Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА IOS-ЗАСТОСУНКУ ДЛЯ ВИЗНАЧЕННЯ СКЛАДУ ПРОДУКТІВ ХАРЧУВАННЯ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-2
Мєшков Д.М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Сакало Є.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Мєшкову Дмитру Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка iOS-застосунку для визначення складу продуктів харчування.

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 29 травня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування Swift, середовище розробки Xcode, фреймворк для роботи графічного інтерфейсу iOS-застосунку UIKit.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз методів розробки iOS-застосунків.
2. Математичні моделі для зберігання та обробки даних.
3. Програмна реалізація iOS-застосунку для визначення складу продуктів харчування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми розробки мобільних iOS-застосунків, постановка задачі, методи розробки мобільних застосунків, архітектурні рішення для реалізації, вибір методу роботи з даними, програмна реалізація.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Аналіз засобів розробки мобільних застосунків	21.04.23-30.04.23	
5	Проєктування застосунку	01.05.23-14.05.23	
6	Програмна реалізація	15.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-28.05.23	
8	Перевірка на плагіат	29.05.23	
9	Рецензування	30.05.23	
10	Підготовка презентації та доповіді	31.05.23	
11	Занесення роботи в електронний архів	01.06.23	
12	Попередній захист кваліфікаційної роботи	07.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ _____доц. Сакало Є.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 60 с., 26 рис., 30 джерел.

IOS, МОДЕЛЬ ВИЗНАЧЕННЯ QR-КОДУ, ДЕКОДЕР ЗОБРАЖЕННЯ, СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ, ОБРОБКА ДВОВИМІРНИХ ДАНИХ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ОЦІНКИ НАЛЕЖНОСТІ ДО КЛАСУ.

Об'єктом роботи є реалізація iOS-застосунку для надання інформації щодо складу продуктів харчування.

Метою цієї роботи є створення MVC застосунку з використанням бази даних та детектором QR-коду. В тому числі і забезпечення швидкої та точної роботи екранів застосунку.

Для досягнення цієї мети будуть використані новітні методи для роботи з текстовими та графічними даними. Застосовано декодування QR-кодів з використанням останньої версії бібліотеки AVFoundation.

У результаті виконання реалізовано програмний застосунок для визначення складу продуктів харчування.

IOS, QR CODE DETECTION MODEL, IMAGE DECODER, COMPUTER VISION SYSTEMS, INTELLIGENT ANALYSIS, 2D DATA PROCESSING, OBJECT ORIENTED PROGRAMMING, CLASSROOM ASSESSMENTS.

The object of the work is the implementation of an iOS application for providing information on the composition of food products.

The purpose of this work is to create an MVC application using a database and a QR code detector. Including ensuring fast and accurate operation of application screens.

To achieve this goal, the latest methods for working with text and graphic data will be used. Implemented QR code decoding using the latest version of the AVFoundation library.

As a result, a software application for determining the composition of food products was implemented.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

iOS – операційна система, розроблена Apple для пристроїв iPhone, iPad та iPod Touch

Xcode – інтегроване середовище розробки (IDE) для створення програм під iOS та інші платформи Apple, такі як macOS, watchOS та tvOS

Swift – мова програмування, розроблена Apple для створення програм для платформи iOS. Swift замінив Objective-C як основну мову програмування для iOS-розробки

UI – користувацький інтерфейс

UIKit – фреймворк, який містить набір графічних елементів інтерфейсу користувача (UI), таких як кнопки, текстові поля та мітки, які використовуються для створення програм iOS

Storyboard – засіб візуального проєктування інтерфейсу користувача в Xcode

View Controller – об’єкт, керуючий поданням інтерфейсу користувача і його логікою в застосунку

Delegation – патерн проєктування, що використовується в iOS-розробці, який дозволяє об’єктам передавати дії іншим об’єктам для виконання

CocoaPods – менеджер залежностей, який використовується для керування сторонніми бібліотеками та фреймворками у застосунку iOS

Core Data – фреймворк, що використовується для зберігання даних програми у базі даних на пристрої

Auto Layout – технологія для створення адаптивного інтерфейсу користувача, який автоматично підлаштовується під різні розміри та орієнтації екранів пристроїв iOS

ВСТУП

Одним із значних напрямків у розробці застосунків на сьогоднішній день є мобільна розробка. Вона забезпечує роботу з великим спектром завдань, таких як обмін, аналіз, зберігання та обробка даних.

Мобільна розробка нерозривно пов’язана з комп’ютерними науками, оскільки вона включає розробку та оптимізацію програмного забезпечення та алгоритмів, роботу з базами даних, створення інтерфейсу користувача та інші комп’ютерні технології: роботу зі штучним інтелектом і техніками машинного навчання для розпізнавання образів, обробки природної мови, передбачення поведінки користувачів, рекомендацій та інших інтелектуальних функцій, використання геолокації для визначення розташування користувача та надання йому відповідної інформації та сервісів, а також у сфері біомедичної та охорони здоров’я можуть бути використані для збору та аналізу медичних даних, моніторингу стану здоров’я, діагностики та надання медичної допомоги на основу симптомів.

Мобільні програми генерують і обробляють великі обсяги даних, що робить науки про дані релевантними. Аналіз поведінки користувача, сегментація аудиторії, прогнозування та інші методи аналізу даних можуть бути застосовані в мобільній розробці.

Існують дві найбільш поширені системи, в рамках яких ведеться мобільна розробка: iOS та Android. Кожна із систем має свої особливості та нюанси при розробці та використанні. В цілому, iOS розробка актуальна через широку аудиторію, високу прибутковість, високу якість та безпеку застосунків, зручних інструментів розробки та підтримки з боку Apple. У цій роботі буде використано саме систему iOS. Метою цієї роботи є створення MVC застосунку з використанням бази даних та детектором QR-коду. В тому числі і забезпечення швидкої та точної роботи екранів застосунку. Для цього обрано напрямок розробки утиліти у сфері охорони здоров’я. Як результат

буде спроєктовано, розроблено та протестовано мобільний iOS застосунок, здатний зберігати та відображати дані про склад продуктів харчування, а також сканувати QR-коди для надання додаткової інформації про продукт.

Актуальність роботи полягає в тому, що з кожним роком збільшується кількість мобільних пристроїв, зокрема із системою iOS, через що виникає необхідність у розробці та підтримці безлічі мобільних застосунків, що допомагають у вирішенні великого спектру завдань, суть яких в оптимізації та автоматизації різних процесів.

1 СУЧАСНІ МЕТОДИ РОЗРОБКИ IOS-ЗАСТОСУНКІВ

На сьогоднішній день, iOS розробка є вкрай актуальною сферою IT. Існує безліч технологічних можливостей для розробки iOS-застосунків, як кросплатформенних так і нативних. У зв'язку з розвитком технологій і зростанням інтересу до iOS розробки з'являється більше потреби у створенні різних програм для iPhone з урахуванням розширення спектру можливостей як апаратної частини пристроїв, так і програмної. Існує безліч підходів до мобільної розробки. Це стосується як архітектурної сторони (перерахувати основні патерни), так і семантичної (мови програмування). Найбільш гнучкою та поширеною є нативна розробка на Swift.

iOS-розробка – це процес створення програм для пристроїв на базі операційної системи iOS.

Для того щоб проаналізувати сучасні методи розробки iOS-застосунків необхідно дослідити основні моменти в розвитку технологій компанії Apple:

– у 2008 році Apple представила App Store, який став ключовим фактором у розвитку мобільних застосунків та iOS-розробки. Це дозволило розробникам легко поширювати свої застосунки та отримувати дохід від їхнього продажу; розробити алгоритм детектування країв на базі методу Кенні;

– у 2014 році Apple представила Swift – нову мову програмування, яка замінює Objective-C. Swift був розроблений для спрощення iOS-розробки та швидкого створення високопродуктивних застосунків. Swift став доступним для розробників у 2014 році і став популярнішим, ніж Objective-C;

– у 2015 році Apple представила нову версію iOS – iOS 9, яка включала нові функції та покращення, такі як розширення застосунків, multitasking на iPad, покращену роботу Siri та багато іншого. Це дало розробникам більше можливостей для створення більш сучасних і функціональних застосунків;

– у 2017 році Apple випустила новий фреймворк для розробки програм – ARKit. Він дозволяє розробникам створювати застосунки із доповненою реальністю. ARKit був доданий до iOS 11 і став доступним для розробників у 2017 році;

– у 2019 році Apple представила нову версію iOS – iOS 13, яка включала нові функції та покращення, такі як темний режим, новий інтерфейс камери та фото, покращену роботу Siri та багато іншого. Це дозволило розробникам створювати більш привабливі та функціональні програми;

– у 2020 році Apple представила нові процесори для своїх пристроїв – Apple Silicon, які виконуватимуть програми для iOS на Mac-комп'ютерах.

iOS-розробка продовжує розвиватися, з появою нових технологій та покращенням інструментів для розробки. В даний час iOS-розробники можуть використовувати безліч інструментів і технологій, таких як Xcode, Swift, Objective-C, SwiftUI, ARKit, Core Data та багато інших. Зі зростанням популярності мобільних пристроїв та збільшенням кількості користувачів iOS-платформи, iOS-розробка залишається перспективною областю для розробників.

1.1 Основні поняття при розробці мобільних iOS-застосунків

Основні компоненти iOS-застосунку включають в себе інтерфейс користувача (UI), який створюється за допомогою набору інструментів Apple – UIKit, а також логіку програми, написану на Objective-C або Swift. Для зберігання даних у iOS-застосунках використовуються бази даних, такі як SQLite або Core Data.

Важливим аспектом нативної iOS-розробки є використання API та бібліотек, що надаються Apple, таких як Cocoa Touch, які полегшують

створення програм та забезпечують швидкий доступ до різних функцій пристрою, таких як камера, GPS, контакти, мікрофон та інші.

В iOS-застосунках існує можливість розпізнавання образів, зокрема QR-кодів, що часто використовуються у якості посилань на різноманітний контент в мережі Інтернет. Розпізнавання QR-кодів у мові Swift, як і будь-якого іншого 2D-штрих-коду, засноване на алгоритмах комп'ютерного зору.

Основний алгоритм, який використовується для розпізнавання QR-кодів – це алгоритм пошуку та декодування синхронізаційного шаблону. QR-код містить три синхронізаційних шаблони, які допомагають пристрою, який зчитує QR-код, вирівнювати і орієнтувати зображення, щоб коректно рахувати інформацію з QR-коду.

Алгоритм пошуку та декодування синхронізаційного шаблону ґрунтується на обробці зображення, отриманого з камери пристрою, та пошуку відповідних областей, що відповідають синхронізаційному шаблону. Потім відбувається декодування інформації всередині кожної області, щоб отримати інформацію, що міститься в QR-коді.

Інші алгоритми, що використовуються для розпізнавання QR-кодів, включають:

– алгоритм бінаризації зображення, який використовується для перетворення кольорового зображення на чорно-біле зображення, яке простіше аналізувати;

– алгоритми вирівнювання та орієнтування зображення, які використовуються для коректного розташування QR-коду у кадрі камери;

– алгоритми декодування інформації, що міститься в QR-коді, які використовуються для отримання текстової або бінарної інформації з QR-коду.

В цілому алгоритми розпізнавання QR-кодів засновані на використанні комп'ютерного зору, обробки зображень та аналізу інформації, що міститься в QR-коді. У мові Swift для реалізації цих алгоритмів використовується

бібліотека AVFoundation, яка надає доступ до камери та медіафайлів на пристроях iOS та macOS.

Після того, як програма розроблена, її можна додати в App Store, де користувачі iOS-пристроїв можуть завантажити та встановити її на свої пристрої.

1.2 Аналіз методів мобільної розробки

Нативна iOS-розробка надає високий рівень продуктивності та інтеграції з пристроями iOS, що робить програми більш чуйними та зручними для користувачів. Однак, для створення нативних iOS-застосунків потрібен додатковий час та ресурси, в порівнянні з іншими методами розробки, такими як кроссплатформенна розробка.

Нативна iOS-розробка та кросплатформна iOS-розробка відрізняються в кількох аспектах:

– мова програмування: нативна iOS-розробка використовує Objective-C або Swift, тоді як кроссплатформенна розробка може використовувати різні мови програмування, такі як JavaScript, C# або Dart;

– інструменти та бібліотеки: для нативної iOS-розробки використовуються інструменти та бібліотеки, що надаються Apple, такі як Xcode та Cocoa Touch; у кросплатформовій розробці використовуються інструменти, такі як React Native, Xamarin, Flutter, які дозволяють розробникам створювати програми для кількох платформ з використанням загального коду;

– продуктивність: програми, створені за допомогою нативної iOS-розробки, зазвичай мають більш високу продуктивність, ніж кросплатформні програми, оскільки вони можуть повністю використовувати можливості пристрою та API iOS, однак, сучасні кросплатформові

фреймворки, такі як Flutter і React Native, надають більш високу продуктивність, ніж раніше;

– вартість та час розробки: розробка програм за допомогою нативної iOS-розробки може зайняти більше часу та ресурсів, оскільки необхідно розробляти окремі програми для кожної платформи; кросплатформова розробка може бути більш економічною, тому що можна використовувати загальний код для створення програм для декількох платформ;

– підтримка пристроїв та оновлень: програми, створені за допомогою нативної iOS-розробки, мають більш високу підтримку пристроїв та оновлень, оскільки вони повністю інтегруються з iOS-системою та API; кросплатформні програми можуть мати проблеми з підтримкою пристроїв та оновлень, оскільки різні платформи можуть мати різні вимоги та можливості.

В цілому, нативна iOS-розробка надає більш високу продуктивність та інтеграцію з iOS-пристроями, тоді як кросплатформова розробка може бути більш економічною та швидкою [1-9]. Вибір методу залежить від конкретних вимог проєкту, бюджету та часу, які доступні для його розробки.

1.3 Аналіз архітектурних підходів при розробці застосунку для iOS

Архітектура при проєктуванні мобільного застосунка відіграє важливу роль у створенні програми, яка є надійною, масштабованою, модульною і легко підтримуваною. Вона забезпечує міцну основу для створення високоякісної програми, спрощує її розробку, дозволяє розробляти застосунок у найкоротші терміни та полегшує його супровід надалі.

Нижче наведено кілька основних причин, чому архітектура важлива при проєктуванні мобільного застосунку:

– поділ відповідальності: хороша архітектура допомагає розділити функціональність програми на логічні компоненти, які можна легко керувати

та обслуговувати; це дозволяє створювати програми, які простіше масштабувати та модифікувати;

– спрощення тестування: хороша архітектура дозволяє розробляти програми, які легко тестувати; компоненти програми можуть бути протестовані незалежно один від одного, що полегшує виявлення та усунення помилок;

– поліпшення продуктивності: хороша архітектура дозволяє розробляти програми, які працюють швидше та ефективніше; це може бути досягнуто шляхом оптимізації коду, а також за допомогою технологій, таких як кешування даних;

– підвищення безпеки: хороша архітектура дозволяє розробляти програми, які більш безпечні і менш схильні до вразливостей; вона забезпечує кращий захист даних користувачів і запобігає можливості атак хакерів на застосунок;

– полегшення підтримки: хороша архітектура дозволяє розробляти програми, які підтримувати легше. Це досягається шляхом поділу функціональності на окремі компоненти, що полегшує виправлення помилок та оновлення програми надалі.

В iOS-розробці часто використовуються такі патерни проєктування:

– MVC (Model-View-Controller): це один із найпоширеніших патернів проєктування в iOS-розробці; він розділяє застосунок на три основні компоненти: модель, подання та контролер [10] (рис. 1.1); модель відповідає за обробку даних, подання – за відображення інтерфейсу користувача, а контролер – за управління взаємодією між моделлю і поданням;

– Delegate: цей патерн використовується для передачі інформації та управління між об’єктами; він дозволяє об’єкту-делегату отримувати повідомлення про події, що відбуваються в інших об’єктах, та обробляти їх [11] (рис. 1.2);

– Singleton: цей патерн використовується для створення єдиного екземпляра об’єкта у застосунку він дозволяє забезпечити доступ до

глобальних ресурсів, таких як бази даних, налаштування програми тощо [12] (рис. 1.3);

– Facade: цей патерн дозволяє приховати складність взаємодії між кількома об’єктами та надає простіший інтерфейс для роботи з ними; наприклад, у iOS-розробці Facade-патерн може використовуватися для надання більш простого інтерфейсу для роботи з низькорівневими API (рис. 1.4);

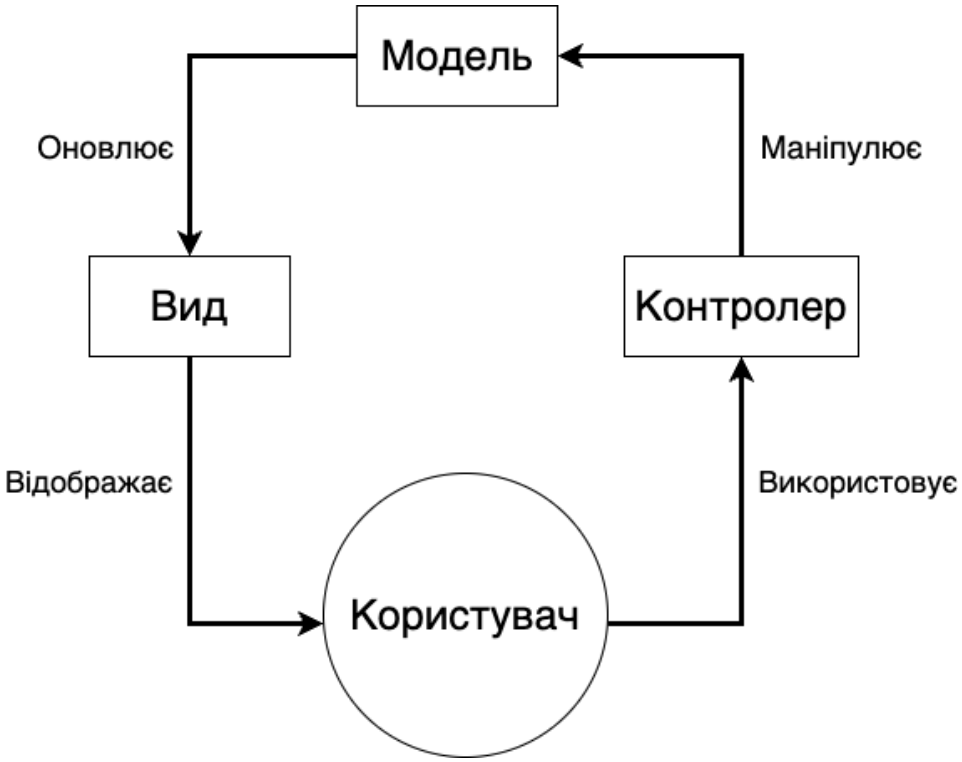

Рисунок 1.1 – Графічне зображення патерну MVC

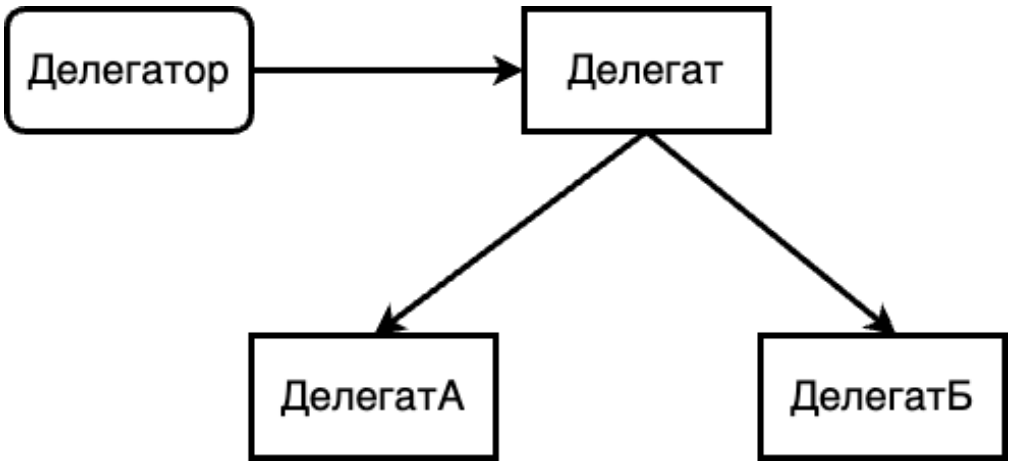

Рисунок 1.2 – Графічне зображення патерну Delegate

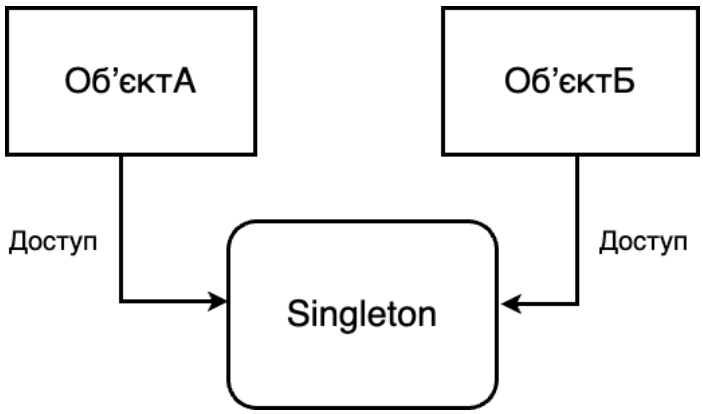

Рисунок 1.3 – Графічне зображення патерну Singleton

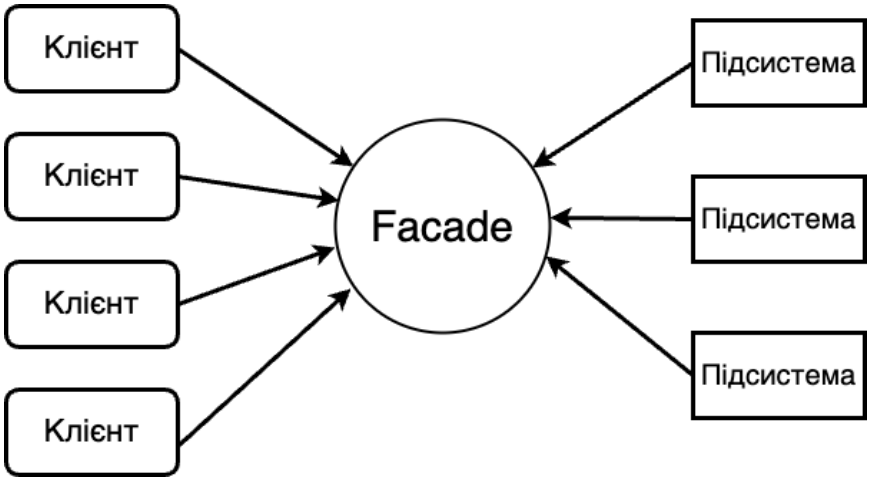

Рисунок 1.4 – Графічне зображення патерну Facade

– Factory: цей патерн використовується для створення об'єктів у програмі; він дозволяє інкапсулювати створення об'єктів та спростити їх створення та використання [13] (рис. 1.5);

– Observer: цей патерн дозволяє об'єктам реєструватися для отримання повідомлень про події, що відбуваються в інших об'єктах [13]; наприклад, в iOS-розробці Observer-патерн може використовуватися для оновлення інтерфейсу користувача при зміні даних у моделі (рис. 1.6);

– Decorator: цей патерн дозволяє динамічно додавати нову функціональність до існуючих об'єктів без зміни їхнього вихідного коду [14]; в iOS-розробці Decorator-патерн може використовуватися для додавання нових функцій до класів UIKit або Core Data (рис. 1.7).

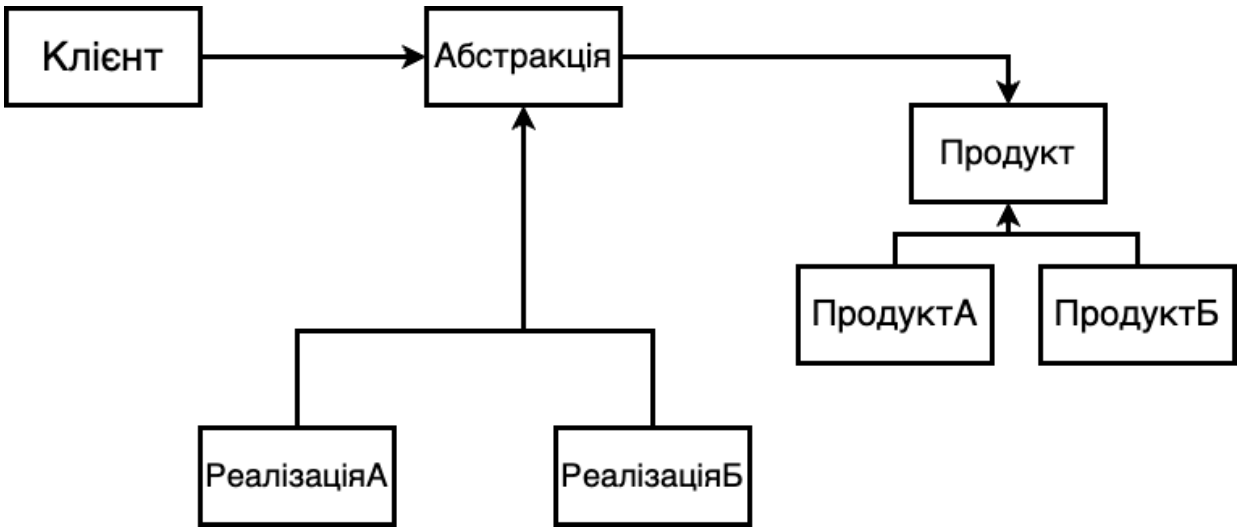

Рисунок 1.5 – Графічне зображення патерну Factory

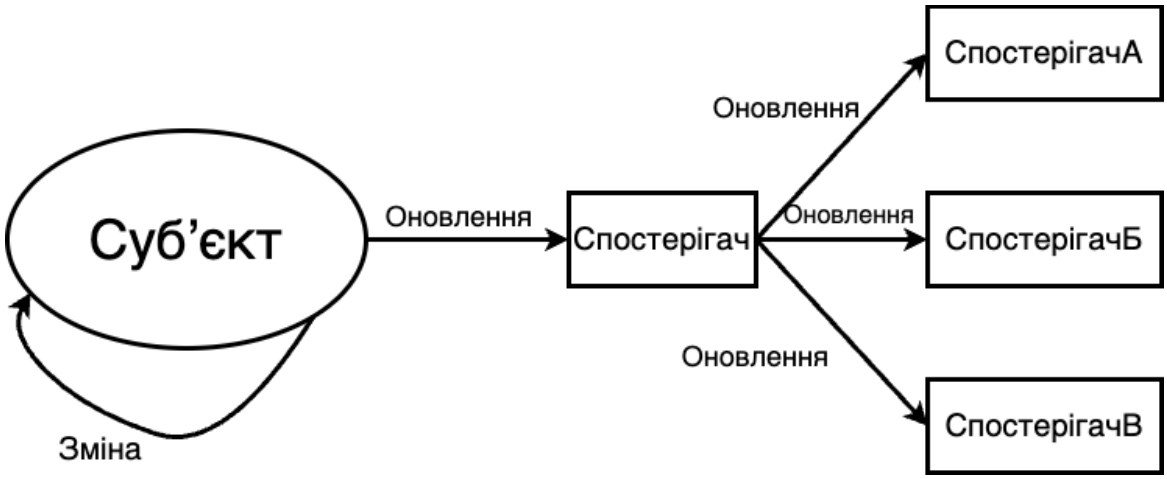

Рисунок 1.6 – Графічне зображення патерну Observer

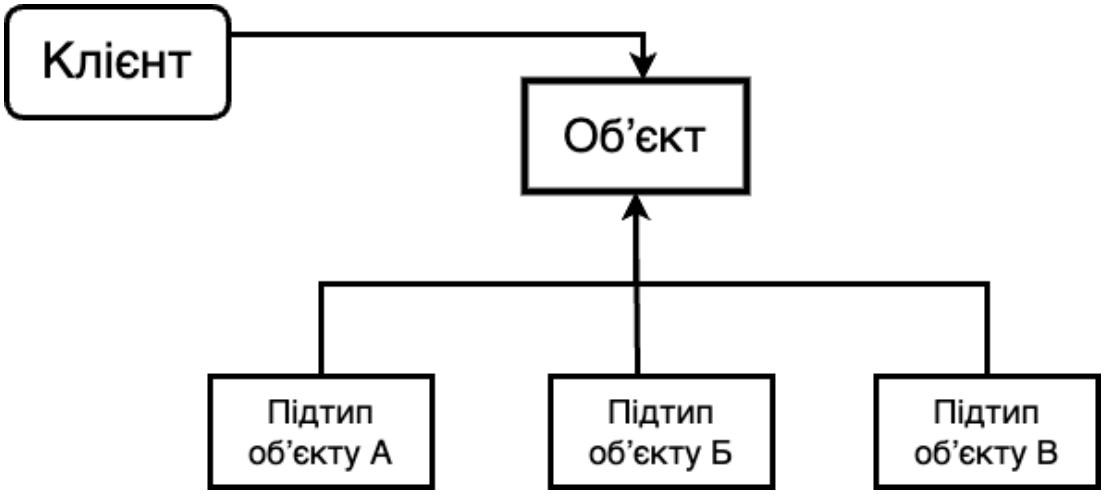

Рисунок 1.7 – графічне зображення патерну Decorator

Це лише деякі з патернів проєктування, які можуть використовуватись у iOS-розробці. Вибір патерну залежить від конкретних вимог проєкту та може бути визначений досвідченим iOS-розробником.

1.4 Постановка задачі

Таким чином, розробка iOS-застосунку з предметною областю «склад продуктів харчування» є актуальним завданням для покращення сфери iOS-застосунків. Тому ставиться завдання розробки нативного iOS-застосунку, що буде надавати інформацію щодо складу продуктів харчування, а також дозволяти сканувати QR-code.

Об’єктом роботи є реалізація iOS-застосунку для надання інформації щодо складу продуктів харчування.

Метою цієї роботи є створення MVC застосунку з використанням бази даних та детектором QR-коду. В тому числі і забезпечення швидкої та точної роботи екранів застосунку.

Для досягнення мети необхідно вирішити такі завдання:

- дослідити методи розробки мобільних застосунків;
- розробити архітектуру мобільного застосунку;
- програмно реалізувати алгоритм пошуку продуктів за назвою;
- розробити дизайн для UI;
- програмно реалізувати функцію зчитування даних з QR-code.

2 МАТЕМАТИЧНІ МОДЕЛІ РОЗРОБКИ IOS-ЗАСТОСУНКУ

Сучасні методи розробки застосунків iOS постійно вдосконалюються. При цьому вирішуються все складніші завдання зберігання, пошуку, обробки, отримання та відображення даних з метою досягнення вищої продуктивності. Програмні інструменти розробника часто ґрунтуються на алгоритмах пов’язаних з оптимізацією. Від них залежить ефективність роботи програми, що включає швидкість виконання операцій, обсяги обчислювальної потужності, необхідної для вирішення завдання, а також пам’яті, що витрачається програмою.

2.1 Моделі для зберігання даних

Математичні моделі, що застосовуються iOS розробником, допомагають створювати ефективні програми, що передбачає стабільність, швидкість роботи, а також ємність споживаної пам’яті. У зв’язку з відносно невеликими обсягами пам’яті, доступною для сучасних смартфонів, є необхідність оптимізації застосунків за допомогою сучасних алгоритмів.

При зберіганні даних у базі даних застосовуються різні алгоритми оптимізації, включаючи:

– індексування: створення індексів для швидкого доступу до даних; популярні алгоритми включають B-дерева, хеш-таблиці та R-дерева;

– оптимізація запитів: вибір найкращого плану виконання запиту за допомогою алгоритмів, таких як оптимізація вартості запиту (cost-based optimization) або використання правил перетворення запитів;

– кешування: використання кешу для зберігання даних, що часто запитуються, що дозволяє знизити навантаження на базу даних і прискорити доступ до даних;

– фрагментація даних: розбиття даних на фрагменти або партиції для розподілу навантаження та підвищення продуктивності;

– компресія даних: стиснення даних для економії простору на диску та збільшення швидкості доступу до даних;

– кластеризація даних: угруповання пов’язаних даних на фізичному рівні для зменшення операцій введення-виведення та покращення продуктивності запитів;

– попереднє обчислення (precomputation): виконання складних операцій або запитів заздалегідь та збереження результатів для повторного використання;

– розпаралелювання: поділ виконання запитів або операцій на кілька паралельних потоків або процесів для прискорення обробки.

Важливо, що алгоритми оптимізації залежить від конкретної бази даних та її системи управління базами даних (СУБД). Різні СУБД можуть використовувати різні алгоритми та стратегії оптимізації.

2.2 Моделі пошуку даних

При використанні великої кількості даних виникає потреба у вибірковому їх відображенні на запит користувача, що призводить до питання про оптимізацію пошукової системи програми.

Для виконання пошукових операцій у базі даних застосовуються різні алгоритми, включаючи:

– лінійний пошук: послідовний вибір всіх елементів бази даних для пошуку збігів; це найбільш простий алгоритм, але може бути неефективним для великих обсягів даних; для списку з n елементів найкращим випадком буде той, при якому потрібне значення дорівнює першому елементу списку і потрібно лише одне порівняння; найгірший випадок буде тоді, коли значення у списку немає (або воно знаходиться в самому кінці списку), у разі чого

необхідно порівняти n; якщо потрібне значення входить до списку k разів і всі входження рівноймовірні, то очікуване число порівнянь

$$\begin{Bmatrix} n, & k = 0 \\ \dfrac{n+1}{k+1}, & 1 \le k \le n \end{Bmatrix}. \tag{2.1}$$

У будь-якому випадку обчислювальна складність алгоритму $O(n)$;

– бінарний пошук: застосовується до впорядкованих даних; він ділить дані навпіл і порівнює потрібне значення з елементом у середині; потім він звужує область пошуку половину і повторює процес до знаходження шуканого значення (рис. 2.1); обчислювальна складність $O(n \log n)$;

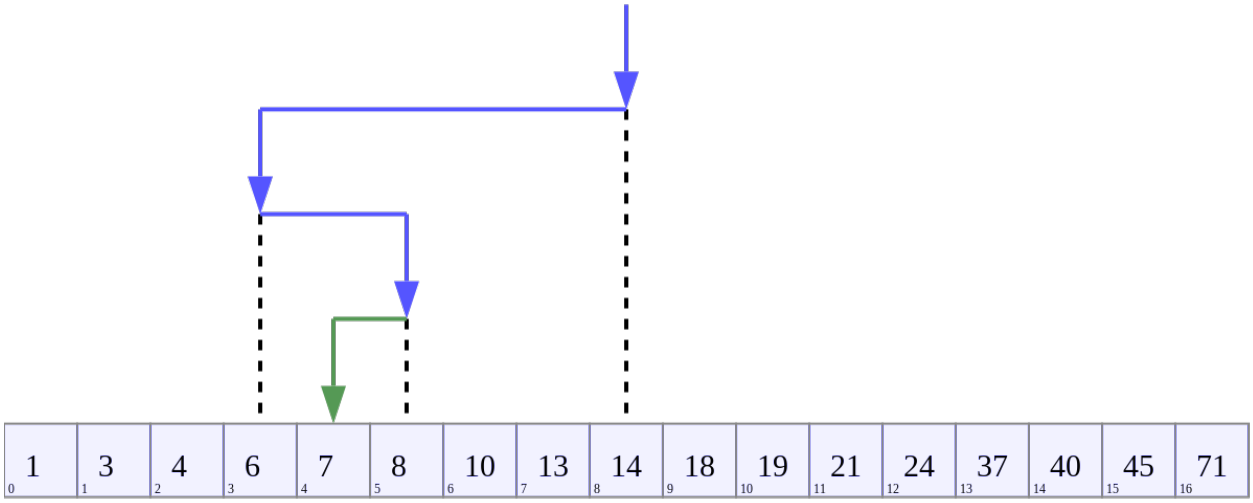

Рисунок 2.1 – Графічне зображення бінарного пошуку

– індексований пошук: використовує структури індексів, такі як B-дерева для швидкого пошуку; індекси дозволяють скоротити кількість даних, які потрібно перевіряти під час пошуку (рис. 2.2);

– хеш-таблиці: використовують хеш-функції швидкого пошуку значень; хеш-таблиця перетворює шукане значення на хеш-код, який використовується для швидкого доступу до відповідного елемента (рис. 2.3);

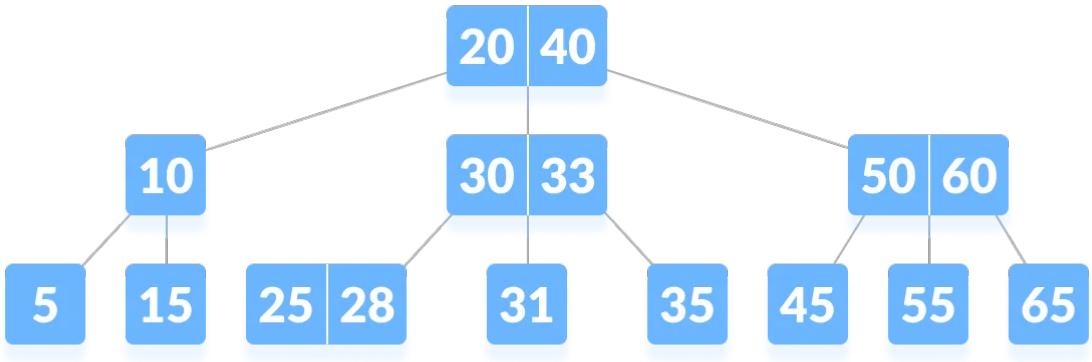

Рисунок 2.2 – Графічне зображення B-дерева

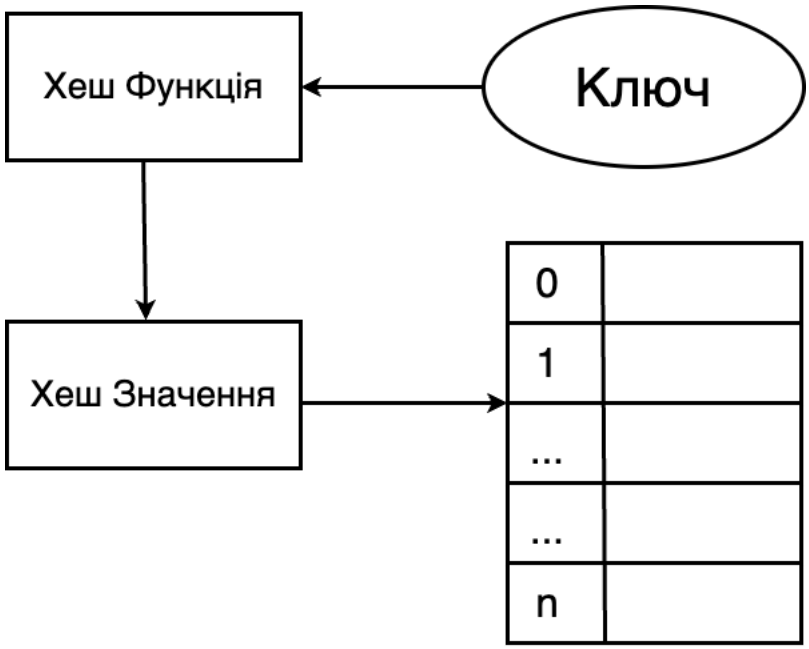

Рисунок 2.3 – Графічне зображення механізму хешування

– повнотекстовий пошук: спеціалізовані алгоритми, такі як алгоритм зворотного індексування (inverted index), використовуються для пошуку за текстовими даними; вони враховують різні параметри, такі як релевантність, повнота та подібність;

– алгоритми оптимізації запитів: системи управління базами даних (СУБД) можуть використовувати різні алгоритми оптимізації запитів для ефективного виконання пошукових операцій; це може включати визначення

найкращого плану виконання запиту, використання індексів чи інших оптимізацій.

Залежно від конкретних вимог та характеристик бази даних може застосовуватися комбінація різних алгоритмів для забезпечення ефективного пошуку.

2.3 Моделі обробки графічних зображень

При роботі з графічними даними на пристроях з iOS системою існує потреба в інструментах для обробки зображень, з метою отримання корисної інформації. Найбільш поширеними завданнями розпізнавання образів на пристроях Apple є розпізнавання облич, тексту та QR-кодів.

iPhone використовує алгоритми розпізнавання облич, засновані на глибокому навчанні та нейронних мережах. Основний алгоритм, який застосовується на iPhone для розпізнавання облич, називається Face ID.

Face ID використовує передові технології, включаючи алгоритми комп’ютерного зору та штучного інтелекту, для точного та безпечного розпізнавання особи користувача. Він працює наступним чином:

– iPhone використовує спеціальний інфрачервоний проєктор, щоб створити точну тривимірну модель особи користувача, проєктуючи тисячі невидимих інфрачервоних точок на обличчя;

– інфрачервона камера на iPhone знімає зображення обличчя, захоплюючи інфрачервоні точки відбиті від особи користувача; це створює детальну інфрачервону карту обличчя;

– отримані дані про особу передаються в глибоку нейронну мережу, яка навчена розпізнавати унікальні особливості особи користувача та порівнювати їх із збереженими шаблонами обличчя; якщо виявиться відповідність між розпізнаною особою та збереженими шаблонами обличчя користувача, iPhone розблокується та надає доступ до функцій та даних.

Face ID також має додаткові функції безпеки, такі як захист від підробки, навчання протягом часу для підвищення точності розпізнавання та адаптація до змін у зовнішності користувача (наприклад, зміна зачіски, носіння головних уборів тощо).

Важливо відзначити, що конкретні деталі алгоритмів Face ID та їх реалізація можуть бути патентовані та бути комерційною таємницею Apple.

Глибока нейронна мережа (Deep Neural Network, DNN) використовується при розпізнаванні особи для аналізу та отримання унікальних особливостей особи, які дозволяють ідентифікувати конкретну людину. Це досягається так:

– спочатку глибока нейронна мережа навчається на великому наборі даних із зображеннями осіб, попередньо розмічених із зазначенням відповідних ідентифікаторів або класів;

– навчена нейронна мережа отримує високорівневі ознаки з вхідного зображення обличчя; у глибоких нейронних мережах зазвичай використовуються згорткові шари, які дозволяють автоматично вивчати ієрархічні особливості зображень;

– вихідні дані мережі, що містять ознаки, подаються у вигляді числового вектора, званого ембедінгом особи; цей вектор є унікальними характеристиками особи, які можуть бути використані для ідентифікації;

– для розпізнавання облич ембеддинг поточного зображення порівнюється зі збереженими ембеддингами, що представляють особи у базі даних; застосовуються різні метрики відстані або подібності, такі як евклідова відстань або косинусна подібність для визначення ступеня відповідності між ембеддингами;

– на основі результату порівняння, наприклад, встановлюється порогове значення для ухвалення рішення про те, чи є розпізнана особа схожою з однією із збережених осіб у базі даних.

Глибокі нейронні мережі, такі як згорткові нейронні мережі (Convolutional Neural Networks, CNN), показують відмінну продуктивність у

завданнях розпізнавання осіб завдяки їх здатності вивчати ієрархічні ознаки та ефективно аналізувати зображення.

iPhone використовує кілька алгоритмів для розпізнавання QR-кодів. Основним алгоритмом є алгоритм комп'ютерного зору, який включає такі кроки:

– алгоритм сканує відеопотік із камери та шукає прямокутні області, які можуть бути потенційними QR-кодами; для цього використовуються різні методи, такі як виявлення країв, кордонів та кутів;

– після детекції алгоритм отримує зображення QR-коду знайденої області і проводить попередню обробку, таку як поліпшення контрастності і видалення шуму, щоб поліпшити розпізнавання (рис. 2.4);

Рисунок 2.4 – Графічне зображення результату роботи механізму зменшення шуму та підвищення контрастності

– алгоритм застосовує спеціальні алгоритми декодування для читання даних, які у QR-коді; це може містити корекцію помилок, розшифровку закодованої інформації та інтерпретацію різних типів даних, таких як текст, URL або інші формати.

Результати розпізнавання можуть бути оброблені та використані у застосунках або операційній системі iPhone відповідно до вимог користувачів або розробників. Наприклад, розпізнаний URL може бути

автоматично відкритий веббраузером або текстова інформація може бути скопійована в буфер обміну.

Ці алгоритми інтегровані у відповідне програмне забезпечення iOS, що надає функцію сканування QR-кодів. Вони оптимізовані для швидкої та точної роботи на пристроях iPhone, забезпечуючи зручність використання та надійність розпізнавання.

Під час детекції QR-кодів використовуються різні алгоритми комп'ютерного зору. Ось деякі з них:

– алгоритми виявлення меж: застосовуються для виділення контурів та меж об'єктів на зображенні; це дозволяє визначити прямокутну форму QR-коду та розділити його від решти зображення;

– алгоритми пошуку кутів: використовуються для виявлення кутових точок QR-коду, які допомагають визначити його орієнтацію та розміри; наприклад, це може бути алгоритм Харріса та Стефена, що є найбільш оптимальним детектором *L*-зв'язкових кутів: для зображення *I* розглянемо вікно *W* (зазвичай розмір вікна дорівнює *5×5* пікселів, але може залежати від розміру зображення) у центрі (*x*, *y*), а також зсув його на (*u*, *v*). Тоді зважена сума квадрата різниць (sum of squared differences (SSD)) між зрушеним і вихідним вікном (тобто зміна околиці точки (*x*, *y*) при зрушенні на (*u*, *v*)) дорівнює:

$$E(u,v) = \sum_{(x,y)\in W} w(x,y)(I(x+u,\, y+v) - I(x,y)^2) \approx$$

$$\approx \sum_{(x,y)\in W} w(x,y)(I_x(x,y)u + I_y(x,y)v)^2 \approx (x,y)M\begin{pmatrix} x \\ y \end{pmatrix}, \qquad (2.2)$$

де $w(x, y)$ – вагова функція (зазвичай використовується функція Гауса чи бінарне вікно);

M – автокореляційна матриця:

$$M = \sum_{(x,y)\in W} w(u,v)\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_v & I_v^2 \end{bmatrix}. \quad (2.3)$$

Кут характеризується великими змінами функції $E(x, y)$ за всіма можливими напрямами (x, y), що еквівалентно великим за модулем власним значенням матриці M. Оскільки безпосередньо рахувати власні значення є трудомістким завданням, Харрісом і Стефеном було запропоновано міру відгуку

$$R = \det M - k(trM)^2 > k, \quad (2.4)$$

де k – емпірична константа, $k \in [0,04; 0,06]$.

Таким чином, значення R є позитивним для кутових особливих точок. Потім проводиться відсікання точок по знайденому порогу R (тобто ті точки, у яких значення R менше деякого порога, виключаються з розгляду). Далі знаходяться локальні максимуми функції відгуку (non-maximal suppression) по околиці заданого радіусу і вибираються як кутові спеціальні точки. Детектор Харріса інваріантний до поворотів, частково інваріантний до афінних змін інтенсивності. До недоліків варто віднести чутливість до шуму та залежність детектора від масштабу зображення (для усунення цього недоліку використовують багатомасштабний детектор Харріса (multi-scale Harris detector));

– алгоритми сканування штрих-коду: ці алгоритми сканують область, де очікується наявність QR-коду, і шукають певні структури, властиві QR-кодам, такі як квадратні модулі та кутові маркери. Щоб визначити, що перед сканером знаходиться QR-код, йому потрібно за щось зачепитися. Для цього існують маркери позиціонування [15-20] – три великі однакові квадрати по краях кожного коду. Якщо подивитися на кожен такий квадрат, він містить 3 квадрати: чорний великий по краю, чорний маленький

посередині і білий між ними. Ці квадрати дозволяють розділити маркер позиціонування на 5 блоків. Два з них по краях чорні, два, такої ж товщини, містять білі та чорні елементи і найширший, по центру, теж складається як із білого, так і чорного кольорів. Співвідношення площ цих блоків становить *1:1:3:1:1*. Таке унікальне співвідношення дозволяє швидко визначити наявність QR-коду на зображенні та його орієнтацію незалежно від того, як код повернутий щодо сканера (рис. 2.5, 2.6).

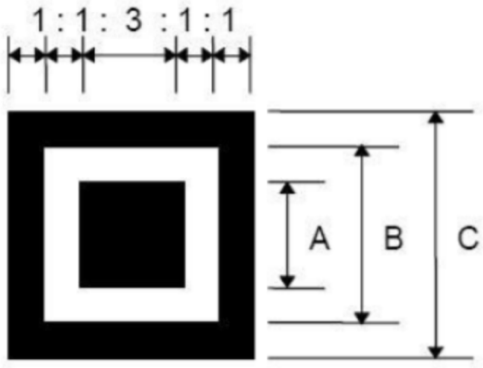

Рисунок 2.5 – Графічне зображення маркеру QR-коду

Рисунок 2.6 – Графічне зображення маркерів для розпізнавання QR-коду

За допомогою цих трьох маркерів код може бути визначений в ідеальних умовах, наприклад, при зчитуванні його з екрана. Ще існує маркер вирівнювання, який зображений на (рис. 2.7). Він допомагає структурувати інформацію.

Необхідним є маркер версій (рис. 2.8). Він визначає, до якої з більш ніж 40 версій належить QR-код [16-19, 20]. Кожна версія має особливості конфігурації та кількості точок (модулів) складових QR-код. Версія 1 містить *21×21* модулів, версія 40 – *177×177*. Зі збільшенням версії змінюється лише кількість інформації, яку можна закодувати в QR-коді. Смартфони зазвичай здатні зчитувати версії з першої по четверту, далі крапки стають для них надто дрібними. інформація про формат.

Рисунок 2.7 – Графічне зображення маркеру вирівнювання QR-коду

Рисунок 2.8 – Графічне зображення маркеру версій QR-коду

Системна інформація міститься у цих частинах QR-коду (рис. 2.9), що підвищує стійкість коду до ушкодженнями, також тут міститься інформація у тому, скільки інформації саме цей код може втратити, доки вона стане критичної для зчитування.

Межі (рис. 2.10) QR-коду потрібні, аби відрізняти його від оточуючого середовища та не обробляти зайву інформацію[15, 21];

Рисунок 2.9 – Графічне зображення частини QR-коду що містить додаткову системну інформацію

Рисунок 2.10 – Графічне зображення частини QR-коду що містить його межі

– алгоритми фільтрації та попередньої обробки: застосовуються для видалення шуму, покращення контрастності та яскравості зображення, що допомагає покращити процес детекції та розпізнавання. Вейвлет-перетворення можуть використовуватися в попередній обробці зображення для покращення якості та полегшення декодування QR-коду. Застосування вейвлет-перетворень може допомогти покращити контрастність зображення QR-коду, що робить його більш чітким та легко читаним [22]. Також вейвлети можуть використовуватись для фільтрації шуму, зменшуючи його

вплив на декодування. Вейвлет-перетворення дають змогу розділити зображення на різні частотні компоненти. Це може бути корисним при вилученні різних деталей QR-коду, таких як кути, модулі або пікселі з високою енергією, що сприяє точному декодуванню. Вейвлет-перетворення дозволяють аналізувати зображення на різних масштабах [23, 24]. Це може бути корисним, якщо розмір QR-коду на зображенні змінюється або коли частина зображення містить масштабований QR-код. Аналіз на різних масштабах допомагає у вилученні даних QR-коду незалежно від його розміру. Однак важливо відзначити, що вейвлет-перетворення не є обов'язковим кроком під час читання QR-кодів, і застосування цих методів залежить від конкретної реалізації та алгоритму читання QR-кодів. Розробники можуть вибирати різні підходи залежно від вимог до якості зображення, ефективності та надійності декодування. Нечіткий пошук не є основним методом для читання QR-кодів, оскільки QR-коди є бінарними матрицями, де інформація кодується у вигляді чорних і білих модулів. Однак, цей алгоритм може використовуватися в попередній обробці зображення QR-коду для покращення якості читання [25-27]. Нечіткий пошук може бути використаний для визначення орієнтації QR-коду на зображенні та його корекції за потреби. Це може допомогти у забезпеченні правильного вирівнювання QR-коду перед його декодуванням. Нечіткий пошук може бути застосований для покращення контрастності та чіткості зображення QR-коду. Він може допомогти пом'якшити шум, збільшити різкість кордонів та покращити якість модулів, що полегшує їх розпізнавання та декодування. Нечіткий пошук може використовуватися для фільтрації та видалення перешкод, таких як шуми, які можуть негативно позначитися на якості зображення QR-коду та ускладнювати його декодування;

– методи машинного навчання: деякі сучасні підходи використовують методи машинного навчання, такі як класифікація або детектування об'єктів на основі нейронних мереж для виявлення QR-кодів. Ці методи вимагають великого обсягу розмічених даних на навчання моделі [28, 29].

Конкретні алгоритми та методи, що застосовуються для детекції QR-кодів, можуть змінюватись в залежності від реалізації та використання. Деякі популярні алгоритми включають Canny Edge Detection, Hough Transform, а також використання машинного навчання із застосуванням глибоких нейронних мереж та нейронних згорткових мереж.

При декодуванні QR-кодів використовуються різні алгоритми для отримання інформації із закодованих даних. Ось деякі з них:

– бібліотеки декодування QR-коду: існують спеціалізовані бібліотеки, такі як ZBar, ZXing (Zebra Crossing), libdecodeQR та інші, які надають готові алгоритми для декодування QR-кодів. Ці бібліотеки зазвичай реалізують різні методи декодування та обробки помилок;

– алгоритми корекції помилок: QR-коди використовують вбудовані механізми корекції помилок підвищення надійності і цілісності даних. Алгоритми корекції помилок, такі як BCH (Bose-Chaudhuri-Hocquenghem) та Ріда-Соломона, застосовуються для виправлення помилок даних QR-коду;

– вилучення інформації: алгоритми декодування отримують інформацію з різних областей і структур QR-коду. Це включає читання маркерів кутів, визначення версії та розміру коду, читання даних із модулів QR-коду та інтерпретацію специфічних форматів даних, таких як текст, URL або інші типи інформації;

– розпізнавання типів даних: QR-коди можуть містити різні типи даних, такі як текст, URL, контактна інформація, географічні координати і т. д. Алгоритми декодування повинні розпізнавати тип даних, щоб відповідним чином обробити та надати інформацію користувачеві або застосунку;

– методи машинного навчання: деякі сучасні підходи використовують методи машинного навчання, такі як класифікація [30] або рекурентні нейронні мережі, для розпізнавання та інтерпретації вмісту QR-кодів. Це може допомогти у більш точному та надійному декодуванні складних QR-кодів або в умовах низької якості зображення.

3 КОМП'ЮТЕРНА МОДЕЛЬ IOS-ЗАСТОСУНКУ

3.1 Обґрунтування вибору інструментарію програмної реалізації

У рамках кваліфікаційної роботи був розроблений застосунок для надання інформації щодо складу продуктів харчування. Програма була розроблена з урахуванням можливостей сучасних технологій. Для реалізації застосунку було обрано мова програмування Swift та фреймворк розробки UIKit та розширення для відкритої бібліотеки Foundation для цієї платформи під назвою AVFoundation, застосунок було розробленно на платформі операційної системи macOS Ventura 13.3.1 та у IDE XCode 2022.

3.1.1 Обґрунтування вибору мови програмування та платформи

Для реалізації було обрано платформу iOS. Платформа iOS від Apple має ряд переваг при розробці утиліти для надання інформації щодоо складу продуктів. При розробці мобільного застосунка для надання інформації про склад продуктів харчування, iOS має такі переваги:

– якість та безпека: iOS має сувору політику якості та безпеки застосунків у App Store. Це означає, що програма повинна відповідати високим стандартам у плані функціональності, безпеки та продуктивності, щоб бути опублікованим. Це сприяє довірі користувачів та забезпечує захист їх даних;

– єдина екосистема: iOS є єдиною екосистемою, що означає, що програма буде працювати на всіх пристроях Apple, включаючи iPhone, iPad та iPod Touch. Це спрощує досягнення широкої аудиторії і забезпечує узгоджений досвід користувача на всіх пристроях;

– інтеграція із системними функціями: iOS надає різні системні функції, які можуть бути корисними для програми, що надає інформацію про

склад продуктів харчування. Наприклад, доступ до камери для сканування штрих-кодів або QR-кодів продуктів, доступ до GPS для визначення розташування користувача та інтеграція з Apple Health для відстеження харчового раціону;

– широкі можливості розробки: iOS пропонує різні інструменти та фреймворки розробки, такі як UIKit, Core Data, Core Location та інші, які спрощують розробку функціональних і привабливих інтерфейсів користувача, обробку даних і роботу з місцезнаходженням;

– потенційна доходність: Платформа iOS має високу платоспроможну аудиторію, що може бути корисним для мобільного застосунка, пов’язаного з харчуванням та здоровим способом життя. Користувачі iOS часто готові платити за якісну програму або її додаткові функції, що може надати можливості для прибутковості.

Однак при розробці iOS-програми також слід враховувати, що вона буде доступна лише для пристроїв Apple і вимагатиме відповідності вимогам та обмеженням, встановленим Apple для публікації програм у App Store.

3.1.2 Обґрунтування вибору середовища розробки

В якості середовища програмної розробки для платформи було обрано IDE XCode 2022.

Розробка за допомогою Xcode, для платформи iOS, має такі переваги:

– інтеграція з iOS та macOS: Xcode повністю інтегрований з платформами iOS та macOS, забезпечуючи підтримку для розробки програм під ці платформи. Він надає доступ до широкого спектру інструментів, фреймворків та ресурсів, які спрощують розробку, налагодження та розгортання програм (рис. 3.1);

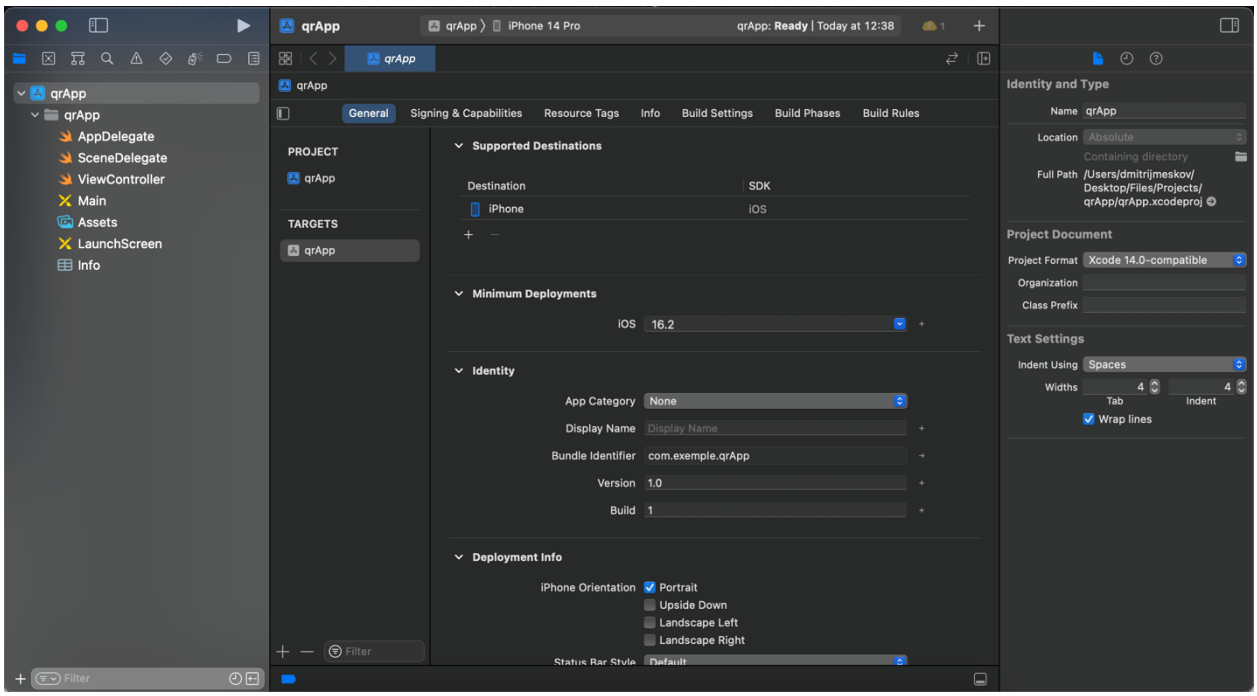

Рисунок 3.1 – Інтерфейс налагодження проєкту

– інтуїтивний інтерфейс: Xcode має інтуїтивний інтерфейс користувача, який полегшує роботу розробника. У ньому є зручні інструменти для створення інтерфейсу користувача, управління файлами проєкту, налаштування збирання та тестування програм;

– потужний налагоджувач: Xcode надає потужний налагоджувач, який допомагає ідентифікувати та виправляти помилки у програмі. Він дозволяє розробникам встановлювати точки зупинки, відстежувати значення змінних, виконувати кроки налагодження та аналізувати стек викликів, що допомагає ефективно знаходити та виправляти проблеми;

– широкий вибір інструментів та фреймворків: Xcode підтримує різні інструменти та фреймворки, які допомагають прискорити розробку програм. Наприклад, Interface Builder дозволяє візуально створювати інтерфейс користувача, а Core Data забезпечує зручну роботу з базою даних;

– інтеграція з іншими сервісами Apple: Xcode інтегрований з іншими сервісами Apple, такими як App Store Connect, TestFlight та Crashlytics. Це дозволяє розробникам керувати процесом розгортання та тестування застосунків, а також отримувати звіти про помилки та збої;

– велика спільнота розробників: Розробка за допомогою Xcode надає доступ до великої спільноти розробників iOS, які можуть надати підтримку, вирішення проблем та рекомендації. Існує безліч онлайн-ресурсів, форумів та спільнот, де можна обговорити та отримати допомогу з розробки з використанням Xcode.

Загалом, Xcode є повнофункціональним та потужним середовищем розробки, спеціально створеним для розробки застосунків під платформи iOS та macOS. Вона має ще ряд переваг, таких як:

– інструменти для автоматичного складання та керування залежностями: Xcode пропонує інтегровану систему складання проєктів, яка автоматично керує залежностями та забезпечує ефективну компіляцію коду. Це допомагає спростити процес складання та покращити продуктивність розробки;

– інструменти для тестування: Xcode надає різні інструменти для тестування програм, включаючи Unit Testing, UI Testing та Performance Testing. Це дозволяє розробникам створювати надійний та стабільний застосунок, а також перевіряти його продуктивність;

– безперервна інтеграція та доставка: Xcode пропонує інтеграцію із системою безперервної інтеграції та доставки (CI/CD) через сервіси, такі як Xcode Server та App Store Connect. Це дозволяє автоматизувати процес складання, тестування та розгортання програми, прискорюючи його доставку на пристрої користувачів;

– підтримка нових технологій та платформ: Xcode регулярно оновлюється, щоб підтримувати останні технології та платформи Apple. Це включає підтримку нових версій iOS, нових пристроїв, таких як iPhone і iPad, а також нові можливості і фреймворки, які можуть бути використані в розробці застосунків;

– легкість інтеграції з іншими інструментами. Xcode інтегрується з іншими популярними інструментами розробки, такими як Git, GitHub та

іншими системами контролю версій. Це полегшує роботу в команді, обмін кодом та управління проєктом;

– безкоштовність: Xcode є безкоштовним середовищем розробки, доступним для скачування з App Store. Це робить його доступним для широкого кола розробників і дозволяє розпочати розробку програм для iOS з мінімальними витратами.

Загалом використання Xcode при розробці iOS-застосунків забезпечує розробникам широкий набір інструментів, зручну інтеграцію з платформою iOS, управління процесом розробки та можливість створення високоякісних та продуктивних застосунків.

3.1.3 Обґрунтування вибору засобів розробки інтерфейсу користувача

Для розробки застосунку було обрано IUKit – фреймворк розробки інтерфейсу користувача (UI) для застосунків під iOS. Він надає набір інструментів, класів і методів, які дозволяють розробникам створювати інтерактивні та естетично привабливі інтерфейси для мобільних застосунків.

Переваги використання UIKit:

– простота використання: UIKit надає простий і інтуїтивно зрозумілий API для створення інтерфейсу користувача. Він пропонує широкий набір готових компонентів, таких як кнопки, текстові поля, таблиці, зображення та інші, які можна легко налаштувати та використовувати у застосунку;

– адаптивність та адаптований для мобільних пристроїв: UIKit розроблений з урахуванням мобільних пристроїв Apple, таких як iPhone та iPad. Він пропонує можливості адаптивного макета та автоматичного розташування елементів інтерфейсу залежно від розміру екрана та орієнтації пристрою;

– візуальна привабливість: UIKit надає потужні інструменти для створення привабливого інтерфейсу користувача. Він підтримує стилі,

анімації, градієнти, тіні та інші ефекти, які можуть поліпшити візуальне враження користувачів;

– інтерактивність та жести: UIKit пропонує підтримку різних жестів та подій, таких як торкання, свайп, жести масштабування та обертання. Це дозволяє створювати інтерактивні та зручні у використанні програми;

– автоматична обробка розташування та розмірів: UIKit пропонує систему автоматичної обробки розташування та розмірів елементів інтерфейсу. За допомогою системи обмежень (constraints) та автопідбору розмірів (autoresizing) можна створювати гнучкий та адаптивний інтерфейс, який добре виглядатиме на різних пристроях та орієнтаціях екрану;

– інтеграція з іншими фреймворками: UIKit інтегрується з іншими фреймворками iOS, такими як Core Animation, Core Graphics, Core Data (рис. 3.2) та іншими. Це дозволяє використовувати їхню функціональність для створення більш складних і потужних застосунків;

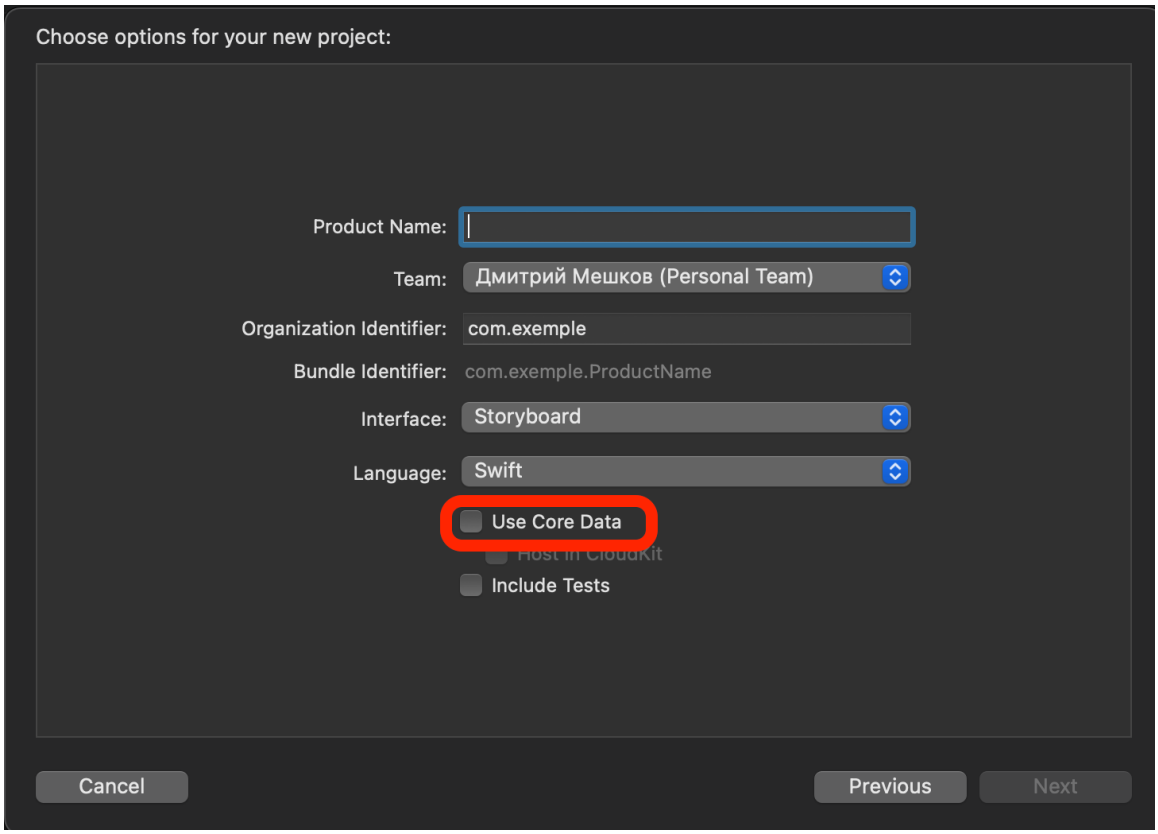

Рисунок 3.2 – Інтерфейс створення проєкту з можливістю підключення CoreData

– оновлення та підтримка: UIKit є основним фреймворком для розробки;

– широка документація та спільнота: UIKit має велику документацію, посібники та приклади, які допомагають розробникам вивчити його функціональність та навчитися використовувати його ефективно. Крім того, існує велика спільнота розробників iOS, які активно обговорюють і діляться досвідом використання UIKit, що полегшує отримання підтримки та допомоги при розробці застосунків;

– підтримка адаптивного дизайну та універсальних програм: UIKit надає засоби для створення адаптивного дизайну, який дозволяє застосуванню виглядати та працювати добре на різних пристроях та орієнтаціях екрану. Крім того, за допомогою UIKit можна створювати універсальні програми, які можуть працювати на різних пристроях iOS, таких як iPhone та iPad;

– інтеграція із системними функціями: UIKit надає доступ до системних функцій та сервісів, таких як камера, геолокація, повідомлення, контакти та інші. Це дозволяє розробникам створювати застосунки, які взаємодіють з різними можливостями пристрою і забезпечують більш багатий досвід користувача.

В цілому, використання UIKit при розробці iOS-застосунків забезпечує розробникам потужний і гнучкий інструментарій для створення якісного інтерфейсу користувача. Він має простоту використання, широкі можливості налаштування та адаптації інтерфейсу під різні пристрої, а також інтеграцію з іншими системними функціями та фреймворками.

У застосунку також було використано AVFoundation – фреймворк, який надається Apple для роботи з мультимедійними даними в iOS та macOS. Він надає різні класи та інструменти, які дозволяють розробникам створювати та обробляти аудіо та відео, працювати з камерою та мікрофоном, а також виконувати інші завдання, пов’язані з мультимедіа.

Ось деякі основні можливості та компоненти AVFoundation:

– захоплення та керування медіаданими: AVFoundation надає можливості захоплення аудіо та відео за допомогою вбудованої камери та мікрофона на пристрої. Він пропонує класи, такі як AVCaptureSession, AVCaptureDevice та AVCaptureInput, які дозволяють налаштовувати та контролювати захоплення медіаданих, включаючи параметри камери та мікрофона;

– відтворення аудіо та відео: AVFoundation дозволяє програвати аудіо та відео файли за допомогою класів AVPlayer та AVPlayerLayer. Він підтримує різні формати медіа, включаючи MP3, AAC, H.264 та інші. Фреймворк також надає можливості контролю відтворення, такі як керування швидкістю, пауза, перемотування та інші;

– обробка аудіо та відео: AVFoundation дозволяє обробляти аудіо та відео дані за допомогою класів AVAsset та AVAssetTrack. Це включає вилучення інформації про медіа, нарізку та склеювання відео та аудіо треків, застосування фільтрів та ефектів, налаштування параметрів та багато іншого;

– генерація та обробка медіафайлів: AVFoundation надає класи для створення та обробки медіафайлів, включаючи класи AVAssetWriter та AVAssetExportSession. Вони дозволяють записувати аудіо та відео дані у файли, експортувати їх у різні формати, застосовувати кодеки, налаштовувати параметри збереження та інші операції;

– розпізнавання та синтез мовлення: AVFoundation надає інструменти для розпізнавання та синтезу мовлення за допомогою класів AVSpeechRecognizer та AVSpeechSynthesizer. Це дозволяє програмам виконувати операції, пов’язані з розпізнаванням голосу та синтезом мови для створення інтерактивних функцій та функцій доступност;

– робота з метаданими: AVFoundation надає можливості роботи з метаданими медіафайлів. За допомогою класів AVAssetMetadata та AVMetadataItem можна отримувати інформацію про різні атрибути медіа, такі як назва, автор, дата створення та інші. Це корисно для організації та відображення додаткової інформації про медіафайли;

– взаємодія із зовнішніми пристроями: AVFoundation дозволяє взаємодіяти із зовнішніми пристроями, такими як аудіо- та відеопристрої через класи AVAudiOSession та AVCaptureDevice. Це дозволяє програмам працювати з підключеними мікрофонами, камерами та іншими пристроями для розширення функціональності та взаємодії із зовнішніми джерелами медіа;

– виявлення облич і розпізнавання об’єктів: AVFoundation надає інструменти для виявлення об’єктів і відео розпізнавання об’єктів. За допомогою класу CIDetector можна виконувати операції виявлення осіб, а також об’єктів, таких як QR-коди та штрих-коди. Це корисно для створення функцій автоматичного фокусування, сценаріїв доповненої реальності та інших інтерактивних можливостей;

– робота з аудіоефектами: AVFoundation пропонує можливості роботи з аудіоефектами, такими як реверберація, еквалайзер та інші. За допомогою класу AVAudioEngine та його компонентів можна створювати складні аудіо-пайплайни, застосовувати ефекти до звукових файлів та реального часу, а також маніпулювати звуком, включаючи зміну тону, гучності та інших параметрів;

– розширена робота з відео: AVFoundation надає розширені можливості роботи з відео, такі як створення власних відеоорендерів з використанням класу AVVideoComposition, обробка та синхронізація відео потоків, застосування відеоефектів за допомогою класу CIFilter та інші операції, що дозволяють створювати більш складні та потужні відеофункції.

Загальний набір функцій та можливостей AVFoundation робить його потужним інструментом для роботи з мультимедійними даними у програмах iOS.

3.2 Програмна реалізація

3.2.1 Вхідні дані та життєвий цикл застосунку

Для вхідних даних було вибрано базу даних United States Department of Agriculture (USDA) та для отримано у вигляді JSON файлу. Через невисоку пропускну здатність API було вирішено зберігати дані локально.

Життєвий цикл iOS-програми може бути описаний наступним чином:

– запуск програми: користувач запускає програму на своєму пристрої, натискаючи на його іконку на головному екрані. Програма завантажується в оперативну пам’ять та запускається;

– завантаження даних з JSON файлу: при запуску програми, зазвичай, відбувається завантаження даних з JSON файлу. Це може бути виконано шляхом надсилання запиту на віддалений сервер або локального ресурсу програми;

– парсинг JSON даних: завантажені дані JSON зазвичай містять структуровану інформацію. Застосунок робить парсинг JSON даних та перетворення їх на об’єкти, які можна використовувати для відображення в таблиці;

– ініціалізація таблиці та відображення даних: після успішного парсингу даних програма ініціалізує таблицю та передає отримані об’єкти даних до таблиці для відображення. Це може бути виконано шляхом налаштування та налаштування джерела даних (наприклад, масиву об’єктів) та делегата таблиці;

– відображення таблиці: програма відображає таблицю даних на екрані пристрою. Кожен елемент даних може бути представлений у вигляді осередку таблиці, що містить інформацію з відповідного об’єкта даних;

– взаємодія з таблицею: користувач може взаємодіяти з таблицею, наприклад прокручувати її, натискати на комірки для перегляду детальної інформації або виконання додаткових дій. Програма реагує на дії

користувача та обробляє їх події, наприклад, відображає деталі вибраної комірки або виконує дії відповідно до вибору користувача;

– оновлення даних: якщо дані в JSON файлі змінюються, програма може виконувати оновлення даних, наприклад, на запит користувача або через певні інтервали часу. При оновленні даних програма повторює процес завантаження даних, парсингу та оновлення таблиці;

– додавання дозволів на використання камери: програма повинна запитати дозвіл користувача на використання камери для сканування QR-кодів. Це може бути виконано при першому запуску або першій спробі відкриття сканера QR-коду;

– ініціалізація сканера QR-коду: застосунок повинен ініціалізувати сканер QR-коду, який використовуватиметься для зчитування QR-кодів. Це може включати налаштування камери та визначення області сканування;

– запуск сканування: користувач може запустити сканування QR-коду, наприклад, натиснувши кнопку або вибравши відповідний пункт меню. Програма активує сканер і починає виявляти QR-коди в області сканування;

– обробка подій сканування: при виявленні QR-коду програма реагує на подію та виконує відповідні дії. Це може включати читання даних з QR-коду, декодування інформації та обробку отриманих результатів;

– відображення результатів сканування: програма відображає результати сканування на екрані пристрою. це може бути виконано шляхом відображення інформації з QR-коду у вигляді тексту, посилання, зображення тощо;

– обробка помилок сканування: якщо виникають помилки під час сканування QR-коду, наприклад, QR-код не може бути прочитаний або містить некоректні дані, програма повинна обробити ці помилки та надати відповідний зворотний зв’язок користувачеві;

– продовження роботи програми: після завершення сканування QR-коду програма може продовжити свою роботу у звичайному режимі,

наприклад, відображати дані з JSON файлу в таблиці або виконувати інші функції програми;

– обробка повторних сканувань: користувач може вирішити повторно відсканувати QR-коди. Програма повинна обробляти такі повторні сканування, оновлювати результати та надавати можливість користувачу повторно сканувати QR-коди;

– вихід із програми: коли користувач закриває програму або перемикається на іншу програму, життєвий цикл програми завершується. Програма зберігає поточний стан та дані, якщо це необхідно, та звільняє ресурси, потім завершує свою роботу.

3.2.2 Реалізація алгоритмів збереження даних

В програмній реалізації була задіяна робота з даними. Спочатку дані отримуються з JSON файлу, в якому вони зберігалися у вигляді ключ – значення. Потім вони містяться в модель, призначену для впорядкованого збереження в масиві. При запиті, данні з моделі постачаються на екран смартфону в інтерфейс таблиці. В такому вигляді користувач вже може взаємодіяти з інформацією з бази даних. При нагоді, є можливість використати текстове поле для рядкового пошуку. Важлива деталь в тому, що в iOS-застосунку дані не зберігаються в таблиці, бо вона – інтерфейс, суть якого у відображенні даних. Завдяки ньому можливо задіяти контекстне меню з додатковою інформацією. У користувача немає права модифікувати дані, додавати, чи видаляти їх. Застосунок не несе в собі мети інтерактивно взаємодіяти з користувачем в CRUD (Create Read Update Delete) моделі. Розроблений застосунок лише є постачальником інформації, вміст якої обирає розробник.

Дані зберігаються локально та не можуть бути видалені із застосунку. Дані не зникають при згортанні чи виходу із мобільної утілити.

3.2.3 Реалізації методів відображення даних на екрані

В iOS розробці часто використовуються протоколи та делегати для оновлення інформації в таблиці на екрані. Ось основні протоколи та делегати, які зазвичай використовуються:

– UITableViewDelegate: цей протокол дозволяє налаштовувати зовнішній вигляд таблиці, реагувати на події вибору комірки, зміни розмірів осередків тощо. Деякі з методів, які можна реалізувати в делегаті UITableViewDelegate, включають: tableView(_:didSelectRowAt:): Викликається при виборі осередку користувачем. Можна обробити цей метод для виконання дій при виборі певного осередку;

– tableView(_:heightForRowAt:): дозволяє налаштувати висоту комірки в таблиці. Можна динамічно налаштовувати висоту осередків залежно від їхнього змісту;

– tableView(_:commit:forRowAt:): викликається під час виконання дії над осередком, таким як видалення або перейменування. Можна обробити ці дії та оновити дані у таблиці;

– UITableViewDataSource: цей протокол несе відповідальність за надання даних для таблиці. Він включає методи, які мають бути реалізовані для надання кількості секцій, числа рядків у кожній секції та змісту осередків. Деякі з методів, які можна реалізувати в делегаті UITableViewDataSource, включають: tableView(_:numberOfRowsInSection:): Повертає кількість рядків у зазначеній секції. Цей метод можна використовувати для визначення кількості рядків у таблиці;

– tableView(_:cellForRowAt:): повертає комірку для зазначеного індексу. Тут можна налаштувати зміст комірки, включаючи текст, зображення та інші елементи інтерфейсу користувача.

Можна реалізувати ці протоколи у класі-делегаті для таблиці. Потім встановіть делегат і джерело даних для об’єкта UITableView, щоб зв’язати

його з класом-делегатом. Це дозволить обробляти події, оновлювати дані і налаштовувати зовнішній вигляд таблиці з урахуванням потреб.

3.2.4 Опис програмної реалізації

Програмна реалізація інтегрувала в себе багато різних патернів та підходів. Для того щоб розроблений застосунок був стабільним, швидким та здатним до зручного масштабування було використано об’єктно-орієнтований та функціональний підходи, патерни Observer, Delegate та MVC.

3.2.4.1 Використання об’єктно-орієнтованого програмування

Swift є мовою програмування, яка підтримує об’єктно-орієнтоване програмування (ООП). ООП – це парадигма програмування, яка дозволяє організовувати програмний код навколо об’єктів, які об’єднують дані та операції над цими даними. Ось основні концепції ООП на Swift:

– класи: класи є основним будівельним блоком об’єктно-орієнтованого програмування в Swift. Класи визначають стан (властивості) та поведінку (методи) об’єктів. Вони можуть успадковувати один від одного і мати поліморфізм;

– властивості (Properties): властивості являють собою дані, які зберігаються всередині об’єктів. У Swift властивості можуть бути змінними (var) або константами (let) і можуть мати різні типи даних;

– методи: методи представляють операції, які можуть бути виконані над об’єктами. Вони поєднують логіку та функціональність, яку об’єкт може виконувати;

– спадкування (Inheritance): спадкування дозволяє створювати нові класи на основі існуючих класів. Підкласи можуть успадковувати властивості та методи від батьківського класу та додавати свої власні властивості та методи;

– поліморфізм (Polymorphism): поліморфізм дозволяє використовувати об’єкти суперкласу замість об’єктів підкласів. Це дозволяє гнучкіше працювати з різними типами об’єктів і спрощує розширення програмного коду;

– інкапсуляція (Encapsulation): інкапсуляція є механізмом приховання внутрішньої реалізації об’єкта і надання доступу тільки до необхідних частин об’єкта. Вона дозволяє контролювати доступ до даних та методів об’єкта;

– абстракція: абстракція дозволяє абстрагуватися від деталей реалізації об’єкта та фокусуватися на його суттєвих аспектах. Це спрощує розуміння та використання об’єктів у програмі.

3.2.4.2 Використання функціонального програмування

Swift підтримує функціональне програмування як одну із парадигм програмування. Ось деякі основні концепції функціонального програмування у Swift:

– функції як типи даних: у функції Swift є повноцінними типами даних. Можна оголошувати функції, передавати їх як параметри іншим функціям, повертати їх з функцій і надавати їх змінним та константам;

– безіменні функції та замикання: Swift підтримує безіменні функції, які називаються замиканнями (closures). Замикання дозволяють визначити та передати логіку виконання як аргументи функцій або зберігати їх у змінних. Вони можуть захоплювати та використовувати значення з навколишнього контексту;

– функції вищого порядку: функції вищого порядку дають змогу використовувати функції як параметри або значення інших функцій, що повертаються. Це дозволяє писати абстрактніший і гнучкий код, який може бути перевикористаний;

– імутабельність (Immutable data): у функціональному програмуванні прийнято працювати з незмінними даними. Swift заохочує використання констант (let) замість змінних (var) завжди, коли це можливо. Це сприяє більш безпечному та передбачуваному коду;

– рекурсія: у функціональному програмуванні рекурсія часто використовується методом вирішення завдань. Swift підтримує рекурсивні функції, які викликають себе для вирішення завдання;

– функції виразу: Swift дозволяє створювати функції у вигляді виразів без необхідності явного оголошення за допомогою ключового слова func. Це робить код компактнішим і лаконічнішим;

– незмінність колекцій: у Swift можна використовувати незмінні колекції (наприклад, масиви та словники) для виконання операцій без зміни вихідних даних. Це сприяє безпечній та передбачуваній роботі з даними.

Функціональне програмування в Swift дозволяє писати більш декларативний, модульний і код, що розширюється. Воно підходить для вирішення різних завдань, включаючи обробку та перетворення даних, паралельні обчислення та багато іншого.

3.2.4.3 Використання патерну MVC

MVC (Model-View-Controller) – це патерн проєктування, що широко використовується при розробці інтерфейсів користувача. Ось було організувано MVC у Swift:

– модель (Model): модель представляє дані та бізнес-логіку програми. Вона не повинна залежати від інтерфейсу користувача і зазвичай реалізується

у вигляді структур або класів. У Swift модель представлена за допомогою структур, класів, перерахувань чи протоколів;

– вид (View): вид відповідає за відображення інтерфейсу користувача і реагує на дії користувача. У Swift це може бути реалізовано за допомогою класів UIViewController, UIView та інших компонентів інтерфейсу користувача. Подання має бути пасивним і не повинне містити бізнес-логіку;

– контролер (Controller): контролер служить сполучною ланкою між моделлю та поданням. Він обробляє дії користувача та оновлює модель або подання на основі цих дій. У Swift контролер реалізується у вигляді класу UIViewController або іншого об’єкта, який відповідає за керування уявленням та взаємодію з моделлю;

– взаємодія: вид передає дії користувача контролеру, наприклад, натискання кнопки. Контролер обробляє ці дії, взаємодіє з моделлю для отримання або оновлення даних та оновлює подання за допомогою методів подання. Оновлення виду може бути зроблено шляхом прямого виклику методів подання або за допомогою патерна спостерігача або протоколів делегатів. Це необхідно для того, аби користувацький інтерфейс був інтерактивним, тобто при натисканні кнопок спрацьовував задуманий алгоритм.

В патерні MVC контролер грає активну роль в управлінні видом і моделлю. Він приймає рішення про те, як дані обробляються і як вони відображаються у поданні. У той же час, подання має бути пасивним і не повинно містити бізнес-логіку або модифікувати модель безпосередньо.

Застосування патерна MVC у Swift допомагає розділити відповідальність між компонентами програми та полегшує підтримку та розширення коду. Він забезпечує кращу організацію коду та покращує його читальність та тестованість.

3.2.4.4 Загальне представлення та опис реалізації

Як представлено на (рис. 3.3) реалізація представлена в вигляді окремих класів які через Dependency Injection впроваджуються в програму як незалежні модулі. Кожен за класів виконує своє призначення та має атомарне призначення для відповідності Single Responsibility Principle (SRP).

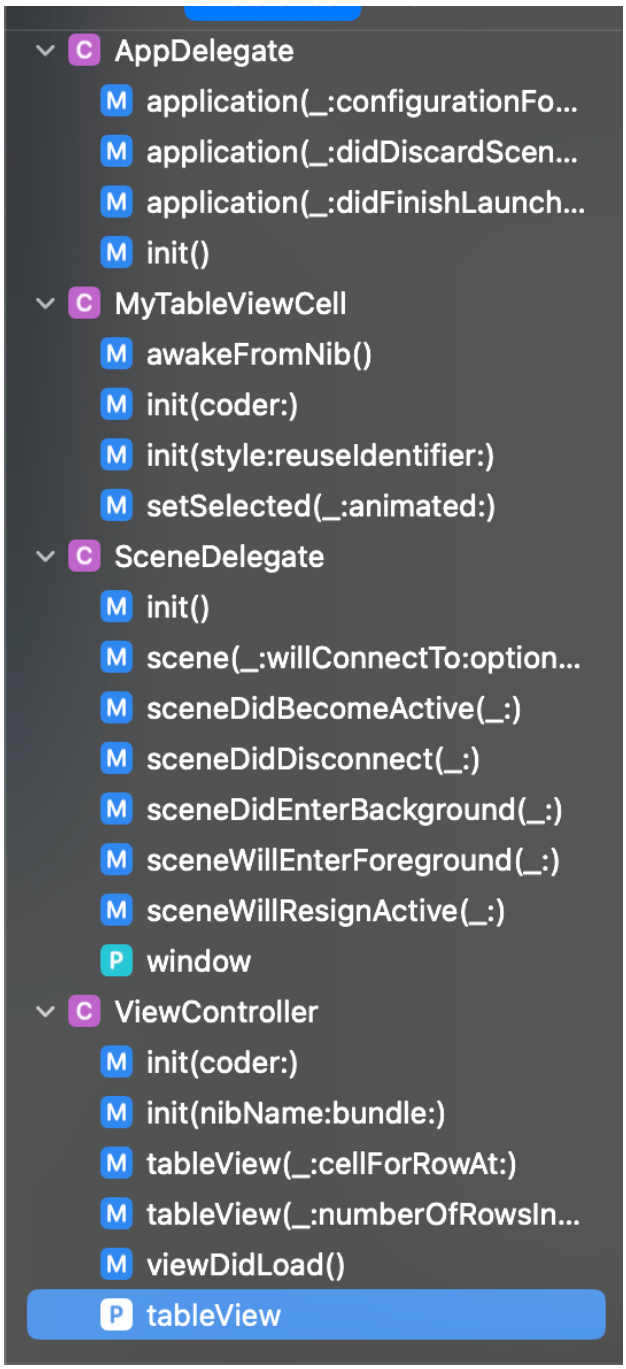

Рисунок 3.3 – Класи програми

З представлених класів видно що кожен клас включає інформацію в вигляді властивостей в яких зберігається інформація про самих себе, наприклад для таблиці зберігається кількість рядків та функціонал взаємодії з ними. Для роботи зі складом продуктів створено структуру даних ProductModel для зручності представлення інформації.

3.3 Інструкція користувача

При запуску застосунку перед користувачем з’явиться головний екран (рис. 3.4).

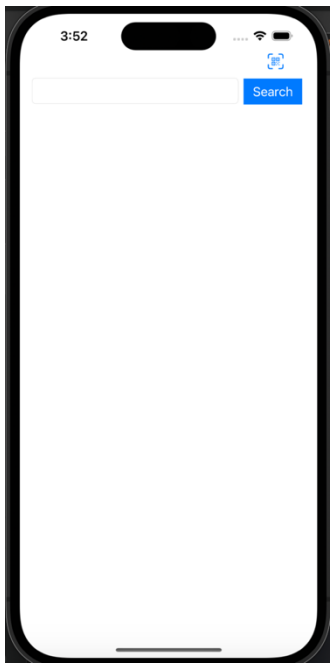

Рисунок 3.4 – Головний екран застосунку

На ньому можна побачити основні елементи користувацького інтерфейсу, а саме: поле пошуку, кнопка пошуку та кнопка для перехода до режиму сканування QR-коду.

Для того, щоб знайти продукт, аби дізнатися його склад , потрібно натиснути на поле пошуку, після чого з’явиться віртуальна клавіатура. Введемо назву будь-якого продукту (рис. 3.5) виключно англійською мовою.

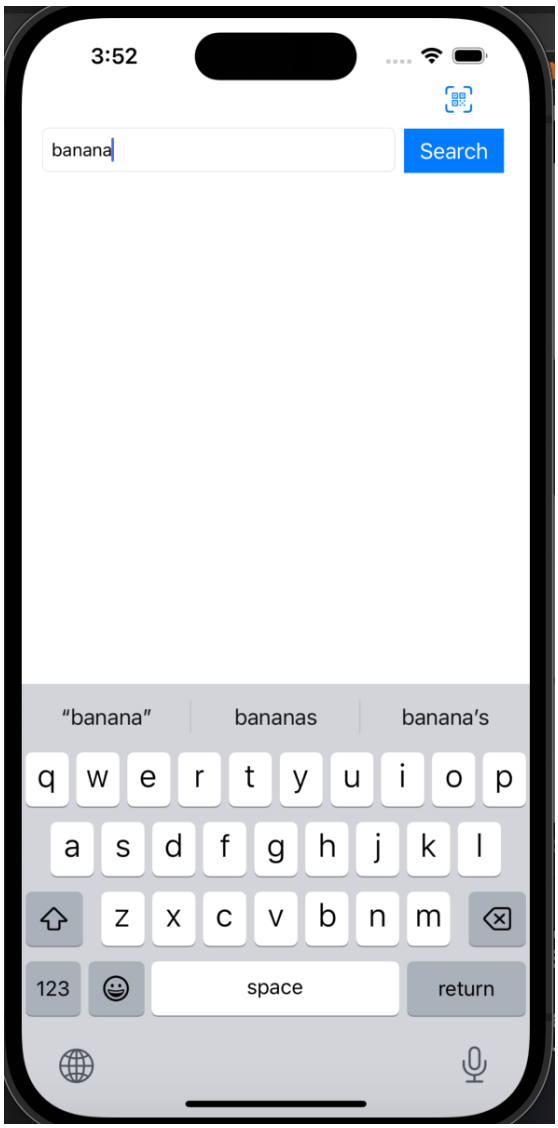

Рисунок 3.5 – Робота з пошуковим рядком

Далі потрібно натиснути кнопку «Search». Перед користувачем з’явиться шуканий продукт (рис. 3.6).

При натисканні на рядок з назвою продукту користувач отримає додаткову інформацію у спливаючому вікні (рис. 3.7).

При натисканні на кнопку «OK» користувач повернеться на головний екран.

При натисканні на кнопку із зображенням QR-коду застосунок перейде у режим камери, у якому почне шукати об’єкти, схожі на QR-код (рис. 3.8).

При знаходженні шуканого зображення застосунок відсканує QR-код та дасть декодований контент (рис. 3.9).

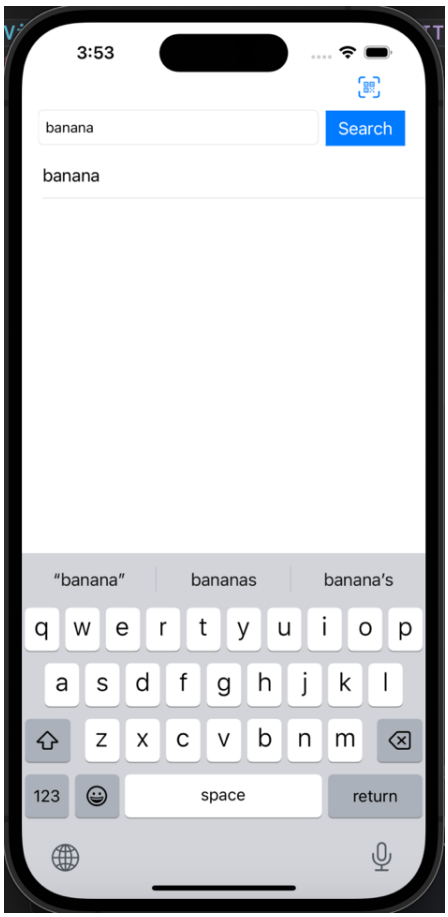

Рисунок 3.6 – Результат пошуку

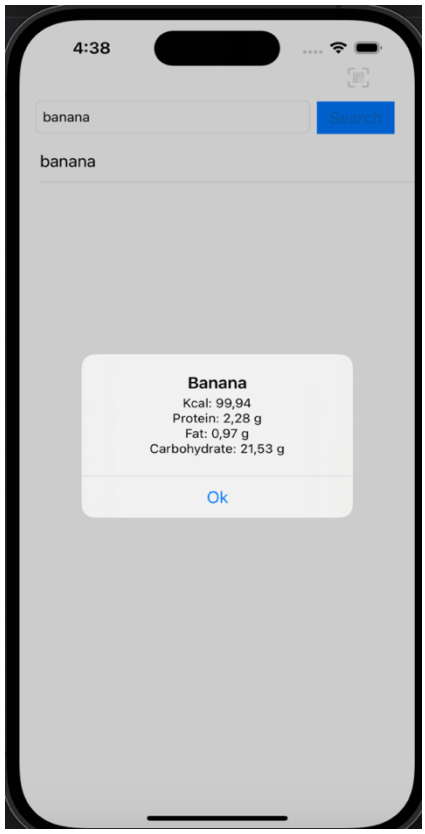

Рисунок 3.7 – Результат натискання на рядок з назвою продукта

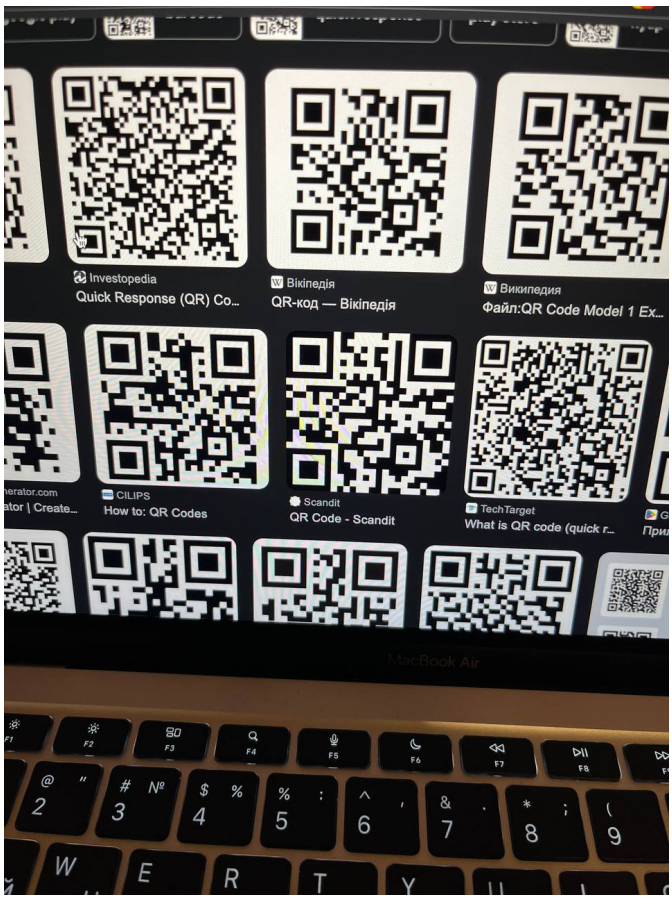

Рисунок 3.8 – Перехід до режиму сканування QR-кодів

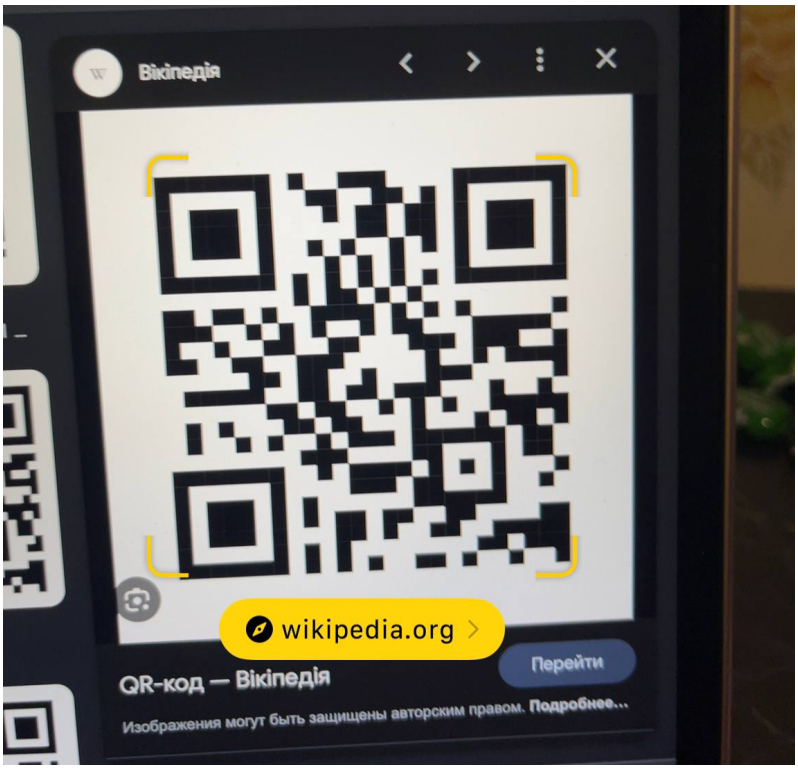

Рисунок 3.9 – Результат сканування

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований мобільний iOS-застосунок, здатний до визначення складу продуктів харчування та сканування QR-кодів.

Дані, необхідні для наданнях релевантної інформації зберігаються локально, що дає змогу користуватися застосунком у разі відсутності зв’язку з мережею Інтернет.

Практичні рекомендації із проведеної роботи полягають у підвищенні рівня відмовостійкості мобільного застосунку за рахунок додаткового покриття Unit-тестами.

Практична значущість роботи – досягнення суттєвого рівня покращення сервісу iOS-застосунків у сфері охорони здоров’я та фітнесу.

Перспективи розробки iOS-застосунків можуть бути пов’язані з випуском нових версій системи iOS та додаванням у них нових інструментів для роботи утиліт.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kuzmin, N., Ignatiev, K., & Grafov, D. (2020). Experience of developing a mobile application using flutter. In Information Science and Applications: ICISA 2019 (pp. 571-575). Springer Singapore.Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for Fast Metric Data Search in Structural Methods for Image Classification. *IEEE Access*, *10*, 124738-124746. p. 6-10.

2. Tashildar, A., Shah, N., Gala, R., Giri, T., & Chavhan, P. (2020). Application development using flutter. International Research Journal of Modernization in Engineering Technology and Science, 2(8), 1262-1266.

3. Biessek, A. (2019). Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2. Packt Publishing Ltd.

4. Meiller, D. (2021). Modern App Development with Dart and Flutter 2. In Modern App Development with Dart and Flutter 2. De Gruyter Oldenbourg.

5. Tyagi, P. (2021). Pragmatic flutter: building cross-platform mobile apps for android, iOS, web & desktop. CRC Press.

6. Payne, R. (2019). Beginning App Development with Flutter: Create Cross-Platform Mobile Apps. Apress.

7. Alessandria, S. (2020). Flutter Projects: A practical, project-based guide to building real-world cross-platform mobile applications and games. Packt Publishing Ltd.

8. Bhangale, P., Bhatt, B., Nandu, M., & Chavda, P. (2021, October). MeetUp: An Appointment Booking System using Flutter and Django Framework: MeetUp: Meetings Made Easy. In 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1425-1431). IEEE.

9. Swathiga, U. U. A. S., Vinodhini, P., & Sasikala, V. An Interpretation of Dart Programming Language. DRSR Journal, 11(3).

10. Jiang, X., Ma, J., Xiao, G., Shao, Z., & Guo, X. (2021). A review of multimodal image matching: Methods and applications. *Information Fusion*, *73*, 22-71.

11. Dobrean, D., & DiOSan, L. (2019, July). Model View Controller in iOS mobile applications development. In SEKE (pp. 547-716).

12. Phan, D. H. (2019). Benchmarking Common Architectural Patterns in iOS Development.

13. Gasic, A., Popic, M., & Gajic, S. (2021, March). Android and iOS Application to Support Tourists in Exploring Belgrade's Museums-MappedCulture: Student paper. In 2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-5). IEEE.

14. Ngaogate, W. (2020, August). Integrating Flyweight Design Pattern and MVC in Development of Web Application. In Proceedings of the 2020 2nd International Conference on Information Technology and Computer Communications (pp. 27-31).

15. Kinoshenko, D., Mashtalir, S., Shlyakhov, V., & Stolbovyi, M. (2019). Video shots retrieval with use of pivot points. In Advances in Computer Science for Engineering and Education 13 (pp. 102-111). Springer International Publishing.

16. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, IEEE Access, 9, pp. 92964-92973, DOI: 10.1109/ACCESS.2021.3093457.

17. Гороховатський, В. О., Стяглик, Н. І., & Жадан, О. В. (2022). Застосування багатокомпонентної моделі даних для описів класів у задачі класифікації зображень. С. 7.

18. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент. С. 4-7.

19. Gorokhovatskyi, V., Tvoroshenko, I., & Chmutov, Y. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. Advanced Information Systems, 6(3), 5-12.

20. Mashtalir S., Mashtalir V. (2020) Spatio-Temporal Video Segmentation. In: Mashtalir V., Ruban I., Levashenko V. (eds) Advances in Spatio-Temporal Segmentation of Visual Data. Studies in Computational Intelligence, vol 876. Springer, Cham. pp. 161-210 .

21. Kinoshenko, D., Kobylin, O., Mashtalir, S., & Stolbovyi, M. (2019, March). Metric video retrieval speedup by irrelevant data elimination. In Eleventh International Conference on Machine Vision (ICMV 2018) (Vol. 11041, pp. 176-183). SPIE.

22. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for Fast Metric Data Search in Structural Methods for Image Classification. IEEE Access, 10, 124738-124746. p. 6-10.

23. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for making reliable decisions on a variety of effective factors using fuzzy logic. International Journal of Advanced Computer Science and Applications, 11(5).

24. Kobylin, O., & Lyashenko, V. (2014). Comparison of standard image edge detection techniques and of method based on wavelet transform.

25. Lyashenko, V., Matarneh, R., Kobylin, O., & Putyatin, Y. (2016). Contour detection and allocation for cytological images using Wavelet analysis methodology.

26. Daradkeh, Y. I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L. A., & Ahmad, N. (2021). Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic. IEEE Access, 9, 13417-13428.

27. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

28. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Al-Dhaifallah, M. (2022). Classification of images based on a system of hierarchical features. Computers, Materials & Continua, 72(1), 1785-1797.

29. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

30. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, *Advances in Electrical and Electronic Engineering*, 21(1), pp. 19-27.