текст. Користувач може написати, що виведе команда, а також може додати змінні до тіла команди.

break point — це інструмент налагодження, який дозволяє призупинити виконання програми до певного моменту. Коли код зупиняється, розробник може бачити, коли і де існують дані у змінній. Ви також можете переглядати дані у змінних на кожному кроці компіляції програми.

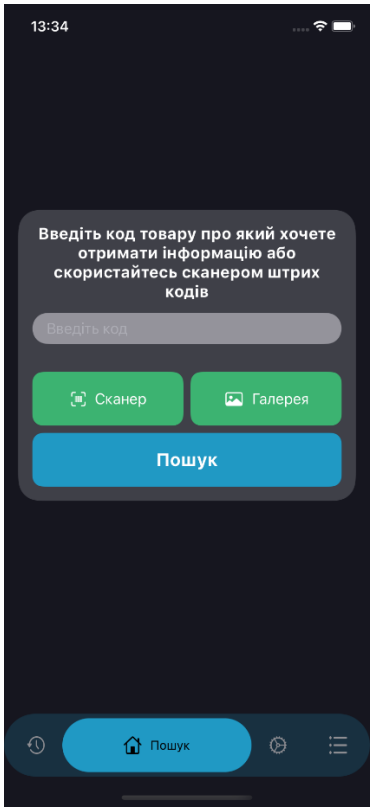
debugPrint - записує текстове представлення зазначеного елемента, яке найбільше підходить для налагодження, у стандартний вивід.

thread - це найпростіший спосіб уникнути проблем. Основна ідея полягає в тому, що всі види та елементи інтерфейсу не повинні бути закриті іншими елементами в основній черзі.

Щороку Apple представляє нову модель iPhone з екраном іншого розміру, тому розробники програмного забезпечення повинні переконатися, що їх цифровий продукт можна використовувати як на маленьких, так і на великих. В результаті під час створення дипломного проекту було адаптовано інтерфейс до різних моделей IPhone.



IPhone 8



IPhone 15

ВИСНОВКИ

Під час створення програмного застосунку у кваліфікаційній роботі були пройдені усі етапи. Ними стали:

1. Проведено пошук та аналіз подібних застосунків таких як Foodstr та Costless, QRbot
2. Була розроблена архітектура мобільного застосунку для платформи IOS
3. Були визначені функціональні та не функціональні вимоги до застосунку для сканування штрих коду
4. Були досліджені методи розробки програмного застосунку за допомогою методів Apple
5. Були досліджені інструменти для розробки застосунків а саме SwiftUI, CarBode, Lottie-ios, SwiftMessages, SwiftyStoreKit
6. Був розроблений дизайн за допомогою нативних UI інструментів
7. Був розроблений мобільний застосунок сканування на мові програмування Swift який відповідав всім сформованим вимогам
8. Були проведені тестування та виправлені всі недоліки та проблеми
9. Була проведена оптимізація та адаптація застосунку для різних моделей IPhone

Апробація результатів дослідження – робота успішно пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в ІКТ» за темами: «Аналіз технологій сканування штрих-кодів для мобільних додатків на прикладі розробки на мові Swift» та «Мобільні додатки мовою програмування Swift», 24.04.2024, ДУІКТ, Київ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт SwiftUI Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/tutorials/swiftui/creating-and-combining-views (дата звернення 04.04.2024)
2. How 5G Will Affect Digital Marketing, Media and IoT [Електронний ресурс] – Режим доступу до ресурсу: https://www.emarketer.com/content/getting-ready-for-5g. (дата звернення 09.03.2024)
3. Developer Survey 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://survey.stackoverflow.co/2023/#most-popular-technologies-language-learn. (дата звернення 28.03.2024)
4. Swift Standard Library [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/swift/swift-standard-library. (дата звернення 08.04.2024)
5. Sample Apps Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/tutorials/sample-apps/.(дата звернення 18.04.2024)
6. Develop in Swift Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/tutorials/develop-in-swift/.(дата звернення 20.04.2024)
7. Офіційний сайт Swift [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swift/. (дата звернення 05.04.2024)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для сканування та аналізу штрих-кодів товарів з аналізом прихильностей споживачів і оцінкою корисності мовою Swift

Виконав студент 4 курсу
Групи ПД-42
Шевченко Дмитро Сергійович
Керівник роботи
Викладач кафедри ІПЗ Савіцький Вячеслав Андрійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**
  Спрощення процесу аналізу штрих кодів та порівняння цін з різних продуктових магазинів
- **Об’єкт дослідження**
  Процес сканування та аналізу штрих кодів товарів з аналізом прихильностей споживачів і оцінкою користності
- **Предмет дослідження**
  Мобільний застосунок для сканування та аналізу штрих-кодів товарів з аналізом прихильностей споживачів і оцінкою корисності

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести пошук та аналіз застосунків для сканування штрих кодів
2. Розробити архітектуру мобільного застосунку для платформи IOS
3. Визначити функціональні та нефункціональні вимоги для застосунку
4. Дослідити методи розробки програмного продукту за допомогою методів Apple
5. Дослідити інструменти для розробки застосунку для сканування штрих кодів
6. Розробити дизайн мобільного застосунка для сканування штрих кодів
7. Розробити мобільний застосунок для сканування штрих кодів
8. Провести тестування та виправити всі недоліки та проблеми мобільного застосунку для сканування штрих кодів
9. Провести оптимізацію да адаптацію застосунку для різних моделей IPhone

3

АНАЛІЗ АНАЛОГІВ

Характеристики	Foodstr	QRbot	Costless	ShopScan
Сканування штрих коду	+	+	+	+
Відображення цін про товар з різних магазинів	-	-	+	+
Можливість ввести штрих код вручну	+	+	-	+
Відображення інформації про товар	+	-	+	+
Можливість порівння цін	-	-	+	+
Перегляд історії пошуку	+	+	-	+
Можливість відсканувати штрих код з фотографії з галереї	-	+	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Сканувати штрих код
2. Можливість вводити штрих код вручну та знаходити товар
3. Сканувати штрих код товару з фотографії яка знаходиться в галереї
4. Відображення цін з різних магазинів
5. Залишати відгуки до товару
6. Складати список покупок
7. Оглядати попередню історію пошуку товарів
8. Знаходити магазин на карті

Нефункціональні вимоги:

1. Застосунок має бути простим у використанні та зрозумілим
2. Застосунок повинен мати можливість масштабування за потреби
3. Застосунок має бути українською мовою

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

SwiftUI

CarBode

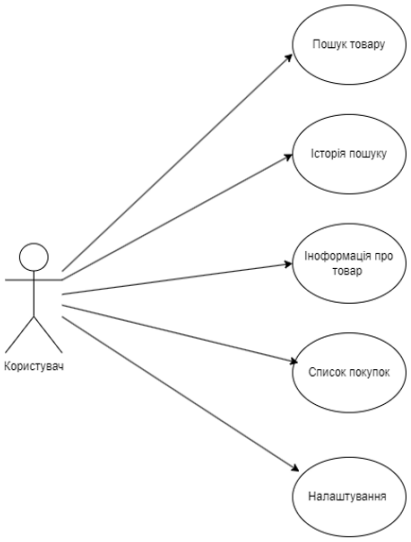
Lottie-ios

SwiftMessages

SwiftyStoreKit

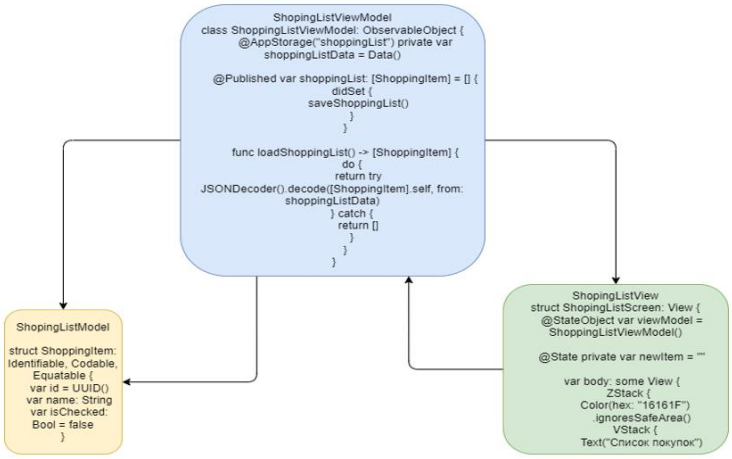
6

Діаграма варіантів використання



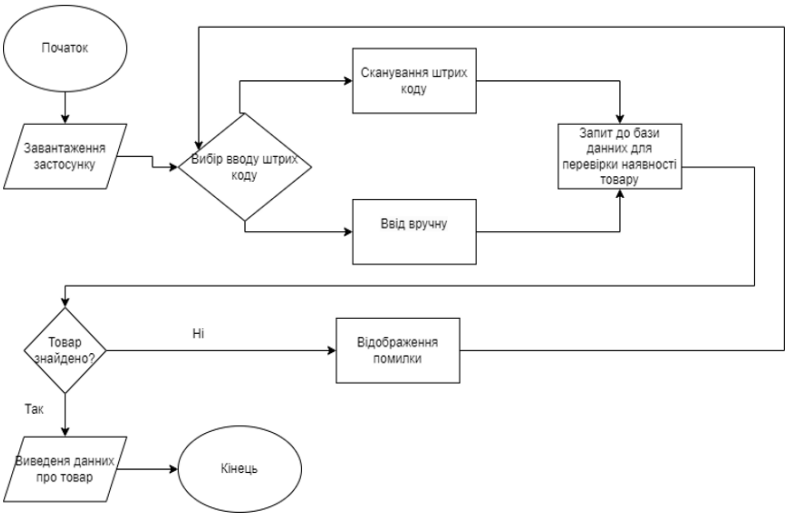
7

Архітектура системи



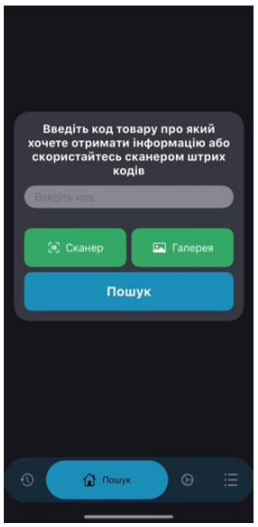
8

Блок схема роботи застосунку

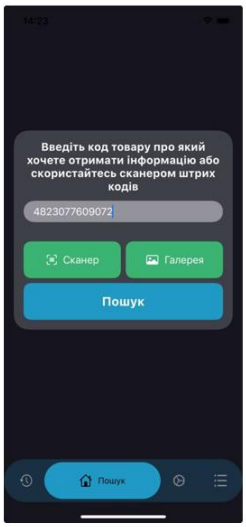


9

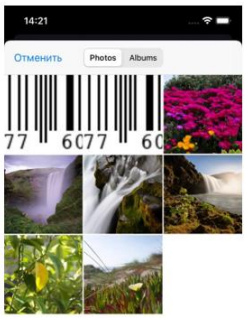
ЕКРАННІ ФОРМИ



Головний екран



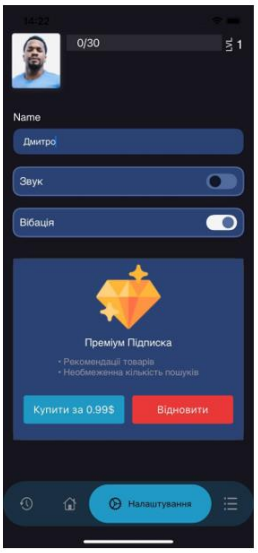
Ввод штрих коду вручну



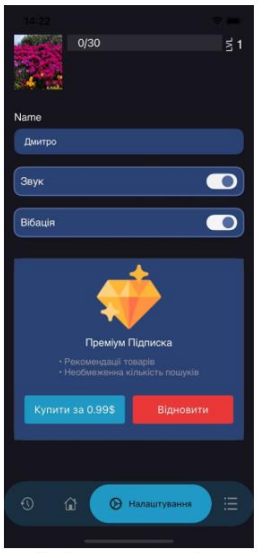
Сканування з готового фото

10

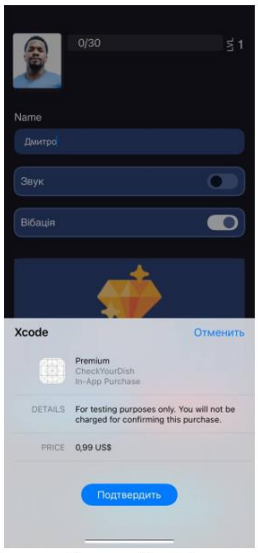
ЕКРАННІ ФОРМИ



Екран налаштування



Зміна параметрів

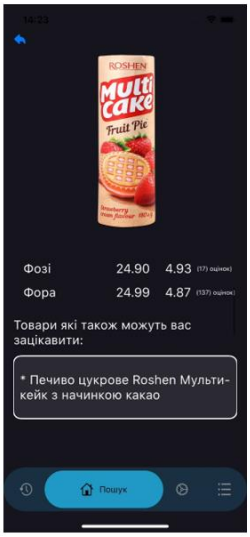


Активація преміум підписки

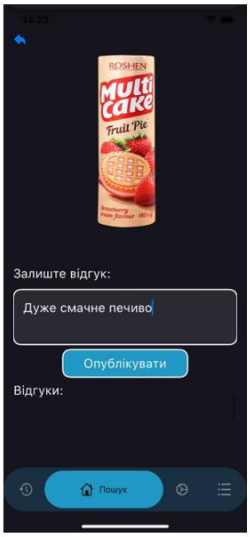
11

ЕКРАННІ ФОРМИ

Екран товару



Рекомендації інших товарів

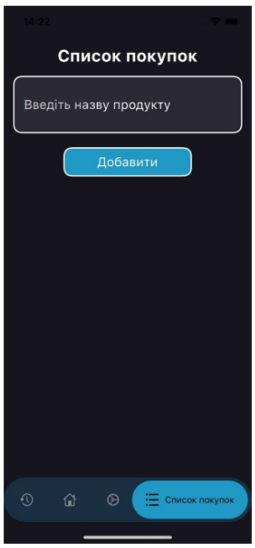


Відгуки

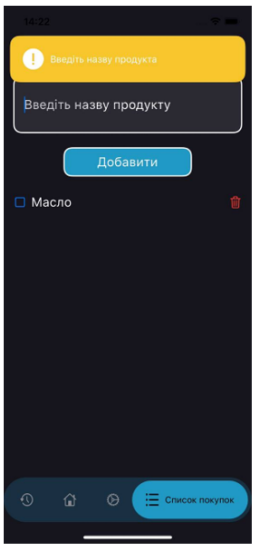
Мапа адрес

12

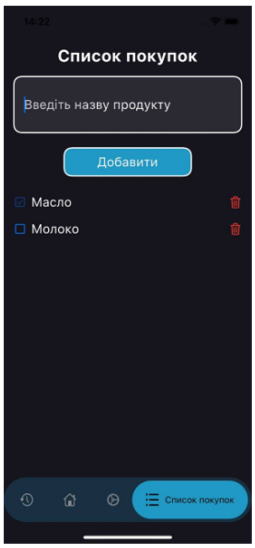
ЕКРАННІ ФОРМИ

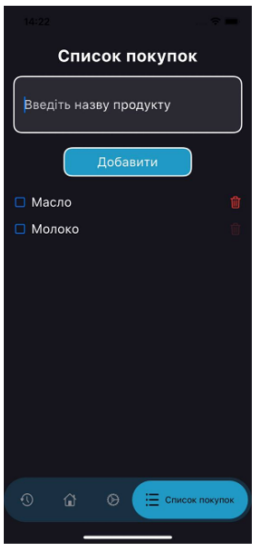


Екран списку продуктів



Підказка користувачу

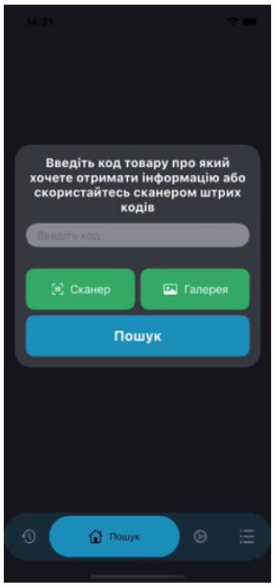




Виділення та видалення вже товару

13

ЕКРАННІ ФОРМИ



14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Шевченко Д.С., Савіцький В.А. Аналіз технологій сканування штрих-кодів для мобільних додатків на прикладі розробки на мові Swift: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ».Збірник тез. 24.04.2024, ДУІКТ, м. Київ, К.: ДУІКТ – с. 82
2. Шевченко Д.С., Савіцький В.А. Мобільні додатки мовою програмування Swift : Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ».Збірник тез. 24.04.2024, ДУІКТ, м. Київ, К.: ДУІКТ – с. 80

ВИСНОВКИ

1. Проведено пошук та аналіз подібних застосунків таких як Foodstr та Costless, QRbot
2. Була розроблена архітектура мобільного застосунку для платформи IOS
3. Були визначені функціональні та не функціональні вимоги до застосунку для сканування штрих коду
4. Були досліджені методи розробки програмного застосунку за допомогою методів Apple
5. Були досліджені інструменти для розробки застосунків а саме SwiftUI, CarBode, Lottie-ios, SwiftMessages, SwiftyStoreKit
6. Був розроблений дизайн за допомогою нативних UI інструментів
7. Був розроблений мобільний застосунок сканування на мові програмування Swift який відповідав всім сформованим вимогам
8. Були проведені тестування та виправлені всі недоліки та проблеми
9. Була проведена оптимізація та адаптація застосунку для різних моделей IPhone

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

HistoryView

```
import SwiftUI

struct HistoryScreen: View {
    @ObservedObject var viewModel =
HistoryViewModel()
    var body: some View {
        NavigationView {
            ZStack {
                Color(hex: "16161F")
                    .ignoresSafeArea()
                VStack {
                    Text("Ваша історія пошуку")
                        .foregroundColor(.white)
                        .font(.custom("SFProText-Bold", size:
24))

                    if viewModel.historyProducts.isEmpty {
                        Spacer()
                        Text("Ви ще не використовували
пошук товарів")
                            .foregroundColor(.white)
                            .font(.custom("SFProText-Bold",
size: 16))
                    } else {
                        ScrollView {

ForEach(viewModel.historyProducts, id: \.self) { product in
                                RoundedRectangle(cornerRadius:
25)
                                    .foregroundColor(Color(hex:
"94939B"))
                                    .frame(height: 100)
                                    .overlay {
                                        NavigationLink(destination:
ResultHistory(selectedProduct: product)) {
                                            HStack {
                                                Spacer()

Image(product.photoName)
                                                    .resizable()
                                                    .scaledToFit()
                                                    .padding(5)
                                                Spacer()
                                                Text(product.name)
.foregroundColor(.white)

.font(.custom("SFProText-Regular", size: 16))
                                                Spacer()
                                            }
                                        }

                                    }

                            }
                        }
                        Spacer()
                    }
                    .padding()
                }
                .navigationBarHidden(true)
                .onAppear {
                    viewModel.historyProducts =
viewModel.loadProducts()
                }
            }
        }
    }

    #Preview {
        HistoryScreen()
    }

    struct ResultHistory: View {
        @State var selectedProduct: Product
        @ObservedObject var viewModel =
SearchViewModel()
        @Environment(\.presentationMode) var
presentationMode
        var body: some View {
            ZStack {
                Color(hex: "16161F")
                    .ignoresSafeArea()
                VStack {
                    Image(selectedProduct.photoName)
                        .resizable()
                        .scaledToFit()
                    VStack {
                        Text(selectedProduct.name)
```

```
                .multilineTextAlignment(.leading)
                .foregroundStyle(.white)
                .font(.custom("SFProText-Bold", size:
24))
            HStack {
              VStack(alignment: .leading) {
                Text("Ціна:")
                  .foregroundStyle(.white)
                  .font(.custom("SFProText-Bold",
size: 22))

ForEach(selectedProduct.prices.sorted(by: { $0.key < $1.key
}), id: \.key) { store, price in
                  HStack {
                    Button {

viewModel.showStoresOnMap(store: store)
                    } label: {
                      Text(store)
                        .foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))
                    }
                    Spacer()
                    Text(String(format: "%.2f",
price))
                      .foregroundStyle(.white)
                      .font(.custom("SFProText-
Regular", size: 20))
                  }
                  .padding(.vertical)
                }
              }
              .padding()
              Spacer()
              VStack(alignment: .leading) {
                Text("Рейтинг:")
                  .foregroundStyle(.white)
                  .font(.custom("SFProText-Bold",
size: 22))

ForEach(selectedProduct.ratings.sorted(by: { $0.key < $1.key
}), id: \.key) { store, rating in
                  HStack {
                    Text(String(format: "%.2f",
rating))
                      .foregroundStyle(.white)
                      .font(.custom("SFProText-
Regular", size: 20))
                    Text("(\(Int.random(in:
0..<200)) оцінок)")
                      .foregroundStyle(.white)
                      .font(.custom("SFProText-
Regular", size: 10))
                  }
                  .padding(.vertical)
                }
              }
            }
            .padding()
          }
        }
        .navigationBarTitleDisplayMode(.inline)
        .navigationBarBackButtonHidden(true)
        .navigationBarItems(leading: Button(action: {
          presentationMode.wrappedValue.dismiss()
        }, label: {
          Text("Назад")
        }))
      }
    }
```

HistoryViewModel

```
import SwiftUI

class HistoryViewModel: ObservableObject {
  @AppStorage("historyProducts") var
historyProductsData: Data?
  @Published var historyProducts: [Product] = []

  func saveProducts(_ products: [Product]) {
    do {
      let encoder = JSONEncoder()
      let data = try encoder.encode(products)
      historyProductsData = data
    } catch {
      print("Помилка при збереженні:
\(error.localizedDescription)")
    }
  }

  func loadProducts() -> [Product] {
```

```
                guard let data = historyProductsData else {
                    return []
                }

                do {
                    let decoder = JSONDecoder()
                    let products = try
decoder.decode([Product].self, from: data)
                    return products
                } catch {
                    print("Помилка при загрузці:
\(error.localizedDescription)")
                    return []
                }
            }
        }
                                MainView
        import SwiftUI
        import CarBode
        import AVFoundation
        import Vision
        import Photos

        struct MainSreen: View {
            @ObservedObject var viewModel =
SearchViewModel()
            @State private var image: UIImage?
            @State private var scannedBarcode: String = ""
            @State private var showingImagePicker = false
            @State private var selectedImage: UIImage?
            var body: some View {
                ZStack {
                    Color(hex: "16161F")
                        .ignoresSafeArea()
                    VStack {
                        RoundedRectangle(cornerRadius: 24)
                            .frame(height: 325)
                            .foregroundColor(Color(hex: "3F4048"))
                            .overlay {
                                VStack {
                                    Text("Введіть код товару про
який хочете отримати інформацію або скористайтесь
сканером штрих кодів")
                                        .font(.custom("SFProText-Bold",
size: 18))
                                        .multilineTextAlignment(.center)
                                        .foregroundColor(.white)

                                    HStack {
                                        RoundedRectangle(cornerRadius:
25)
                                            .frame(height: 35)
                                            .overlay {
                                                TextField("Введіть код",
text: $viewModel.search)
                                                    .padding()
                                                    .foregroundColor(.white)
                                            }
                                            .foregroundColor(Color(hex:
"94939B"))
                                    }
                                    Spacer()
                                    HStack {
                                        Button {
                                            viewModel.isScanning = true
                                        } label: {

RoundedRectangle(cornerRadius: 12)
                                                .foregroundColor(Color(hex:
"3CB371"))
                                                .frame(height: 60)
                                                .overlay {
                                                    HStack {
                                                        Image(systemName:
"barcode.viewfinder")

.foregroundColor(.white)
                                                        Text("Сканер")

.foregroundColor(.white)
                                                    }
                                                }
                                        }

                                        Button {
                                            self.selectedImage = nil

self.showingImagePicker.toggle()

                                        } label: {

RoundedRectangle(cornerRadius: 12)
```

```
                        .foregroundColor(Color(hex:
"3CB371"))
                        .frame(height: 60)
                        .overlay {
                            HStack {
                                Image(systemName:
"photo")
.foregroundColor(.white)
                                Text("Галерея")
.foregroundColor(.white)
                            }
                        }
                    }
                }
                Button {
                    if !viewModel.search.isEmpty {
                        if let selectedProduct =
viewModel.products.first(where: { $0.barcode ==
viewModel.search}) {
                            viewModel.historyProducts
= viewModel.loadProducts()
viewModel.historyProducts.append(selectedProduct)
viewModel.saveProducts(viewModel.historyProducts)
                            viewModel.selectedProduct
= selectedProduct
                        } else {
                        }
                    } else {
                    }
                } label: {
                    RoundedRectangle(cornerRadius:
12)
                        .foregroundColor(Color(hex:
"2099C4"))
                        .frame(height: 60)
                        .overlay {
                            Text("Пошук")
                                .foregroundColor(.white)
.font(.custom("SFProText-Bold", size: 20))
                        }
                }
                    }
                    .padding()
                }
                .padding()
            }
            Result(viewModel: viewModel)
.opacity(viewModel.selectedProduct.barcode.isEmpty ? 0 : 1)
        }
        .sheet(isPresented: $showingImagePicker) {
            ImagePicker2(selectedImage:
self.$selectedImage, scannedBarcode: self.$viewModel.search)
        }
        .sheet(isPresented: $viewModel.isScanning) {
            CBScanner(
                supportBarcode: .constant([.ean8, .ean13,
.qr, .code128]),
                scanInterval: .constant(5.0),
                mockBarCode:
.constant(BarcodeData(value:"22", type:.qr))
            ) {
                print("BarCodeType =",$0.type.rawValue,
"Value =",$0.value)
                viewModel.search = $0.value
                viewModel.isScanning = false
            }
            onDraw: {
                print("Preview View Size =
\($0.cameraPreviewView.bounds)")
                print("Barcode Corners = \($0.corners)")
            }
            }
            .ignoresSafeArea()
            Result(viewModel: viewModel)
        }
    }

    #Preview {
        MainSreen()
    }

    struct ImagePicker2: UIViewControllerRepresentable
{
        @Binding var selectedImage: UIImage?
        @Binding var scannedBarcode: String
```

```
        @Environment(\.presentationMode) var
presentationMode

        func makeUIViewController(context: Context) ->
UIImagePickerController {
            let picker = UIImagePickerController()
            picker.delegate = context.coordinator
            picker.sourceType = .photoLibrary
            return picker
        }

        func updateUIViewController(_ uiViewController:
UIImagePickerController, context: Context) {

        }

        func makeCoordinator() -> Coordinator {
            return Coordinator(parent: self)
        }

        class Coordinator: NSObject,
UINavigationControllerDelegate,
UIImagePickerControllerDelegate {
            let parent: ImagePicker2

            init(parent: ImagePicker2) {
                self.parent = parent
            }

            func imagePickerController(_ picker:
UIImagePickerController, didFinishPickingMediaWithInfo
info: [UIImagePickerController.InfoKey : Any]) {
                if let uiImage = info[.originalImage] as?
UIImage {
                    parent.selectedImage = uiImage
                    parent.scanBarcode(uiImage)
                }

parent.presentationMode.wrappedValue.dismiss()
            }
        }

        func scanBarcode(_ image: UIImage) {
            guard let cgImage = image.cgImage else { return
}
            let barcodeRequest =
VNDetectBarcodesRequest { request, error in
                guard error == nil else {
                    print("Barcode detection error:
\(error!.localizedDescription)")
                    return
                }

                guard let barcodes = request.results as?
[VNBarcodeObservation] else {
                    print("else")
                    self.scannedBarcode = "Штрих-код не
знайдено"
                    return
                }
                print(barcodes)


                for barcode in barcodes {
                    print("for")
                    if let payload =
barcode.payloadStringValue {
                        print("for2")
                        self.scannedBarcode = payload
                        return
                    } else {

                    }
                }

                self.scannedBarcode = "Штрих-код не
знайдено"
            }

            let requestHandler =
VNImageRequestHandler(cgImage: cgImage)
            do {
                try
requestHandler.perform([barcodeRequest])
            } catch {
                print("Error performing barcode detection:
\(error.localizedDescription)")
            }
        }
    }
```

Model

```
import Foundation

struct Product: Encodable, Decodable, Identifiable, Hashable {
    var id: UUID = UUID()
    let barcode: String
    let name: String
    let photoName: String
    let prices: [String: Double]
    let ratings: [String: Double]
    var notes: [String] = []

    init(barcode: String, name: String, photoName: String, prices: [String : Double], ratings: [String : Double], notes: [String]) {
        self.barcode = barcode
        self.name = name
        self.photoName = photoName
        self.prices = prices
        self.ratings = ratings
        self.notes = notes
    }
}
```

ResultView

```
import SwiftUI
import MapKit

struct Result: View {
    @ObservedObject var viewModel: SearchViewModel
    @AppStorage("fullVersion") private var fullVersion: Bool = false
    @AppStorage("userName") private var userName: String = ""

    @State private var randomNumbers: [String: Int] = [:]
    var body: some View {
        NavigationView {
            ZStack {
                Color(hex: "16161F")
                    .ignoresSafeArea()
                VStack {
                    HStack {
                        Button {
                            print("kasfcklasmlkamsclk")
                            withAnimation(.easeInOut(duration: 0.5)) {
                                viewModel.selectedProduct = Product(barcode: "", name: "", photoName: "", prices: [:], ratings: [:], notes: [])
                            }

                        } label: {
                            Image(systemName: "arrowshape.turn.up.backward.fill")
                        }
                        Spacer()
                    }
                    .padding(.horizontal)

                    Image(viewModel.selectedProduct.photoName)
                        .resizable()
                    ScrollView {
                        VStack {

                            Text(viewModel.selectedProduct.name)
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-Bold", size: 24))
                            HStack {
                                VStack(alignment: .leading) {
                                    Text("Ціна:")
                                        .foregroundStyle(.white)
                                        .font(.custom("SFProText-Bold", size: 22))

                                    ForEach(viewModel.selectedProduct.prices.sorted(by: { $0.key < $1.key }), id: \.key) { store, price in
                                        HStack {
                                            Button {

                                                viewModel.showStoresOnMap(store: store)
                                            } label: {
                                                Text(store)

                                                    .foregroundStyle(.white)

                                                    .font(.custom("SFProText-Regular", size: 20))
                                            }
                                            Spacer()
```

```
                                    Text(String(format:
"%.2f", price))

.foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))
                                }
                                .padding(.vertical, 3)
                            }
                        }
                        .padding()
                        Spacer()
                        VStack(alignment: .leading) {
                            Text("Рейтинг:")
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-
Bold", size: 22))

ForEach(viewModel.selectedProduct.ratings.sorted(by: {
$0.key < $1.key }), id: \.key) { store, rating in
                                HStack {
                                    Text(String(format:
"%.2f", rating))

.foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))
                                    if let randomNumber =
randomNumbers[store] {

Text("(\(randomNumber)) оцінок)")

.foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 10))
                                    }
                                }
                                .padding(.vertical, 3)
                                .onAppear {
                                    if randomNumbers[store]
== nil {
                                        randomNumbers[store]
= Int.random(in: 0..<200)
                                    }
                                }
                            }
                        }
                    }
                }
                let firstProductName =
viewModel.selectedProduct.name.components(separatedBy: "
").first
                let productsWithSameFirstWord =
viewModel.products.filter { product in
                    let productNameFirstWord =
product.name.components(separatedBy: " ").first ?? ""
                    return productNameFirstWord ==
firstProductName
                }
                if fullVersion {
                    HStack {
                        Text("Товари які також можуть
вас зацікавити: ")
                            .foregroundStyle(.white)
                            .font(.custom("SFProText-
Regular", size: 20))

.multilineTextAlignment(.leading)
                            Spacer()
                    }
                    VStack(alignment: .leading
,spacing: 8) {

ForEach(productsWithSameFirstWord, id: \.self) { product in
                            NavigationLink(destination:
ResultHistory(selectedProduct: product)) {

RoundedRectangle(cornerRadius: 12)

.foregroundStyle(Color(hex: "2A2A32"))
                                .frame(height: 100)
                                .overlay {

RoundedRectangle(cornerRadius: 12)
                                        .stroke(.white,
lineWidth: 2)
                                }
                                .overlay {
                                    VStack {
                                        Text("*
\(product.name)")
```

```
.foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))

.multilineTextAlignment(.leading)
                                    }
                                }
                                .padding(.bottom)
                            }
                        }
                    }
                    VStack {
                        HStack {
                            Text("Залиште відгук:")
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-
Regular", size: 20))

.multilineTextAlignment(.leading)
                            Spacer()
                        }
                        RoundedRectangle(cornerRadius:
12)
                            .foregroundStyle(Color(hex:
"2A2A32"))
                            .frame(height: 88)
                            .overlay {

RoundedRectangle(cornerRadius: 12)
                                    .stroke(.white, lineWidth: 2)
                            }
                            .overlay {
                                VStack {
                                    TextField("Введіть
коментар", text: $viewModel.comment)
                                        .foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))
                                    Spacer()
                                }
                                .padding()
                            }
                        Button {

viewModel.selectedProduct.notes.append(viewModel.comment
)
                            viewModel.comment = ""
                        } label: {
                            RoundedRectangle(cornerRadius:
12)
                                .foregroundStyle(Color(hex:
"2099C4"))
                                .frame(width: 200, height: 44)
                                .overlay {

RoundedRectangle(cornerRadius: 12)
                                        .stroke(.white, lineWidth:
2)
                                }
                                .overlay {
                                    Text("Опублікувати")
                                        .foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 20))
                                }
                        }


                        HStack {
                            Text("Відгуки:")
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-
Regular", size: 20))

.multilineTextAlignment(.leading)
                            Spacer()
                        }

ForEach(viewModel.selectedProduct.notes, id: \.self) { note in
                            RoundedRectangle(cornerRadius:
12)
                                .foregroundStyle(Color(hex:
"2A2A32"))
                                .frame(height: 80)
                                .overlay {

RoundedRectangle(cornerRadius: 12)
                                        .stroke(.white, lineWidth:
2)
```

```
                                }
                                .overlay {
                                    HStack {
                                        VStack(alignment:
.leading) {

Text(userName.isEmpty ? "Анонім" : userName)

.foregroundStyle(.white)

.font(.custom("SFProText-Bold", size: 20))
                                            Text(note)

.foregroundStyle(.white)

.font(.custom("SFProText-Regular", size: 16))
                                        }
                                        Spacer()
                                    }
                                    .padding()
                                }
                                .padding(.top)
                            }
                        }
                    }
                    .padding()

                }
            }
            .navigationBarHidden(true)
        }
    }
}

#Preview {
    Result(viewModel: SearchViewModel())
}

struct MapView: UIViewRepresentable {
    func makeUIView(context: Context) ->
MKMapView {
        MKMapView(frame: .zero)
    }

    func updateUIView(_ mapView: MKMapView,
context: Context) {
        let coordinate =
CLLocationCoordinate2D(latitude: 50.4501, longitude:
30.5234)
        let span = MKCoordinateSpan(latitudeDelta:
0.1, longitudeDelta: 0.1)
        let region = MKCoordinateRegion(center:
coordinate, span: span)
        mapView.setRegion(region, animated: true)

        let annotation = MKPointAnnotation()
        annotation.coordinate = coordinate
        annotation.title = "Магазин"
        mapView.addAnnotation(annotation)
    }
}
                    SearchViewModel
import SwiftUI
import MapKit

class SearchViewModel: ObservableObject {
    @Published var comment: String = ""
    @Published var products: [Product] = [
        Product(barcode: "5410146415616", name:
"Продукт кисломолочний 1.5% Полуниця Actimel",
photoName: "ActimelStrawberry", prices: ["Сільпо": 22.79,
"Фора": 21.69, "Фозі": 20.7], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
            Product(barcode: "5410146415678", name:
"Продукт кисломолочний 1.5% Лісові ягоди Actimel",
photoName: "ActimelWildberrie", prices: ["Сільпо": 21.99,
"Фора": 21.59, "Фозі": 20.7], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
            Product(barcode: "4820226167105", name:
"Продукт кисломолочний 1,4% мультифрукт Actimel",
photoName: "ActimelMultifruit", prices: ["Сільпо": 23.79,
"Фора": 21.69, "Фозі": 20.7], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
            Product(barcode: "4820226167853", name:
"Йогурт Деліссімо Фантазія шоколадні кульки",
photoName: "DelissimoBalls", prices: ["Сільпо": 33.79,
"Фора": 32.99, "Фозі": 31.5], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
            Product(barcode: "4820226168447", name:
"Йогурт Деліссімо Фантазія зірочки у шок глаз",
photoName: "DelissimoStars", prices: ["Сільпо": 35.69,
```

```
"Фора": 32.99, "Фозі": 30.9], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4820226168553", name:
"Йогурт Деліссімо Фантазія з хрусткими пластівцями",
photoName: "DelissimoFlakes", prices: ["Сільпо": 35.69,
"Фора": 32.99, "Фозі": 31.5], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4000177605004", name:
"Сік Capri-Sun Мультивітамінний", photoName:
"CapriSunMultivitamin", prices: ["Сільпо": 29.99, "Фора":
32.99, "Фозі": 31.5], ratings: ["Сільпо": 4.88, "Фора": 4.87,
"Фозі": 4.93], notes: []),
                Product(barcode: "4000177407608", name:
"Напій соковмісний Safari Fruits Capri-Sun", photoName:
"CapriSunSafari", prices: ["Сільпо": 29.99, "Фора": 30.5,
"Фозі": 31.5], ratings: ["Сільпо": 4.88, "Фора": 4.87, "Фозі":
4.93], notes: []),
                Product(barcode: "4820003681053", name:
"Сік «Наш Сік» мультивітамінний", photoName:
"OurJuiceMultivitamin", prices: ["Сільпо": 39.19, "Фора":
40.99, "Фозі": 43.6], ratings: ["Сільпо": 4.88, "Фора": 4.87,
"Фозі": 4.93], notes: []),
                Product(barcode: "4820016252271", name:
"Сік «Наш Сік» мультифруктовий" , photoName: "multik",
prices: ["Сільпо": 39.19, "Фора": 40.99, "Фозі": 43.6], ratings:
["Сільпо": 4.88, "Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4820192263924", name:
"Напій соковий Гранат Наш Сік", photoName:
"OurJuiceGranate", prices: ["Сільпо": 43.96, "Фора": 38.99,
"Фозі": 46.9], ratings: ["Сільпо": 4.88, "Фора": 4.87, "Фозі":
4.93], notes: []),
                Product(barcode: "4820192262781", name:
"Нектар Яблуко-виноград Наш Сік", photoName:
"OurJuiceAppleV", prices: ["Сільпо": 43.96, "Фора": 42.99,
"Фозі": 40.9], ratings: ["Сільпо": 4.88, "Фора": 4.87, "Фозі":
4.93], notes: []),
                Product(barcode: "4820003680179", name:
"Сік томатний з сіллю Наш сік", photoName:
"OurJuiceTomate", prices: ["Сільпо": 55.14, "Фора": 48.89,
"Фозі": 54.7], ratings: ["Сільпо": 4.88, "Фора": 4.87, "Фозі":
4.93], notes: []),
                Product(barcode: "4823077639449", name:
"Батончик Roshen молочно-шоколадний з мигдалем та
кокосовою начинкою", photoName: "RoshenKokos", prices:
["Сільпо": 15.59, "Фора": 14.39, "Фозі": 14.3], ratings:
["Сільпо": 4.88, "Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4823077613630", name:
"Батончик Roshen молочно-шоколадний з начинкою крем-
брюле", photoName: "RoshenKrem", prices: ["Сільпо": 14.89,
"Фора": 14.39, "Фозі": 14.3], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4823077621154", name:
"Батончик Roshen шоколадний з начинкою", photoName:
"RoshenChocolate", prices: ["Сільпо": 14.79, "Фора": 14.39,
"Фозі": 14.3], ratings: ["Сільпо": 4.88, "Фора": 4.87, "Фозі":
4.93], notes: []),
                Product(barcode: "4823077613630", name:
"Батончик Roshen молочно-шоколадний з начинкою
Chocolate&Peanuts", photoName: "RoshenPenuts", prices:
["Сільпо": 14.89, "Фора": 16.89, "Фозі": 15.1], ratings:
["Сільпо": 4.88, "Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4823077609077", name:
"Печиво цукрове Roshen Мульти-кейк з начинкою какао",
photoName: "RoshenKakao", prices: ["Сільпо": 27.39,
"Фора": 25.49, "Фозі": 24.8], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4823077609084", name:
"Печиво цукрове Roshen Мульти-кейк з начинкою молоко-
крем", photoName: "RoshenMilk", prices: ["Сільпо": 27.39,
"Фора": 25.49, "Фозі": 24.8], ratings: ["Сільпо": 4.88,
"Фора": 4.87, "Фозі": 4.93], notes: []),
                Product(barcode: "4823077609072", name:
"Печиво цукрове Roshen Мульти-кейк з начинкою
полуниця-крем", photoName: "RoshenRaspberry", prices:
["Сільпо": 27.39, "Фора": 24.99, "Фозі": 24.9], ratings:
["Сільпо": 4.88, "Фора": 4.87, "Фозі": 4.93], notes: [])
                ]

        @Published var search: String = ""
        @Published var code: String = ""
        @Published var isScanning = false
        @Published var selectedProduct: Product =
Product(barcode: "", name: "", photoName: "", prices: [:],
ratings: [:], notes: [])

        func showStoresOnMap(store: String) {
            let searchRequest = MKLocalSearch.Request()
            searchRequest.naturalLanguageQuery = store
            let region = MKCoordinateRegion(center:
CLLocationCoordinate2D(latitude: 50.4501, longitude:
30.5234), latitudinalMeters: 10000, longitudinalMeters: 10000)
```

```
            let localSearch = MKLocalSearch(request:
searchRequest)
            localSearch.start { (response, error) in
                guard let response = response else {
                    if let error = error {
                        print("Ошибка при поиске
местоположения: \(error.localizedDescription)")
                    }
                    return
                }

                let mapView = MKMapView(frame:
UIScreen.main.bounds)
                mapView.setRegion(region, animated: true)

                for mapItem in response.mapItems {
                    let annotation = MKPointAnnotation()
                    annotation.coordinate =
mapItem.placemark.coordinate
                    annotation.title = mapItem.name
                    mapView.addAnnotation(annotation)
                }

                let viewController = UIViewController()
                viewController.view.addSubview(mapView)
                viewController.title = "Магазини"
                let navController =
UINavigationController(rootViewController: viewController)


viewController.navigationItem.leftBarButtonItem =
UIBarButtonItem(barButtonSystemItem: .done, target: self,
action: #selector(self.dismissMapAction))


UIApplication.shared.windows.first?.rootViewController?.pres
ent(navController, animated: true, completion: nil)
            }
        }

        @objc private func dismissMapAction() {

UIApplication.shared.windows.first?.rootViewController?.dism
iss(animated: true, completion: nil)
        }
```

```
        @AppStorage("historyProducts") var
historyProductsData: Data?
        @Published var historyProducts: [Product] = []

        func saveProducts(_ products: [Product]) {
            do {
                let encoder = JSONEncoder()
                let data = try encoder.encode(products)
                historyProductsData = data
            } catch {
                print("Помилка при збереженні:
\(error.localizedDescription)")
            }
        }

        func loadProducts() -> [Product] {
            guard let data = historyProductsData else {
                return []
            }

            do {
                let decoder = JSONDecoder()
                let products = try
decoder.decode([Product].self, from: data)
                return products
            } catch {
                print("Помилка при загрузці:
\(error.localizedDescription)")
                return []
            }
        }

    }
```

SettingsView

```
    import SwiftUI
    import SwiftyStoreKit

    struct SettingsScreen: View {
        @State private var isShowingImagePicker = false
        @State private var selectedPhotoData: Data?
        @State private var selectedImage: UIImage?
        @AppStorage("fullVersion") private var
fullVersion: Bool = false
        @Environment(\.presentationMode) var
presentationMode
        @AppStorage("search") private var search: Int = 0
```

```
        @AppStorage("userName") private var userName:
String = ""
        @AppStorage("sound") private var sound: Bool =
true
        @AppStorage("haptic") private var haptic: Bool =
true
        @AppStorage("profileImageData") private var
profileImageData: Data?
        var body: some View {
          ZStack {
            Color(hex: "16161F")
              .ignoresSafeArea()

            VStack {
              HStack {
                Button {
                  isShowingImagePicker = true
                } label: {
                  if let selectedImage = selectedImage {
                    Image(uiImage: selectedImage)
                      .resizable()
                      .frame(width: 75, height: 85)

.contentShape(RoundedRectangle(cornerRadius: 4))
                  } else {
                    Image("avatar")
                      .aspectRatio(contentMode: .fit)
                      .frame(width: 75, height: 85)

.contentShape(RoundedRectangle(cornerRadius: 4))
                  }
                }
                VStack {
                  HStack(spacing: 0) {
                    ProgressView(value:
Double(search)) {
                      Text("\(search)/30")
                    }

.progressViewStyle(BarProgressStyle(height: 25,
labelFontStyle: .body))
                    Text("LVL")
                      .font(.custom("SFProText-
Medium", size: 13))
                      .rotationEffect(.degrees(-90))
                      .foregroundColor(.white)
                    Text("1")
                      .font(.custom("SFProText-
Medium", size: 20))
                      .foregroundColor(.white)
                  }

                  NavigationLink {

                  } label: {
                    RoundedRectangle(cornerRadius: 4)
                      .fill(Color.clear)
                      .frame(height: 45)

                  }

                }
              }
              .padding(.horizontal)
              .frame(height: 85)
              VStack {
                Rectangle()
                  .frame(height: 2)
                  .foregroundColor(Color(hex:
"16161F"))
                VStack(spacing: 20) {
                  VStack(alignment: .leading) {
                    Text("Name")
                      .font(.custom("SFProText-
Medium", size: 16))
                      .foregroundColor(.white)
                    RoundedRectangle(cornerRadius:
12)
                      .frame(height: 40)
                      .foregroundColor(Color(hex:
"2C4375"))
                      .overlay (
                        TextField("Name", text:
$userName)
                          .font(.custom("SFProText-
Medium", size: 14))
                          .foregroundColor(.white)
                          .padding()
                      )
                  }

                  RoundedRectangle(cornerRadius: 12)
```

```
                .frame(height: 44)
                .foregroundStyle(Color(hex:
"2C4375"))
                .overlay {
                    RoundedRectangle(cornerRadius:
12)
                        .stroke(Color(hex: "536B9F"),
lineWidth: 2)
                }
                .overlay {
                    Toggle(isOn: $sound) {
                        HStack {
                            Text("Звук")
                                .foregroundStyle(.white)
                                .font(.custom("SFUIText-
Medium", size: 16))
                            Spacer()
                        }
                    }
                    .padding(10)

.toggleStyle(SymbolToggleStyle())
                    .onChange(of: sound) {
newValue in
                        if newValue {

SoundManager.shared.playBackgroundMusic()
                        } else {

SoundManager.shared.stopBackgroundMusic()
                        }
                    }
                }

            RoundedRectangle(cornerRadius: 12)
                .frame(height: 44)
                .foregroundStyle(Color(hex:
"2C4375"))
                .overlay {
                    RoundedRectangle(cornerRadius:
12)
                        .stroke(Color(hex: "536B9F"),
lineWidth: 2)
                }
                .overlay {
                    Toggle(isOn: $haptic) {
                        HStack {
                            Text("Вібація")
                                .foregroundStyle(.white)
                                .font(.custom("SFUIText-
Medium", size: 16))
                            Spacer()
                        }
                    }
                    .padding(10)

.toggleStyle(SymbolToggleStyle())
                }

            }
            .padding()
            Rectangle()
                .frame(height: 2)
                .foregroundColor(Color(hex:
"16161F"))
            ZStack {
                RoundedRectangle(cornerRadius: 12)
                    .stroke(Color(hex: "3F4048"),
lineWidth: 1)
                    .background(Color(hex: "2C4375"))
                    .frame(height: 282)
                    .overlay (
                        ZStack {

                            VStack(spacing: 12) {
                                Image("premium")
                                Text("Преміум Підписка")
                                    .foregroundColor(.white)

.font(.custom("SFProText-Semibold", size: 16))
                                VStack(alignment: .leading)
{
                                    Text("• Рекомендації
товарів")

.foregroundColor(Color(hex: "94939B"))

.font(.custom("SFProText-Semibold", size: 14))
                                    Text("• Необмеженна
кількість пошуків")

.multilineTextAlignment(.leading)
```

```
.foregroundColor(Color(hex: "94939B"))

.font(.custom("SFProText-Semibold", size: 14))
                                    }
                                    HStack(spacing: 12) {
                                       Button {
                                          purchaseProduct()
                                       } label: {

RoundedRectangle(cornerRadius: 4)
                                             .frame(height: 44)

.foregroundColor(Color(hex: "2099C4"))
                                             .overlay(
                                                Text("Купити за
0.99$")

.font(.custom("SFProText-Regular", size: 16))

.foregroundColor(.white)
                                             )
                                       }

                                       Button {
                                          restorePurchase()
                                       } label: {

RoundedRectangle(cornerRadius: 4)
                                             .frame(height: 44)

.foregroundColor(Color(hex: "E93939"))
                                             .overlay(

Text("Відновити")

.font(.custom("SFProText-Regular", size: 16))

.foregroundColor(.white)
                                             )
                                       }
                                    }
                                    .padding()
                                 }
                              }
                           )
                              }
                              .padding(.horizontal)
                           }
                           Spacer()
                        }
                     }
                     .sheet(isPresented: $isShowingImagePicker) {
                        ImagePicker(selectedImage: $selectedImage,
isPresented: $isShowingImagePicker)
                     }
                     .onDisappear{
                        if let selectedImage = selectedImage {
                           profileImageData =
selectedImage.jpegData(compressionQuality: 0.5)
                        }
                     }
                     .onAppear{
                        if let profileImageData = profileImageData {
                           selectedImage = UIImage(data:
profileImageData)
                        }
                     }
                  }

                  func purchaseProduct() {

SwiftyStoreKit.purchaseProduct("com.myapp.analytics",
quantity: 1, atomically: true) { result in
                     switch result {
                     case .success(let purchase):
                        print("Purchase Success:
\(purchase.productId)")
                        fullVersion = true
                     case .error(let error):
                        switch error.code {
                        case .unknown: print("Unknown error.
Please contact support")
                        case .clientInvalid: print("Not allowed to
make the payment")
                        case .paymentCancelled: break
                        case .paymentInvalid: print("The purchase
identifier was invalid")
                        case .paymentNotAllowed: print("The
device is not allowed to make the payment")
                        case .storeProductNotAvailable: print("The
product is not available in the current storefront")
```

```
                    case .cloudServicePermissionDenied:
print("Access to cloud service information is not allowed")
                    case
.cloudServiceNetworkConnectionFailed: print("Could not
connect to the network")
                    case .cloudServiceRevoked: print("User has
revoked permission to use this cloud service")
                    default: print((error as
NSError).localizedDescription)
                    }
                case .deferred(purchase: _): break

                }
              }
            }

            func restorePurchase() {
              SwiftyStoreKit.restorePurchases(atomically:
true) { results in
                if results.restoreFailedPurchases.count > 0 {
                  print("Restore Failed:
\(results.restoreFailedPurchases)")
                } else if results.restoredPurchases.count > 0 {
                  fullVersion = true
                  print("Restore Success:
\(results.restoredPurchases)")
                } else {
                  print("Nothing to Restore")
                }
              }
            }
          }

          #Preview {
            SettingsScreen()
          }

          struct BarProgressStyle: ProgressViewStyle {

            var height: Double = 20.0
            var labelFontStyle: Font = .body

            func makeBody(configuration: Configuration) ->
some View {
              let progress = configuration.fractionCompleted
?? 0.0

              GeometryReader { geometry in
                ZStack {
                  RoundedRectangle(cornerRadius: 0)
                    .fill(Color(hex: "2A2A32"))
                    .frame(width: geometry.size.width,
height: height)
                  HStack {
                    RoundedRectangle(cornerRadius: 0)
                      .fill(Color(hex: "2099C4"))
                      .frame(width: geometry.size.width *
CGFloat(progress), height: height)
                      .alignmentGuide(.leading) {
dimension in
                        -(geometry.size.width -
dimension.width) / 2
                      }
                    Spacer()
                  }
                  HStack {
                    if let currentValueLabel =
configuration.currentValueLabel {
                      currentValueLabel
                        .font(.headline)
                        .foregroundColor(.white)
                    }

                    configuration.label
                      .padding(.horizontal)
                      .font(labelFontStyle)
                      .foregroundColor(.white)
                    Spacer()
                  }
                }
              }
            }

            struct SymbolToggleStyle: ToggleStyle {
              func makeBody(configuration: Configuration) ->
some View {
                HStack {
                  configuration.label
```

```
                Spacer()

                RoundedRectangle(cornerRadius: 50)
                    .fill(configuration.isOn ? Color(hex:
"F6F6F8") : Color(hex: "536B9F"))
                    .overlay (
                        Circle()
                            .fill(configuration.isOn ? Color(hex:
"536B9F") : Color(hex: "11172D"))
                            .padding(3)
                            .offset(x: configuration.isOn ? 10 : -
10)

                    )
                    .frame(width: 48, height: 24)
                    .onTapGesture {
                        withAnimation(.spring()) {
                            configuration.isOn.toggle()
                        }
                    }
                }
            }
        }
                        ShopingListModel
        import Foundation

        struct ShoppingItem: Identifiable, Codable, Equatable
{
            var id = UUID()
            var name: String
            var isChecked: Bool = false
        }
                        ShopingListView
        import SwiftUI

        struct ShopingListScreen: View {
            @StateObject var viewModel =
ShoppingListViewModel()

            @State private var newItem = ""

            var body: some View {
                ZStack {
                    Color(hex: "16161F")
                        .ignoresSafeArea()
                    VStack {
                        Text("Список покупок")
                            .foregroundStyle(.white)
                            .font(.custom("SFProText-Bold", size:
26))
                        RoundedRectangle(cornerRadius: 12)
                            .foregroundStyle(Color(hex: "2A2A32"))
                            .frame(height: 88)
                            .overlay {
                                RoundedRectangle(cornerRadius: 12)
                                    .stroke(.white, lineWidth: 2)
                            }
                            .overlay {
                                TextField("Введіть продукт", text:
$newItem)
                                    .placeholder(when:
newItem.isEmpty) {
                                        Text("Введіть назву
продукту").foregroundColor(Color.white)
                                    }
                                    .foregroundStyle(.white)
                                    .font(.custom("SFProText-Regular",
size: 20))
                                    .padding()
                            }

                        Button(action: {
                            if newItem.isEmpty {
                //                  TopMessage.show(message:
"Введіть назву продукта")
                            } else {
                                viewModel.addItem(newItem)
                                newItem = ""
                            }
                        }) {
                            RoundedRectangle(cornerRadius: 12)
                                .foregroundStyle(Color(hex:
"2099C4"))
                                .frame(width: 200, height: 44)
                                .overlay {
                                    RoundedRectangle(cornerRadius:
12)
                                        .stroke(.white, lineWidth: 2)
                                }
                                .overlay {
                                    Text("Добавити")
                                        .foregroundStyle(.white)
```

```
                        .font(.custom("SFProText-
Regular", size: 20))
                    }
                }
                .padding()

                ScrollView {
                    ForEach(viewModel.shoppingList) {
item in
                        HStack {
                            Button(action: {
                                viewModel.toggleItem(at:
viewModel.shoppingList.firstIndex(of: item)!)
                            }) {
                                Image(systemName:
item.isChecked ? "checkmark.square" : "square")
                            }

.buttonStyle(BorderlessButtonStyle())

                            Text(item.name)
                                .foregroundStyle(.white)
                                .font(.custom("SFProText-
Regular", size: 20))
                            //
.strikethrough(item.isChecked)

                            Spacer()

                            Button(action: {
                                viewModel.removeItem(at:
viewModel.shoppingList.firstIndex(of: item)!)
                            }) {
                                Image(systemName: "trash")
                                    .foregroundColor(.red)
                            }

.buttonStyle(BorderlessButtonStyle())
                        }
                        .padding(.vertical, 5)
                    }
                }
            }
            .padding()
        }
        .onAppear {
                viewModel.shoppingList =
viewModel.loadShoppingList()
            }
        }
    }

    #Preview {
        ShopingListScreen()
    }

    extension View {
        func placeholder<Content: View>(
            when shouldShow: Bool,
            alignment: Alignment = .leading,
            @ViewBuilder placeholder: () -> Content) ->
some View {

            ZStack(alignment: alignment) {
                placeholder().opacity(shouldShow ? 1 : 0)
                self
            }
        }
    }
                ShopingListViewModel
    import Foundation
    import SwiftUI


    class ShoppingListViewModel: ObservableObject {
            @AppStorage("shoppingList") private var
shoppingListData = Data()

            @Published var shoppingList: [ShoppingItem] =
[] {
                didSet {
                    saveShoppingList()
                }
            }

            func loadShoppingList() -> [ShoppingItem] {
                do {
                    return try
JSONDecoder().decode([ShoppingItem].self, from:
shoppingListData)
                } catch {
                    return []
```

```
            }
        }

        private func saveShoppingList() {
            do {
                shoppingListData = try
JSONEncoder().encode(shoppingList)
            } catch {
                print("Unable to encode shopping list
data.")
            }
        }

        func addItem(_ name: String) {
            let newItem = ShoppingItem(name: name)
            shoppingList.append(newItem)
        }

        func removeItem(at index: Int) {
            shoppingList.remove(at: index)
        }

        func toggleItem(at index: Int) {
            shoppingList[index].isChecked.toggle()
        }
    }
```