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 МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ СПРОЩЕННЯ РОБОТИ РЕСТОРАННОГО СЕРВІСУ

(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-2

Григор'єв Д.Д.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст.викл. Кіношенко Д.К.
(посада, прізвище, ініціали)

Допускається до захисту

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

2022 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)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Григор'єву Дмитру Денисовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку для спрощення роботи ресторанного сервісу

затверджена наказом університету від 16 травня 2022 року № 541Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23 травня 2022 р.
3. Вихідні дані до роботи застосунок який виконує функцію електронного меню, інструкція для користувача, теоретичні відомості про особливості розробки, тестування застосунку, середовище розробки XCode 12.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих аналогів.
2. Особливості розробки мобільних застосунків.
3. Дослідження принципів UI/UX проектування.
4. Дослідження можливостей смартфонів компанії Apple.
5. Розробка функціоналу застосунку.
6. Порівняння якості роботи розробленого застосунку з аналогами.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ключові частини коду, схема моделей даних за синтаксисом Чена, таблиці бази даних, тестова база даних, скріншоти готового застосунку, опис можливостей користувача, дослідження результатів, висновки.

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 Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технічних і програмних засобів	26.04.22-30.04.22	
5	Розробка методу	01.05.22-14.05.22	
6	Програмна реалізація	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	28.05.22	
9	Рецензування	29.05.22	
10	Підготовка презентації та доповіді	29.05.22-30.05.22	
11	Занесення роботи в електронний архів	31.05.22	
12	Попередній захист кваліфікаційної роботи	31.05.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ ____ст.викл. Кіношенко Д.К. _
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 58 с., 31 рис., 32 джерел.

IOS, APPLE, SWIFT, XCODE, ЛОКАЛЬНА БАЗА ДАНИХ, COREDATA, FIREBASE, АРХІТЕКТУРА VIPER, АРХІТЕКТУРА MVC, CORE LOCATION, QR.

Об’єктом роботи є процес розробки електронного меню закладів харчування для мобільних пристроїв на базі системи iOS.

Метою роботи є розробка застосунку, що буде використовуватися у закладах харчування за для спрощення роботи офіціантів та зменшення використання паперових виробів у сфері харчування.

У результаті роботи розроблено застосунок для мобільних пристроїв на базі iOS який використовується у закладах харчування як електронне меню.

IOS, APPLE, SWIFT, XCODE, LOCAL DATABASE, COREDATA, FIREBASE, VIPER ARCHITECTURE, MVC ARCHITECTURE, CORE LOCATION, QR

The object of the work is the process of developing an electronic menu of restaurants for mobile devices based on the iOS system.

The aim of the work is to develop an application that will be used in catering establishments to simplify the work of waiters and reduce the use of paper products in the food industry.

As a result of work, an application for mobile devices based on iOS was developed, which is used in restaurants as an electronic menu.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SDK – набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи (програмної або програмно-апаратної).

IDE – інтегрована середа розробки.

IB – Interface Builder

ВСТУП

Сучасний світ складно уявити без присутності в ньому сучасних смартфонів, якими усі люди щодня користуємося. В цих смартфонах люди щодня користуються різними застосунками, такими як - месенджери, пошта, соціальні мережі, професійні застосунки (photoshop, калькулятор, gps, AR застосунки, тощо), замітками, іграми і багато чим ще. З кожним новим днем все складніше уявити життя без гаджетів, які допомагають щодня і будуть допомагати та спрощувати використання якихось речей з кожним днем. Наприклад – газети, які раніше потрібно було друкувати кожен день, витрачати купу паперу та чорнил, використовувати купу друкувальних станків, потім розвозити ці газети, витрачалась купа грошей, було задіяно багато людей, зараз же можна просто завантажити додаток новин і бути у центрі всіх світових подій. Дана робота має стати таким самим замінником старих традицій на нові, сучасні та технологічні, за прикладом газет у свій час але тепер з меню у закладах харчування та тим самим, поліпшення роботи з клієнтом цього самого закладу.

За допомогою цього застосунку клієнт зможе:

– сканувати QR-код з меню без допомоги офіціанта;

– подивитися запропоновані в меню блюда з фотографіями та детальним описом що є у составі цього блюда;

– замовити блюдо без офіціанта;

– замовити оплату чеку або розплатитися карткою онлайн;

– не чіпляти меню руками (актуально за часи епідемії COVID);

– вибрати блюдо заздалегідь (наприклад, коли клієнт їде до закладу на таксі).

Сучасні засоби програмування дозволяють зробити красивий та зручний дизайн програми, таким засобом є SwiftUI та storyboard. Завдяки цим інструментів можливо легко і швидко розробити iOS застосунок.

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ З ПРОВАДЖЕННЯМ СУЧАСНИХ ТЕХНОЛОГІЙ У ЕЛЕКТРОННЕ МЕНЮ

Дослідження та аналіз вже розроблених та працюючих програмних систем є дуже важливим етапом в плануванні та формулюванні точного технічного завдання, збору вимог до програмного продукту та проектуванні системи.

Без даного етапу теоретичної підготовки до розробки програми, можна розробити застосунок, який вже існує і використовується через що розроблений застосунок не буде використовуватися зовсім. Завдяки аналізу можна зробити ті ж самі помилки, що і програмісти, які розроблювали ці застосунки. Також це є гарним способом для виявлення корисних функцій або інтерфейсних рішень.

Таке рішення для меню дуже популярне у наші часи, тому було розроблено багато рішень електронного меню. Як тільки закінчився найжорсткіший період локдауна і почали відкриватися заклади харчування, майже всі перейшли на електронне меню і воно виявилось зручнішим, ніж звичайне, паперове.

1.1 Існуючі програмні засоби

На даний момент існує багато аналогів електронного меню, але ці всі популярні аналоги не є програмними засобами. Але є і програмні застосунки, наприклад застосунок харківського відкритого закладу харчування 7йсклад, але його приклад зараз неможливо продемонструвати, так як на період даний період сервера застосунку не працюють.

Зазвичай, меню які використовують заклади харчування– це посилання на веб застосунок, закріплена історія Instagram, пост у Facebook, Google Docs тощо. Наприклад, найпопулярніший сервіс для розробки електронного меню це OddMenu (вебзастосунок).

OddMenu – платний веб застосунок, в якому ресторатор має просто додати свої блюда, розділити його по підгрупам (рис. 1.1). Ресторатору потрібно представити базу даних, а все інше зробить сам сайт.

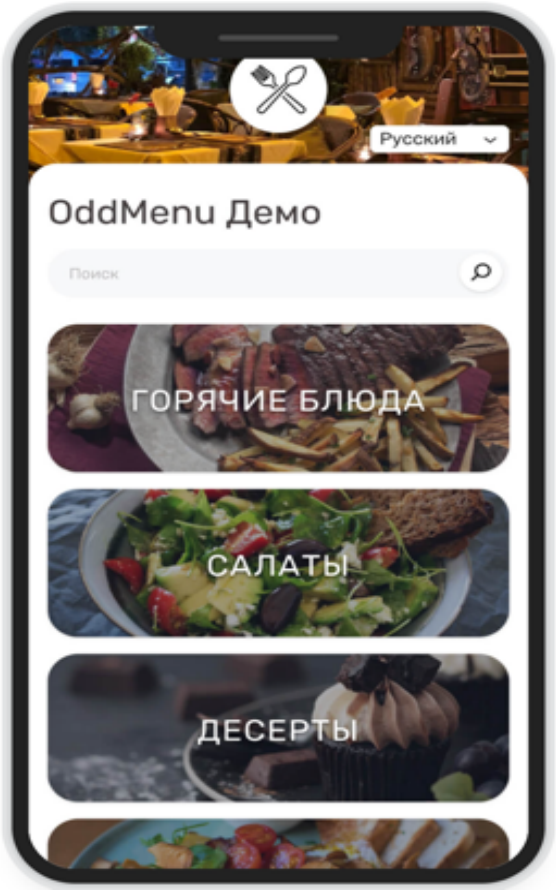

Рисунок 1.1 – Зовнішній вигляд OddMenu

Розглянемо основні переваги на недоліки даного рішення.

Переваги:

- зручний та красивий дизайн;
- простий у використанні;
- простий для розробки (з боку ресторатора);
- дешева підписка (близько 10$ на місяць).

Недоліки:

– кожного разу, коли ви будете приходити до закладу, вам потрібно завантажити меню;

– неможливість подивитись меню без сканування QR-коду;

– для замовлення все одно потрібно чекати офіціанта.

1.2 Постановка задачі

Об’єктом роботи є процес розробки електронного меню закладів харчування для мобільних пристроїв на базі системи iOS.

Метою роботи є розробка застосунку, що буде використовуватися у закладах харчування за для спрощення роботи офіціантів та зменшення використання паперових виробів у сфері харчування.

Для досягнення мети необхідно вирішити такі завдання:

– розробити додаток для платформи iOS з використанням усіх вище наведених інструментів;

– використати для розробки мову програмування Swift та середовище розробки XCode;

– розробити інтуїтивно зрозумілий інтерфейс додатка;

– розробити базу даних для загрузки меню з ресторанів до додатку;

– розробити реєстрацію нового користувача та автентифікацію вже існуючого профіля

– реалізувати систему підтвердження присутності користувача саме у цьому закладі;

– реалізувати сканування QR-коду з меню;

– реалізувати систему оплати замовлення.

1.3 Етапи розробки мобільного застосунку

Розробка застосунку розділена на три основні кроки. Перш ніж перейти до детальної версії, проаналізуємо етапи процесу розробки програми. Мобільний додаток побудовано на трьох основних компонентах, тобто серверних технологіях (Backend), API (інтерфейсах прикладного програмування) і розробки інтерфейсу (Frontend). Елементи, які користувачі бачать і з якими взаємодіють – це інтерфейс програми, розробка для роботи на стороні сервера – це бекенд, а набір функцій, який дозволяє додаткам отримувати доступ до даних і взаємодіяти з зовнішніми програмними компонентами, операційними системами або мікросервісами – це API.

Першим треба розробити Frontend, тобто інтерфейс додатку, далі додати усе API, а вже потім розробити серверну частину.

2 ОГЛЯД ОСНОВНИХ ІНСТРУМЕНТІВ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ

Зараз дуже важливо, щоб система, яка проектуються та розроблюється, була доступна до використання завжди, двадцять чотири години на добу. Важко уявити сучасну людину без смартфону. Найпопулярнішим смартфоном у світі прийнято вважати саме iPhone, тому саме і обрано розробку даної системи для платформи iOS.

Вважається, що операційна система від компанії «Apple» є кращою серед мобільних операційних систем, так як вона є закритою та дуже надійною, що мінімізує риски хакерською атаки на цю операційну систему, саме цим компанія гарантує надійність і безпеку для особистих даних користувача. Так як компанія «Apple» є найбагатшою приватною компанію у світі, в систему iOS вкладуть великі кошти, щоб тримати заданий гідний рівень якості.

2.1 Призначення та користувачі

Дане програмне забезпечення призначення для поліпшення і часткової автоматизації обслуговування клієнтів у сфері харчування. За допомогою даного застосунку користувач зможе не чекати офіціанта, який можливо зайнятий, а зможе самостійно вибрати та замовити обрану страву. Тобто користувачу по-перше не потрібно чекати доки офіціант піднесе меню, щоб переглянути і обрати страву, по-друге не потрібно чекати офіціанта для того, щоб переглянувши меню, зробити замовлення, по-третє, можливо розрахуватися за допомогою картки не чекавши офіціанта. Таким чином, завдяки цьому застосунку, заклад може зменшити кількість офіціантів, придбавши замість цього застосунок і обслуговування застосунку.

Кінцевими користувачами можуть бути люди які часто бувають у закладах харчування, а також часто користуються смартфоном. Це можуть бути люди різних професій та різного віку.

2.2 Задачі які розв’язує програмне забезпечення

Першу задачу описано у пункті 1.1, а саме зменшення кількості офіціантів та поліпшення та прискорення обслуговування кліентів.

Другою задачею яку вирішує даний застосунок є зменшення використання паперу на землі, у тому числі у ресторанному бізнесі. По-перше, для надрукування меню використовується паперові вироби, по-друге для надрукування чеку використовуються паперові вироби, обидві ці питання вирішує даний застосунок тим, що він і є меню, чим зменшує використання паперу, а чеки надсилаються електронним листом, що також зменшує використання паперу. Кожен раз коли меню змінюється або доповнюється, потрібно друкувати нове меню, навіщо це робити, якщо можна змінити декілька строчок коду?

Третя задача це зменшення контактування з навколишніми загальними предметами, що зменшує ризик захворіти вірусом, який передається завдяки безпосереднім контактом з поверхнями (актуально при пандемії COVID).

2.3 Сучасні інструменти для розробки мобільних застосунків

2.3.1 Середовище розробки XCode

Компанія «Apple» пропонує лише одне середовище розробки для створення додатків для операційнїх систем iOS, MacOS, iPadOS – це середовище має назву XCode.

XCode – інтегроване середовище розробки (IDE) власного виробництва компанії «Apple». Даним IDE можливо користуватися лише на ноутбуках чи комп'ютерах виробництва цієї ж компанії. Перша версія випущена у 2001 році наступні версії випускаються кожен рік після великої конференції, яку проводять у Каліфорнії – «WWDC». Усі стабільні версій поширюються безкоштовно через вбудований у систему магазин «App Store». XCode включає в себе велику частину документацій розробника від Apple [1].

В XCode вбудовано Interface Builder – застосунок, що використовується для створення графічних інтерфейсів. В пакеті Xcode міститься великий набір компіляторів – GNU Compiler Collection і підтримує мови C, C++, Objective-C, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи (але не обмежуючись) Cocoa, Carbon і Java.

2.3.2 Мова програмування Swift

Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple на заміну більш старої та менш гнучкої мови програмування – Objective C. Swift співіснує з мовою Objective C, але більш стійкіша до помилкового коду. Swift була представлена на конференції розробників «WWDC 2014».

Розробку мови Swift почав Chris Lattner у співпраці із багатьма іншими програмістами. Ідеї для Swift запозичені із таких мов програмування як Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, та багатьох інших зі списку. Так як мова використовується багатьма програмістами по усьому світі, ця мова вже має за плечима роки розвитку, і вона продовжує розвиватися, включаючи в себе все нові і нові можливості [1].

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас

відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду [2].

Мова пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. Ця мова є дуже сучасною та зручною для використання. Основним та можна сказати єдиним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням бібліотек Cocoa, Cocoa Pods та Cocoa Touch. При цьому мова Swift надає об'єктну модель та використовує усі методи ООП, ця мова сумісна з Objective-C. Код мовою Swift можна змішуватися з кодом на C, Objective-C та C++ в одному проекті, просто підключаючи бібліотеки цих мов [2].

Swift – перша потужна мова програмування, яка використовує найліпші функції інших мов, легко зрозуміла і захоплююча, як скриптова мова. Swift підтримує так звані playground-и, схожі на проектування на мові HTML або C# які дозволяють програмістам експериментувати з кодом, завдяки цьому результат роботи можна бачити в режимі реального часу без необхідності компілювати і запускати додаток [2].

Swift виключає велику кількість поширених програмних помилок за допомогою застосування сучасних програмних паттернів.

2.3.3 Конструктор інтерфейсів XCode

Interface Builder – інструмент розробки від Apple для операційної системи MacOS. Цей інструмент є частиною Xcode, розроблений для спеціальної системи інструментів для розробників Apple Developer Connection [3].

Interface Builder вперше з’явився в 1986 році і був розроблений за допомогою мови Lisp. Interface Builder був розроблений і задуманий Джин-Мері Холлотом з використанням усіх інструментів об'єктно-орієнтованого програмування в ExperLisp і інтегрований з інструментами які використовує Macintosh [4].

Interface Builder надає палітри та колекції, об'єктів інтерфейсу користувача для Objective-C і Swift розробників. Ці об'єкти інтерфейсу містять елементи текстових полів, таблиці даних, слайдери і спливаючі меню. Палітри Interface Builder є повністю розширюваними, за рахунок чого будь-який розробник може розробляти нові об'єкти і додавати їх до палітри у IB.

Для створення та розробки інтерфейсу, розробник може перетягувати елементи інтерфейсу з палітри на вікно або storyboard. Об'єкти, які отримують повідомлення вказуються в програмному коді застосунку. Таким чином всі ініціалізації відбуваються до виконання, що призведе до підвищення продуктивності будь якого застосунку, розробленого за допомогою IB.

2.3.4 Фреймворк Core Data

Core Data – фреймворк від компанії Apple, який є частиною операційної системи iOS та MacOS, який дозволяє розробнику взаємодіяти з локальною базою даних. Був представлений компанією Apple з анонсом MacOS X 10.4 Tiger і iOS з iPhone SDK 3.0 на конференції «WWDC» [5].

Цей інструмент дозволяє даним бути організованими в Сутність-Атрибут-Значення. Управляти цими даними просто, за допомогою маніпуляцій сутностями, як об’єктами звичайних класів, створених безпосередньо в коді [6].

Core Data описує дані, які зберігаються в iOS додатку, код може використовувати для збереження і записи даних в додатку. Модель БД

створюється в Interface Builder. Код пишеться на Objective-C або Swift. Core Data організований в величезні класи інструментів.

Структура Core Data:

– Managed Object Context – компонент з яким відбувається взаємодія, кожен раз, коли йде збереження або оновлення;

– Persistent Store Coordinator – виконує зберігання даних. Посередник між базою даних і контекстом;

– Persistent Store – сховище, де зберігаються дані попередньо записані за допомогою Core Data;

– Managed Object Model - модель бази даних.

Core Data може конвертувати дані в формат XML, бінарний код або SQLite, конвертування робиться для зберігання в електронному девайсі на яких встановлена операційна система iOS або macOS. Якщо на комп'ютері відсутній Xcode – завжди можна переглянути файли бази даних. Ця опція розповсюджується лише на власників комп'ютерів від компанії «Apple».

Основна частина фреймворка була написана під час роботи Стіва Джобса в компанії NeXT, Enterprise Objects Framework (EOF) на мові Objective-C [7].

Хоча даний інструмент на перший погляд не має мінусів, існує один великий недолік при роботі з даним інструментом – це швидкість виконнаня методів об'єктами з бази даних. Даний недолік пояснюється тим, що класи з Core Data використовують Objective-C Runtime, який уступає в швидкості сучасним аналогам цієї бази даних.

Також замість статичної диспетчеризації використовується динамічна, що теж робить свій внесок в швидкість роботи та виконання методів об'єктами цих класів.

2.3.5 Менеджер залежностей Cocoa Pods

CocoaPods – менеджер залежностей для проектів, які використовують в своєму написані мови Swift або Objective-C [8].

Даний менеджер містить в собі тисячі бібліотек і допомагає розробникам легко та швидко масштабувати проекти. Основна задача цього менеджера залежносте – покращити доступність та участь в сторонніх бібліотеках, код яких знаходиться у відкритому доступі. Також задачей CocoaPods є створення єдиної централізованої системи залежностей у проекті.

Залежності для проєктів вказуються в текстовому файлі, який називається «Podfile». Також існує файл залежностей з розширенням .lock, цей файл оновлюється кожен раз, коли вносяться зміни в основний Podfile.

Цей менеджер є дещо схожим на Maven для Java, або Ruby Gems для Ruby, ідеї яких близькі до ідей закладених до CocoaPods. Основна ідея цього менеджеру – після того, як додається нова бібліотека до проекту, не потрібно слідкувати за її оновленнями, за цим слідкує менеджер залежностей, він буде виконувати все це автоматично без необхідного втручання розробника.

Досить важливим є також те, що CocoaPods має прямий зв’язок з основним файлом проекту – .xcodeproj. Це обумовлено тим, що використання сторонніх бібліотек в середовищі розробки Xcode вимагає формування нового файлу запуска проекта – .xcworkspace.

2.3.6 UIKit

UIKit – це один з основних та фундаментальних інструментів розробки додатків для платформи iOS. Без нього неможливо розробити інтерфейс застосунка, тому він використовується для побудови інтерфейсів.

Класи які містяться в даній бібліотеці:

– допоміжні елементи – UIEvent, UIFont, UIColor, UIDevice, UIScreen, UIImage [8, 9];

– макети, які можуть прокручуватись UIScrollView , UITableView, UICollectionView [8, 9];

– макети, які можуть прокручуватись UIScrollView , UITableView, UICollectionView [8, 9];

– контроли – UITextField, UIButton, UIDatePicker, UIPageControl, UISlider [8, 9];

– візуальні елементи – UIWindow, UILabel, UIPickerView, UIImageView, UIWebView, UIToolbar [8, 9];

– контролери видів – UIViewController, UINavigationController, UITabBarController [8, 9];

– відповідачі – UIResponder та UIApplication [8, 9].

За допомогою цих класів в основну і будується весь інтерфейс. Наприклад, клас UIViewController використовується в будь-якому модулі в якому є відображення даних на екрані. Компанія «Apple» радить використовувати саме контролери видів з цієї бібліотеки, тому що життєвий цикл контролера тісно пов’язаний з життєвим циклом вида.

Основою будь-якого додатка для платформи iOS становлять таблиці за даними. Це обумовлюється тим, що це дуже зручний елемент для відображення великої кількості схожих даних. В даній бібліотеці передбачена можливість відображення таблиць двох видів – згруповану таблицю та плоску. Також, кожна запропонована таблиця може мати свій заголовок зверху таблиці та колонтитул знизу. Ці елементи можуть мати властивість закріплення на заданій позиції, яка під час прокручування буде залишатися на одному місці.

В деяких випадках потрібно використовувати UICollectionView, якщо потрібно організувати горизонтальне прокручування або ж потрібно розмістити елементи більш ніж в один стовбець.

В бібліотеці UIKit вже передбачені чотири типи комірок для використання в таблицях, за рахунок чого розробникам не потрібно витрачати зайвий час на побудову деяких базових речей [10].

Комірки доступні в UIKit по замовчуванням:

– Basic (Звичайна комірка з одним полем для тексту);
– Subtitle (Комірка з двома текстовими полями, один з яких менший);
– Right Detail (Комірка з іконкою в правій стороні);
– Left Detail (Комірка з іконкою зліва).

Також передбачена можливість додати навігаційне або пояснювальне зображення для будь-якого з цих типів комірок.

За допомогою об’єктів класу типу UIResponder відбувається відстежування жестів та дій користувача. Методи цього класу викликаються на першому об’єкті (виді) по ієрархії та йому передається відповідна дія або жест. Якщо даний об’єкт не може опрацювати дію або жест, наприклад тому що, об’єкт не доступний, або невидимий (атрибут даного об’єкта – IsHidden = true) , то тоді дія або жест передається наступному елементу по ієрархії.

2.3.7 Firebase

Firebase – це платформа розробки мобільних програм з величезним функціоналом. Починалася вона як стартап, а сьогодні її використовують при розробці найкращих кросплатформових додатків [11].

Головне достоїнство платформи у тому, що вона дозволяє розробнику не відволікатися на створення серверної частини застосунку. Це спрощує і прискорює створення мобільних додатків, дає можливість повністю зосередитися саме на UX/UI, тобто, на інтерфейсі користувача і досвіді, а не на написані складного та великого серверного коду.

Firebase має багато функцій; такі як Firebase Analytics, Firebase Cloud Messaging, Firebase Auth, Firebase real-time database, Firebase Storage, Firebase

hosting and function. У даній роботі використовується лише Firebase Auth та Firebase real-time database [12]. В майбутньому, завдяки гнучкості програмного коду та нескінченими можливостями вдосконалення застосунку, можливо буде використовувати також інші інструменти Firebase.

2.3.7.1 Firebase Auth

Firebase Auth – це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта [12]. Він підтримує такі соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase. В цій програмі користувачи не авторизуються, вхід до застосунку виконується анонімно, але кожен анонімний користувач фіксується з унікальним ID та геолокацією до Firebase auth system.

2.3.7.2 Firebase real-time database

Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase.Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосунками Swift та інших мов програмування. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері.

В даному застосунку цей сервіс використовуються з метою імітування ресторанної системи, тобто системи внесення замовлення до комп’ютера. При

підтвердженні замовлення, це замовлення передається до Firebase real-time database з усіма даними замовлення: локацією клієнта (задля підтвердження знаходження клієнта саме у цьому закладі), кількість замовлених страв, суми до сплати.

2.3.8 Core Location

Core Location надає послуги, які визначають географічне розташування, висоту та орієнтацію пристрою або його положення відносно сусіднього пристрою iBeacon. Фреймворк збирає дані за допомогою всіх доступних компонентів пристрою, включаючи Wi-Fi, GPS, Bluetooth, магнітометр, барометр і стільникове обладнання [13, 14].

В цій роботі Core Location використовується для відстежування локації користувача та вирахування відстані між локацією користувача та локацією закладу харчування.

2.4 Технологія проектування Viper

Для проектування iOS застосунку за використання стилю «Clean Architecture», який було запропоновано компанією «Rambler & Co» було вирішено використовувати архітектуру «VIPER», що на даний момент вважається ліпшим рішенням для проектування з використовуванням стилю «Чиста архітектура». Чиста архітектура використовується щоб ділити логічну структуру програми на різні рівні обов'язків. Завдяки цьому спрощується ізолювання залежностей та тестування взаємодії на межах різних рівнях, підвищує рівень абстракції, знижує зв'язаність коду та вирішує проблеми маршрутизації у програмному коді [14].

Зв'язки в даній архітектурі (рис. 2.1):

– View – відповідає за відображення даних на екрані та оповіщає Presenter про дії користувача. Пасивний, сам ніколи не запитує дані, тільки отримує їх від представника [15];

– Interactor – взаємодіючий – містить всю бізнес-логіку, потрібну для роботи поточного модуля [15];

– Presenter – представник отримує від View інформацію про дії користувача і перетворює її у запити до Router'a, Interactor'a, а також отримує дані від Interactor'a, готує їх і відправляє View для відображення [15];

– Entity – сутність об'єкти моделі, що не містять жодної бізнес-логіки [15];

– Router – маршрутизатор відповідає за навігацію модуля [15].

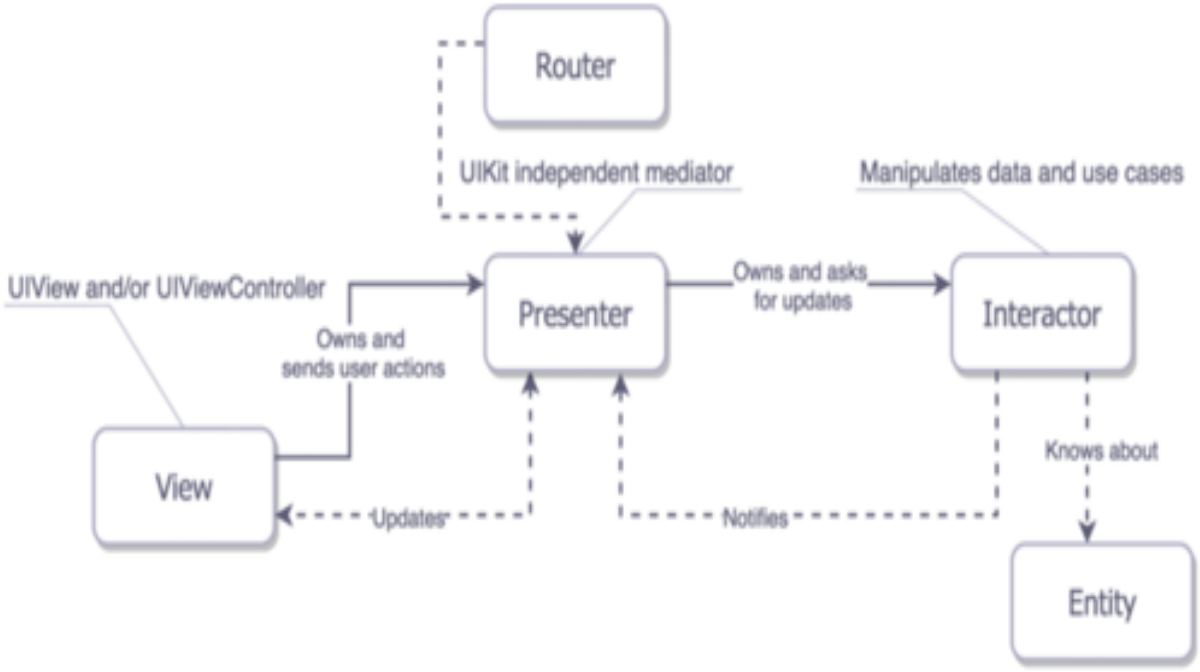

Рисунок 2.1 – Архітектура Viper

Вид має у собі посилання на протокол (інтерфейс) представника. Представник, у свою чергу має містити послання на протокол виду, що взаємодіє та маршрутизірує. Маршрутизатор має у собі посилання на вид для забезпечення переходів між модулями. Взаємодіючий – знає про існування

представника за допомогою протоколу. Він містить у собі необхідні сервіси для реалізації бізнес-логіки застосунку.

Перевагою такого модуля також можна назвати те, що він може бути покритий тестами на 90-100%.

Головні характеристики архітектури VIPER:

– допомагає слідувати SOLID принципам при розробці [16];

– полегшення написання тестів;

– дає можливість покривати тестами модуль на 90-100%;

– закриває кожен прошарок проколом (інтерфейсом), тобто напряму прошарки не можуть взаємодіяти між собою;

– зменшує швидкість розробки через опис додаткових сутностей модулів системи.

2.5 Технологія проектування MVC

Модель-вигляд-контролер (або Модель-представлення-контролер. Model-view-controller, MVC) – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення [17].

Цей шаблон передбачає розділення системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль управління. Цей архітектурний шаблон застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мали мінімальний вплив на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача [17].

В даному випадку використовувався стандартний шаблон MVC від компанії «Apple» з пасивною моделлю для відображення даних. Розглянемо цей шаблон у модулі додавання нової відповіді на запитання (рис 2.2).

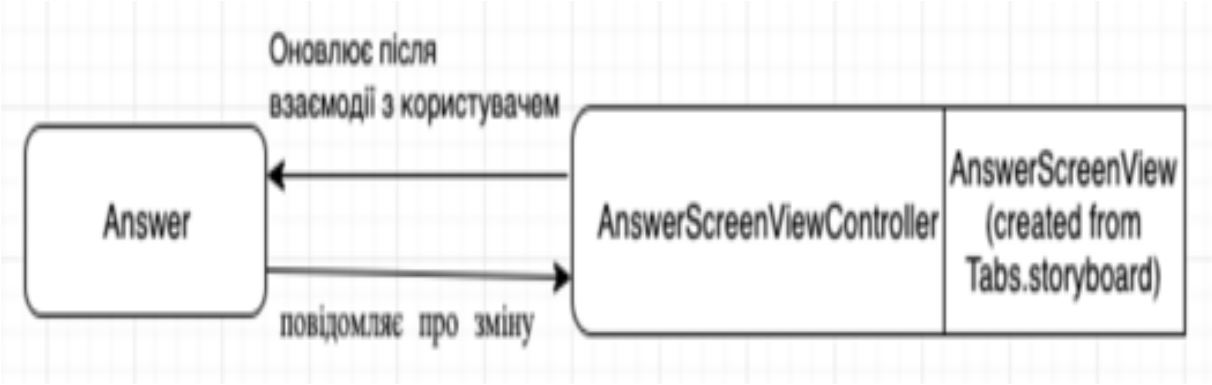

Рисунок 2.2 – Архітектура MVC для модуля додавання нової відповіді

Голови характеристики архітектури MVC від Apple:

– з початку розробки та збільшення її швидкості – складно тестувати великі модулі;

– легкість для розуміння;

– переважно використовується з пасивним модулем.

Саме через ці характеристики було вирішено використовувати дану архітектуру тільки в модулях, які відповідають за відображення даних і не виконують жодної складної роботи.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В наступних розділах буде розглянуто основні аспекти реалізації даного програмного забезпечення

3.1 Архітектура додатку

Проектування системи є головним етапом у розробці застосунків з того часу, як програмування стало об'єктно орієнтованим. Ця тенденція обумовлена важливістю наявності засобів для швидкого розширення програмного продукту без пошкодження чи зміни вже існуючого функціоналу.

В результаті розробки застосунку FoodApp, проект застосунку має наступний вигляд (рис. 3.1).

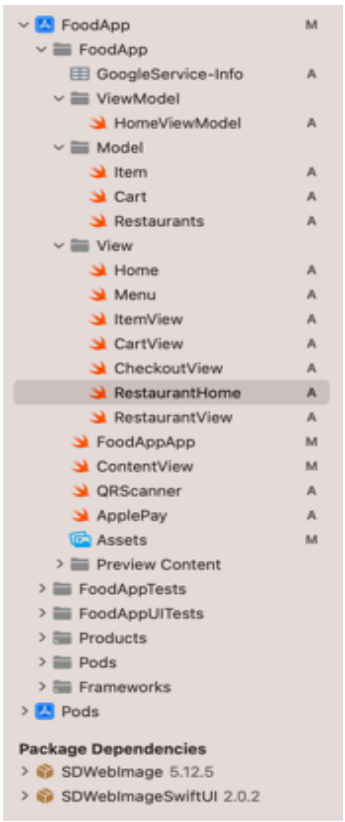

Рисунок 3.1 – Древо застосунку

У розробці використовувалась технологія проектування Viper та MVC [18].

Проект складається з шістнадцяти файлів, із яких 3 були розроблені за шаблоном Viper, десять за шаблоном MVC, та 3 файли з даними, які використовуються у обох технологіях проектування.

Класи в модулях згруповані та взаємодіють між собою за шаблонами проектування VIPER та MVC. Шаблон VIPER був вибраний для модулів, які вимагають містити в собі багато різного функціоналу, наприклад – завантаження даних та запис їх до локальної бази даних. Шаблон MVC обирався для модулів, які мали відповідати лише за відображення даних.

3.2 Структура бази даних

Як уже описувалося раніше, для роботи з локальною базою даних було рішення від компанії Google – Firebase [19].

Сутності в базі даних також була описана за допомогою конструктора інтерфейсів вбудованого в середу розробки Firebase.

Кожна сутність має свій унікальний набір даних, деякі сутності зв’язані між собою за допомогою Foreign Key [20]. Сутність Restaurants (табл. 3.1) має унікальний айді, назву ресторану, локацію ресторану, короткий опис, зображення ресторану та рейтинг ресторану.

Таблиця 3.1 – Опис сутностей ресторанів в базі даних.

Ресторани(Restaurants)	
Атрибут	Тип даних
ID_restaurant	NSNumber
Item_name	String
Item_location	NSNumber
Item_details	String

Продовження таблиці 3.1

Item_image	String
Item_ratings	String

Сутність меню ресторану (табл. 3.2) вимагає містити в собі атрибут, в якому буде зберігатися посилання на приналежність цього меню до конкретного ресторану. Також, сутність меню має унікальний id блюда, назву, вартість, посилання на зображення, опис, популярність (рейтинг).

Таблиця 3.2 – Опис сутностей меню в базі даних.

Меню(menu)	
Атрибут	Тип даних
ID_dishes	NSNumber
Item_name	String
Item_details	String
Item_ratings	NSNumber
Item_image	String
Item_cost	String
Id_restaurnt	Foreign Key (from restaurants)

Сутність користувачі (табл. 3.3) має лише два поля, це унікальне id користувача та його поточна локація. Ця сутність потрібна лише за для того, щоб підтвердити присутність користувача у ресторані, тому що можливість замовити є лише коли локація користувача знаходиться поруч з локацією ресторану. Так як користувач не реєструється у застосунку, користувач залишається анонімним. Ця таблиця також зв’язана з іншою таблицею (табл. 3.5), коли користувач робить замовлення, через Foreign Key ресторан зможе дізнатися де саме знаходиться користувач та чи існує цей користувач, яи підтверджувати замовлення.

Таблиця 3.3 – Опис сутності користувачей в базі даних.

Користувач(Users)	
Атрибут	Тип даних
ID_user	NSNumber
Location	NSNumber

Сутність замовлення (табл. 3.4) має три атрибути, один з яких Foreign key. Сутність має унікальне id замовлення, foreign key з таблиці menu з усім описом блюда та кількість їжі в замовленні.

Таблиця 3.4 – Опис сутності замовлення в базі даних.

Замовлення(order)	
Атрибут	Тип даних
ID_order	NSNumber
Quantity	NSNumber
ID_dishes	Foreign key

Також була розроблена допоміжна сутність (табл. 3.5), яка буде передавати інформацію на термінал до ресторану, ця сутність складається з двох foreign key та одного власного атрибута. Власний атрибут це загальна вартість, яка виліковується за допомогою функції у файлі проекту і передається у цю таблицю.

Таблиця 3.5 – Опис сутності допоміжного замовлення у базі даних.

Замовлення користувача(User_order)	
Атрибут	Тип даних
ID_order	Foreign key from order
ID_user	Foreign key from User
Total_cost	NSNumber

Проаналізувавши всі вимоги, всі наведені критерії, можна розробити повну картину структури бази даних даного проекту, яка буде містити в собі всі таблиці і показувати зв’язані дані (рис. 3.2). За допомогою складеної бази даних, до застосунку можливо буде легко додавати нові дані, наприклад, змінювати вартість товару або додавати нові страви чи ресторани. Розроблений програмний код буде самостійно адоптувати дані до застосунку, тобто від ресторатора потрібно лише заповнити дві таблиці, це таблицю ресторан та таблицю меню. Для побудови даної діаграми, було використано безкоштовний онлайн застосунок.

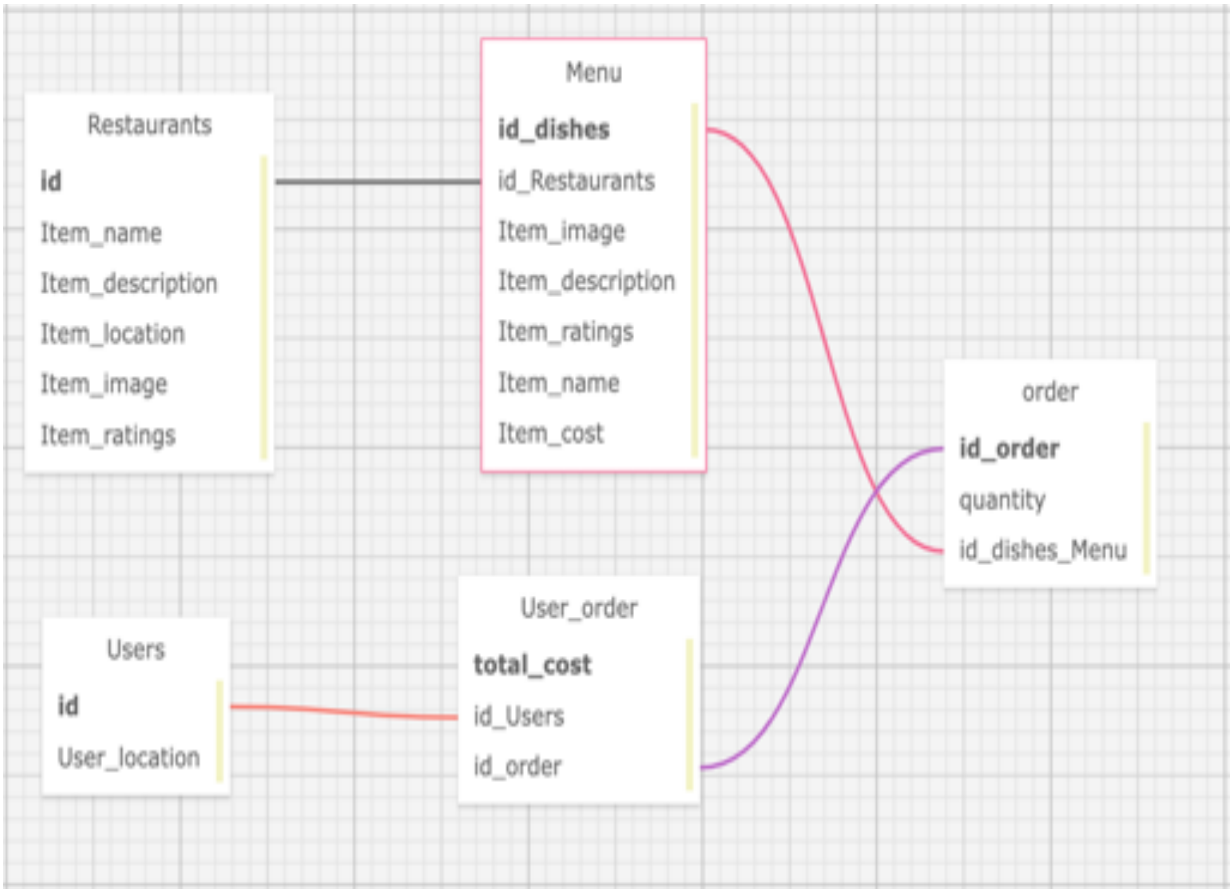

Рисунок 3.2 – Діаграма бази даних

Для унікальних ідентифікаторів було обрано використовувати тип String, оскільки він займає менше пам'яті в базі ніж тип UUID. Але при записі об'єкта використовується UUID зведень типу String. Дана операція не є дуже складною та затратною за ресурсами для операційної системи iOS.

3.2.1 Встановлення SDK для підключення до бази даних

Для того щоб можливо було підключитися і використовувати базу даних Firebase з середовищем розробки XCode, потрібно завантажити та встановити декілька SDK пакетів та бібліотек [21]. Усі SDK були встановлені за допомогою термінала вбудованого у операційну систему MacOS.

Перший SDK який було встановлено, це SDK Cocoa Pods (рис. 3.3).

Далі був редагований pod file та за рахунок цього встановлено Firebase SDK, що допомогло підключатися до бази даних і взаємодіяти з нею в режимі онлайн. Останнім SDK став пакет, який допомагає завантажувати програмі зображення лише за посиланням на URL фотографії. Такий метод був обраний задля того, щоб лише продемонструвати що не обов'язково додавати зображення напряму до бази даних, а можна лише залишити посилання на фото, чим програміст розвантажує систему бази даних.

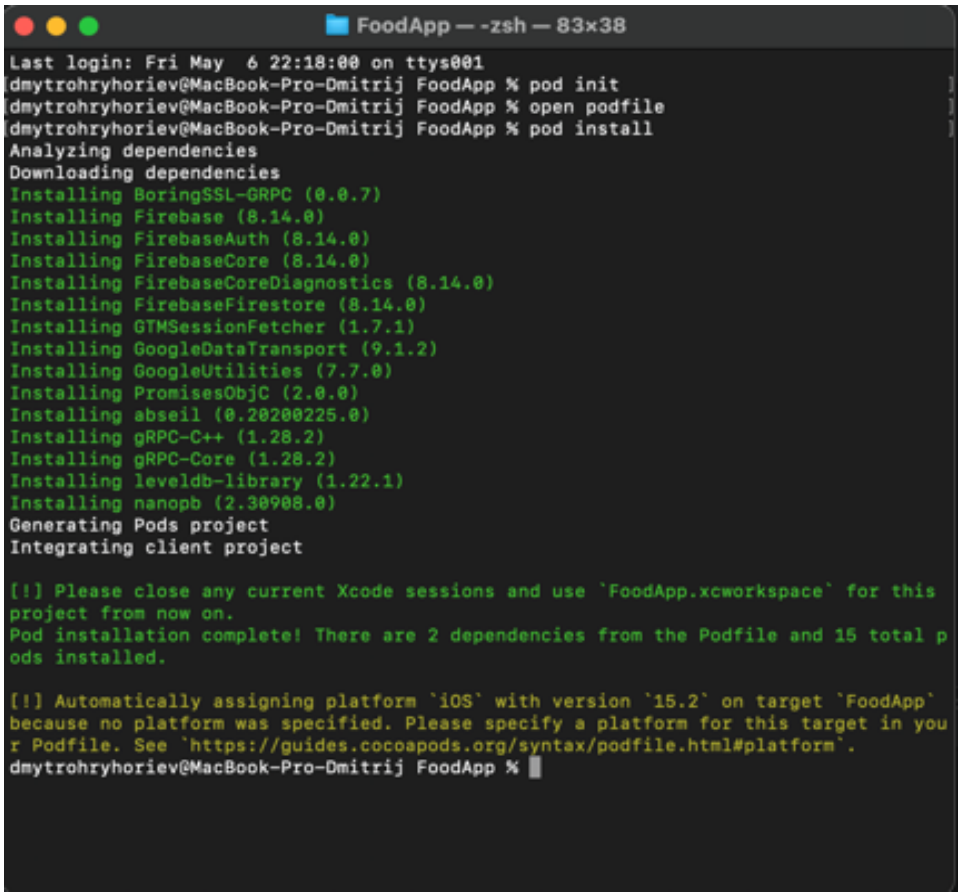

Рисунок 3.3 – Встановлення cocoa pods та firebase SDK через термінал

3.2.2 Підключення до бази даних

Для підключення до бази даних розроблена окрема функція, яка спрацьовує коли користувач заходить до застосунку. Щоб підключитися до бази даних, спочатку декларується перемінна типу constant з шляхом підключення до бази даних, потім декларується саме таблиця, яка має завантажитись першою. Так як при старті застосунку немає ніякої реєстрації, а є лише анонімна автентифікація, першою буде заповнюватися таблиця users, яка містить локацію та унікальний id користувача, потім, завантажується таблиця restaurants, бо це перші дані які побачить користувач на екрані. Таблиця restaurants містить лише дані ресторанів, чиє меню можна подивитися саме в цьому застосунку, тому, таблиця restaurants завантажуються першою (рис. 3.4). Після завантаження першої таблиці, при натисканні на кнопку «See menu» буде завантажуватися таблиця з меню саме цього ресторану, який вирішив подивитися користувач.

```
func fetchData(){

    db.collection("Restaurants").getDocuments{(snap,err) in
        guard let itemRestaurantData = snap else{return}

        self.restaurantItems = itemRestaurantData.documents.compactMap{ (doc) -> Restaurants? in

            let restaurantId = doc.documentID
            let restaurantName = doc.get("item_name") as! String
            let restaurantLocation = doc.get("item_location") as! String
            let restaurantImage = doc.get("item_image") as! String
            let restaurantDetails = doc.get("item_details") as! String

            return Restaurants(id: restaurantId, item_name: restaurantName, item_location: restaurantLocation, item_details: restaurantDetails,
                item_image: restaurantImage)

        }
        self.restaurantFiltered = self.restaurantItems

    }
```

Рисунок 3.4 – Завантаження з бази даних таблиці restaurants

Таблица Menu завантажується за тією самою логікою, що і таблиця restaurants, але в додаток є одна умова, це ID_restaurant. Ця умова допомагає завантажити саме те меню, яке бажає переглянути користувач (рис. 3.5).

```
func fetchMenuData(menuID : String){

    db.collection(menuID).getDocuments{(snap,err) in
        guard let itemData = snap else{return}

        self.items = itemData.documents.compactMap{ (doc) -> Item? in

            let id = doc.documentID
            let name = doc.get("item_name") as! String
            let cost = doc.get("item_cost") as! NSNumber
            let ratings = doc.get("item_ratings") as! String
            let image = doc.get("item_image") as! String
            let details = doc.get("item_details") as! String



            return Item(id: id, item_name: name, item_cost: cost, item_details: details, item_image: image, item_ratings: ratings)

        })
        self.filtered = self.items

    }

}
```

Рисунок 3.5 – Завантаження з бази даних menu

3.2.3 Запис даних до бази даних

В даному застосунку існує лише одна таблиця для запису у базу даних, це таблиця User Order. В цю таблицю передаються дані з місцезнаходженням користувача (зроблено це за для того, щоб користувач міг замовити їжу лише якщо він знаходиться поблизу закладу), страва або декілька страв, які замовив користувач, кількість страв певного типу та загальна вартість замовлення, яку сплатив користувач (рис. 3.6). Оплата у застосунку можлива лише за допомогою Apple Pay, тому дані які передаються у базу даних зроблені виключно для бухгалтерії закладу.

Для запису у базу даних використовується окрема функція, яка спочатку підключається до таблиці Order і заповнює дані про замовлення, потім, підключається до таблиці User Order, передає у останню таблицю дані замовлення та заповнює завдяки функції розрахунку загальної вартості загальну вартість замовлення (рис. 3.7), а також додає у таблицю місцезнаходження клієнта.

```
func updateOrder(){
    //creating dict of food details
    if ordered{
        ordered = false
        db.collection("Users").document(Auth.auth().currentUser!.uid).delete{
            (err) in
            if err != nil{
                self.ordered = true
            }
        }
        return
    }
    var details : [[String : Any]] = []

    cartItems.forEach{ (cart) in

        details.append([
            "item_name" : cart.item.item_name,
            "item_quantity" : cart.quantity,
            "item_cost" : cart.item.item_cost

        ])
    }
    ordered = true
    db.collection("Users").document(Auth.auth().currentUser!.uid).setData([

        "ordered_food" : details,
        "total_cost": calculateTotalPrice(),
        "location" : GeoPoint(latitude: userLocation.coordinate.latitude, longitude: userLocation.coordinate.longitude)

    ]){ err in
        if err != nil{
            self.ordered = false
            return
        }
        print("success")
    }
}
```

Рисунок 3.6 – Функція запису до бази даних

```
func calculateTotalPrice() -> String{

    var price : Float = 0

    cartItems.forEach{(item) in
        price += Float(item.quantity) * Float(truncating: item.item.item_cost)
    }
    return getPrice(value: price)
}
```

Рисунок 3.7 – Функція розрахунку загальної вартості замовлення

3.3 Програмна реалізація

3.3.1 Автентифікація користувача та відстежування локації

Для даного застосунку було прийнято рішення не робити класичну автентифікацію користувача, а зробити її анонімною. Тобто, коли користувач включає застосунок, в нього з’являється вікно з проханням дозволити

відстежувати поточну локацію (рис. 3.8), якщо користувач дає згоду, проводиться його автентифікація.

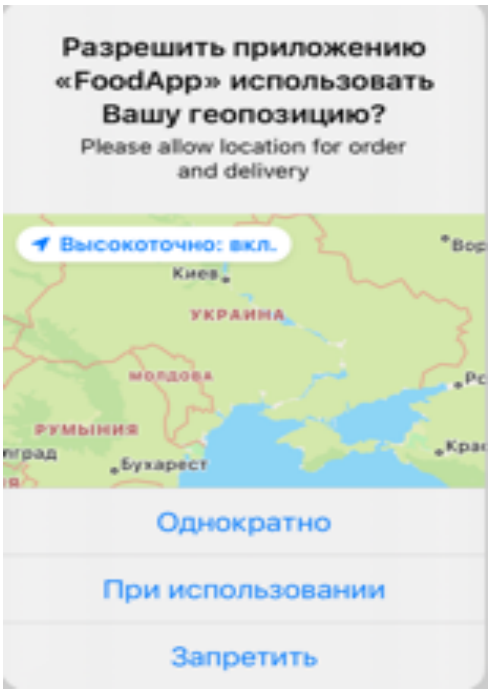

Рисунок 3.8 – Запит на відстежування локації

До бази даних у таблицю Users (табл. 3.3) передається унікально згенерований ID та поточна локація (рис. 3.9), яка буде змінюватися якщо користувач буде пересуватися. Передача локації зроблена лише для того, щоб ідентифікувати присутність користувача у закладі, в якому він хоче зробити замовлення. Це зроблено за для того, щоб не було фішингових атак на базу даних закладу харчування, тобто боти не могли зробити купу замовлень. Так як оплата проходить в мить, коли користувач натискає кнопку «Підтвердити замовлення», риск фішингової атаки ще більше зменшується. Унікальний ID користувача розроблено задля того, щоб не було плутанини з локацією (так як багато користувачів можуть бути в одному закладі одночасно).

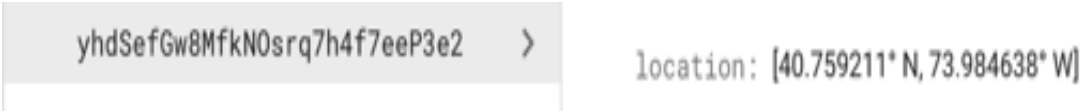

Рисунок 3.9 – Зовнішній вигляд автентифікації у базі даних

У самому закладі можливо буде пронумерувати столи у приміщенні за допомогою координат і коли користувач буде передавати у замовленні свою поточну локацію на момент замовлення, можливо буде зіставити локацію стола і локацію користувача, тим самим вирахувати за яким саме столом буле зроблено замовлення, це суттєво вплине на зручність користування даним застосунком і допоможе рестораторам ще ліпше обслуговувати клієнтів.

Для вирахування локації використовується вбудована в XCode бібліотека Core Location [22]. В бібліотеці Core Location є багато методів для відстежування локація, в цьому проекті використовувалися CL Location Manager та CL Location (рис. 3.10).

```
21      @Published var userLocation : CLLocation!
22      @Published var locationManager = CLLocationManager()
23      @Published var userAddress = ""
24      @Published var noLocation = false
```

Рисунок 3.10 – Об’явлення змінних зв’язаних з вирахуванням локації

Для відстежування локації було розроблено декілька функції, які відповідають за різні дії з локацією. Core Location отримує широту та довготу локації користувача, тому перша функція яка викликається коли користувач запускає додаток, це функція відстежування локації та конвертування їй в звичний для людей формат локації, тобто номер будівлі та вулиця (рис. 3.11), але локація яка передається до бази даних, це саме широта та довгота, зроблено це для зручності відстежування локації при підтвердженні замовлення у самому закладі, та для можливості відстежувати за яким самим столом сидить користувач.

Для реалізації цієї функції потрібно відстежити локацію кожного столу та кожного стільця за цим столом, потім, розробити алгоритм який буде робити припущення, бо клієнт може трошки пере двинути стіл або стілець, чим самим буде змінено локацію. Тобто повинен бути «люфт» у локації.

```
func extractLocation(){
    CLGeocoder().reverseGeocodeLocation(self.userLocation){ (res, err) in
        guard let safeData = res else{return}
        var address = ""

        //getting area and locality name

        address += safeData.first?.name ?? ""
        address += ", "
        address += safeData.first?.locality ?? ""

        self.userAddress = address

    }
}
```

Рисунок 3.11 – Функція конвертування адресу

Також розроблені функції які будуть змінювати локацію у застосунку, якщо користувач буде рухатись, функції, які виводять помилки, якщо щось не так з відстеженням локації, (рис. 3.12), а також критичну помилку, яку не можливо прибрати, вона з'являться якщо користувач не дозволив відстежувати локації, прибрати її можна лише якщо зайти в настройках смартфона до настройок застосунку і там дозволити відстежувати локацію. (рис. 3.13). Ця помилка розроблена спеціально, бо без локації робота застосунку не має сенсу [22–25].

```
46  func locationManagerDidChangeAuthorization(_ manager: CLLocationManager) {
47      switch manager.authorizationStatus{
48      case .authorizedWhenInUse:
49          print("autorized")
50          self.noLocation = false
51          manager.requestLocation()
52      case .denied:
53          print("denied")
54          self.noLocation = true
55      default:
56          print("unknown")
57          self.noLocation = false
58          locationManager.requestWhenInUseAuthorization()
59          //modifying info.plist
60      }
61  }
62   func locationManager(_ manager: CLLocationManager, didFailWithError error: Error){
63      print(error.localizedDescription)
64  }
65
66  func locationManager(_ manager:CLLocationManager, didUpdateLocations locations: [CLLocation]){
67      //reading user locations and extracting details
68      self.userLocation = locations.last
69      self.extractLocation()
70      //after extracting location logging in
71      self.login()
72  }
```

Рисунок 3.12 – Функції зміни локації та помилок при відстеженні локації

```
        //non closable alert if premission denied
        if HomeModel.noLocation {
            Text("Please enable location access in settings to further move on!")
                .foregroundColor(.black)
                .frame(width: UIScreen.main.bounds.width - 100, height: 120)
                .background(Color.white)
                .cornerRadius(10)
                .frame(maxWidth: .infinity, maxHeight: .infinity)
                .background(Color.black.opacity(0.3).ignoresSafeArea())
        }
    }
    .onAppear(perform: {
        //call location delegate
        HomeModel.locationManager.delegate = HomeModel
```

Рисунок 3.13 – Функція критичної помилки при забороні на відстежування локації

3.3.2 Розраховувано відстані до закладу харчування

Для зручності користування застосунком розроблена система вирахування відстані між користувачем та закладом харчування. Таку систему дуже часто використовують в застосунках, зав’язаних на закладах харчування, такі як Glovo чи Rocket. Розрахунок відстані допомога користувачу швидко знайти заклад харчуванню поблизу, наприклад, якщо користувач зголоднів і не знає де поблизу знаходяться заклади харчування, він може просто зайти до застосунку, подивитися які заклади знаходиться поруч, подивитися більш привабливе меню із закладів поруч, та почати рухатися у напрямку цього закладу. Також, цей застосунок може допомогти в розвитку «гастрономічного туризму», який з кожним роком набуває все більшої популярності.

Також, можливо у майбутньому розширити функціонал даного застосунку і адаптувати його під різні міста та країни, щоб користувач, приїжджаючи до іншої країни або міста зміг смачно пообідати, не знаючи при цьому ні одного закладу в місті.

Для розрахування відстані використовується лише одна функція [26-29], яка порівнює широту та довготу між двома локаціями та вертає різницю між цими локаціями у вигляді кілометрів до закладу (рис. 3.14).

```
207
208     func distanceToRest(latitude : Double, longitude: Double) -> String{
209
210         var restLocationLatitude : Double
211         var restLocationLongitude : Double
212         restLocationLatitude = latitude
213         restLocationLongitude = longitude
214         let restLocation = CLLocation(latitude: restLocationLatitude, longitude: restLocationLongitude)
215         let distance = restLocation.distance(from: userLocation) / 1000
216         self.text1 = " \(String(format:"%.02f", distance)) KM "
217         if(distance < 0.01){
218             text1 = "You already here"
219         }
220         return text1
221     }
```

Рисунок 3.14 – Функція розрахунку відстані між локаціями

3.3.3 Apple Pay

Apple pay дуже сучасний та дуже популярний метод розрахунку будь де. У всьому світі майже 700 мільйонів користувачів має Apple Pay. Apple Pay використовується майже в усіх застосунках на базі систем Apple, а також все частіше можна зустріти Apple pay як спосіб оплати на різноманітних сайтах. Apple pay, якщо пояснювати простою мовою, це картка банку в смартфоні, тобто на сьогоднішній день взагалі не потрібно носити гаманець з собою на прогулянку, потрібен лише смартфон, в Україні тим паче, ми маємо документі в смартфоні (Дія) а також маємо змогу оплачувати усе смартфоном (Apple pay, Google pay).

Apple pay був обраний для цього застосунку як єдиний спосіб оплати замовлення. Майже кожен прогресивний Українець користується банківською картою (а саме для прогресивних користувачів розроблений даний

застосунок), а прив’язати карту до Apple pay або Google pay можна у декілька кроків. Apple pay у даному застосунку вирішить декілька проблем сучасного світу. По-перше, через це неможливо буде здійснити фішінгову атаку на сервера закладу, по-друге, цей застосунок зроблений не лише з метою поліпшити якість обслуговування клієнтів, а й з епідеміологічних та екологічних спонукань. Завдяки впровадженню такої системи оплати, клієнт а також офіціант буде менше контактувати з готівковими коштами та з паперовим меню.

Впровадити Apple pay до будь-якого застосунку дуже просто, на офіційному сайті Apple є докладна інструкція як це зробити [30]. Тому, розробити Apple pay у даному застосунку було дуже просто. По-перше, щоб додати Apple pay з тестовою карткою до застосунку, потрібно мати аккаунт розробника Apple, який не є безкоштовним, та коштує 99$ на рік. Після реєстрації акаунта розробника, потрібно вказати свій Apple ID та завантажити для XCode SDK Apple pay. Потім, потрібно налаштувати реквізити тестової картки та куди нараховуються оплачені гроші (рис. 3.15).

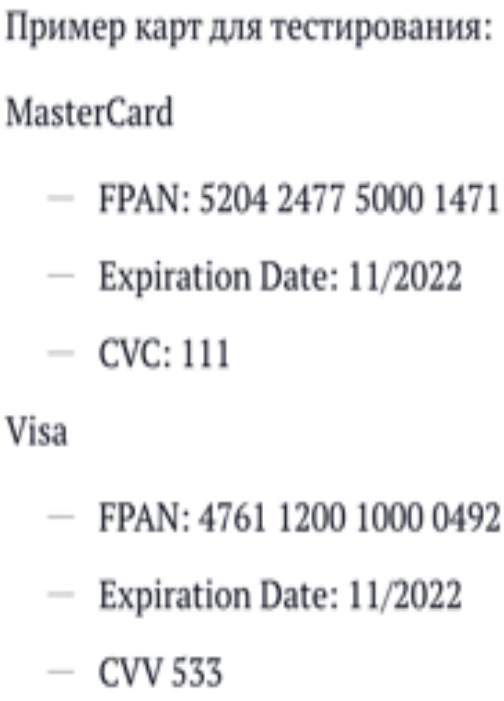

Рисунок 3.15 – Приклад карток для тестування Apple pay

Після налаштування XCode, треба написати трохи Swift коду. Для початку потрібно імпортувати бібліотеку Pass Kit та прописати методи оплати для цього додатку (рис. 3.16), в яких буде прописано, яку валюту оплати приймає Apple pay у даному застосунку, які мережі карток приймає застосунок, розмір шрифт, іконку яка символізує оплату за допомогою сервісу та багато чого іншого.

```
// MARK - Properties
 private var paymentRequest: PKPaymentRequest = {
   let request = PKPaymentRequest()
   request.merchantIdentifier = "merchant.stfalcon.com.applepayexample"
   request.supportedNetworks = [.visa, .masterCard]
   request.supportedCountries = ["UA"]
   request.merchantCapabilities = .capability3DS
   request.countryCode = "UA"
   request.currencyCode = "UAH"
   request.paymentSummaryItems = [PKPaymentSummaryItem(label: "iPhone
Xs 64 Gb", amount: 34999.99)]
   return request
 }()
```

Рисунок 3.16 – Налаштування методів оплати для Apple pay

Далі потрібно створити екземпляр класу PK Payment Request та створити IB Action для кнопки (рис 3.17)

```
// MARK: - @IBActions
@IBAction func purchase(_ sender: Any?) {
    if let controller =
PKPaymentAuthorizationViewController(paymentRequest: paymentRequest) {
        controller.delegate = self
        present(controller, animated: true, completion: nil)
    }
}
```

Рисунок 3.17 – Створення PK Payment Request та IB Action

Відкриваючи PK Payment Authorization View Controller, необхідно підписатися на його delegate, таким чином, буде видно статус транзакції.

З іншого боку, необхідно імплементувати методи делегата. Так як для тестування неможливо провести справжню транзакцію, імітувати успішну транзакцію ми будемо в такий спосіб (рис. 3.18).

```
extension ViewController: PKPaymentAuthorizationViewControllerDelegate {

    func paymentAuthorizationViewController(_ controller:
PKPaymentAuthorizationViewController, didAuthorizePayment payment:
PKPayment, handler completion: @escaping (PKPaymentAuthorizationResult)
-> Void) {
        completion(PKPaymentAuthorizationResult(status: .success,
errors: nil))
    }

    func paymentAuthorizationViewControllerDidFinish(_ controller:
PKPaymentAuthorizationViewController) {
        controller.dismiss(animated: true, completion: nil)
    }

}
```

Рисунок 3.18 – Імітування успішної транзакції

Після всіх цих дій Apple pay успішно налаштований для даного застосунку, тобто, коли користувач вибере з меню страви, система автоматично вирахує сумарну вартість усіх страв, та після того як користувач підтвердить свій вибір, в нього «вилізе» вікно з запитом Apple pay на загальну суму замовлення (рис. 3.19), після успішної оплати замовлення передається ресторану для подальшою роботою з ним.

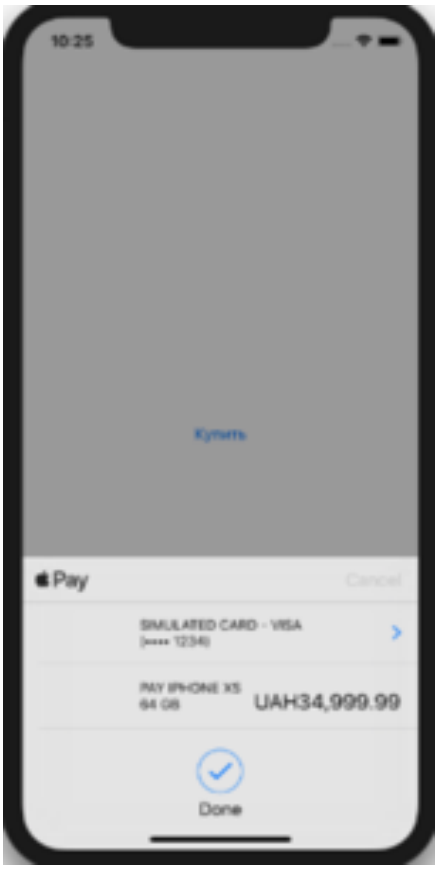

Рисунок 3.19 – Успішна оплата завдяки Apple pay

3.3.4 QR-сканер

QR-код у даному застосунку допомагає швидко знайти потрібне меню, коли клієнт вже прийшов до закладу харчування. Тобто ресторану не потрібно більше друкувати кожен раз купу меню, а потрібно 1 раз надрукувати потрібний qr-код з посиланням на меню у даному застосунку, і приклеїти цю наліпку з qr-кодом до стола. Так, користувачу не потрібно нікого чекати, доки хтось винесе меню, прийме замовлення, а потрібно просто прийти до закладу, дістати смартфон, включити застосунок і відскакувати qr-код, подивитися меню і зробити замовлення.

Так як компанія Apple активно використовує qr-коди, та навіть вбудувала до додатка «Камера» сканер, для реалізації коду сканера не потрібно завантажувати ніяких бібліотек, вона вже вбудована в XCode, це

бібліотека AV Foundation [31, 32]. Завдяки цієї бібліотеки розроблено декілька функцій, які запускають окреме вікно з камери смартфона, де запускається камера та сканується qr-код. Також у цих функціях обрано формати, які буде зчитувати камера та дії, які буде робити застосунок після того, які qr-код буде від сканований, в даному випадку, це перехід до меню закладу харчування, якому цей код належить.

3.4 Інструкція користувача

У наступних розділах буде розглянута взаємодія користувача з програмним додатком, починаючи від звичайної інсталяції. Також буде детально описано роботу основних модулів програмної системи.

3.4.1 Інсталяція та системні вимоги

Компанія «Apple» дуже старанно ставиться до безпеки своїх користувачів, через це завантажувати та встановлювати додатки, які не знаходяться в магазині «App Store», у користувача – не вийде. Всі програми, які потрапили до маркету, обов'язково пройшли кілька етапів детальної перевірки.

Цей додаток для iOS поки що не додавався в маркет, головною причиною є що за цю функцію потрібно заплатити, але в майбутньому це планується здійснити. Тому тільки після цього він буде доступний для кінцевих користувачів.

Для того щоб користувач міг завантажити додаток, йому потрібно мати iPhone або iPad з версією iOS не нижче 10.0, та бути зареєстрованим у «App Store», також потрібно мати мінімум 100 мегабайтів вільної пам'яті на смартфоні. Якщо попередні вимоги виконуються, то потрібно лише в пошуках

від маркету знайти додаток під назвою «Food app» та натиснути на кнопку «Встановити». Після встановлення на головному екрані смартфона з`явиться іконка Food App (рис. 3.20).

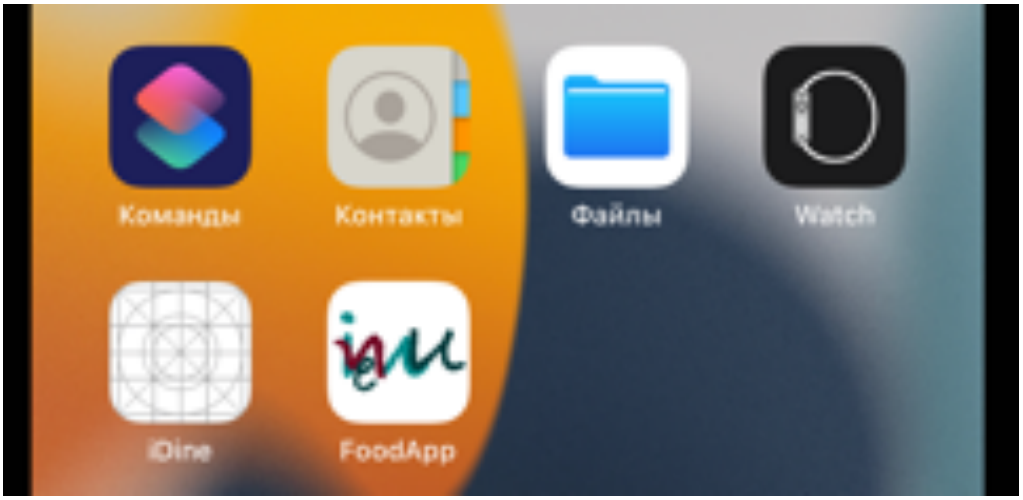

Рисунок 3.20 – Іконка на головному екрані смартфона

3.4.2 Стартовий екран

Після того, як користувач завантажить та відкриє даний програмний продукт, він відразу побачить повідомлення про запитування на відстежування локації (рис. 3.21), якщо користувач не дає згоду на обробку локації, він побачить не закриваємо вікно, з проханням дозволити у налаштуваннях відстежувати локацію, бо без локації робота застосунку неможлива. Додатки для платформи iOS мають властивість під час запуску завантажувати деякі з файлів проекту в оперативну пам'ять і відповідна операція займає певний час. Чим новіше у користувача iPhone, тим швидше буде завантажуватися застосунок. За попередніми даними у наступній версії операційної системи планується збільшити швидкість запуску додатку на 50%.

Саме цей факт обумовлює високу швидкодію програмних застосунків для даної операційної системи.

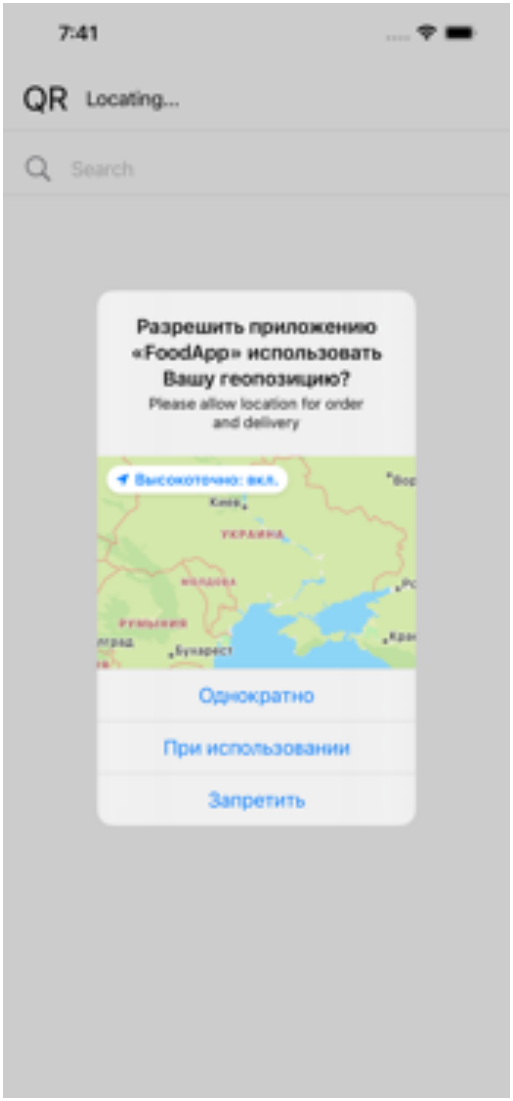

Рисунок 3.21 – Запит на відстежування програмою локації

3.4.3 Головна сторінка

Після того як користувач дозволить відстежувати свою локацію, йому одразу буде дозволено користування застосунком, так як за логікою застосунку автентифікація проходить анонімно та непомітно для користувача, його буде перенаправленню на головну сторінку застосунку (рис. 3.22). Головна сторінка оформлена виключно з використанням SwiftUI.

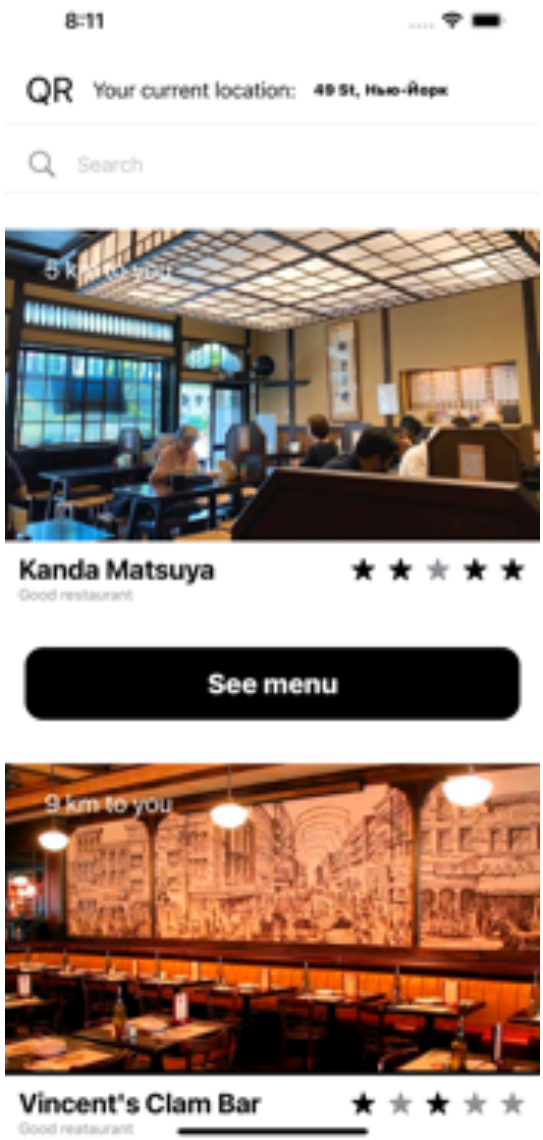

Рисунок 3.22 – Головна сторінка додатку Food App

На головній сторінці додатку користувач одразу бачить список запропонованих закладів харчування. Зверху, користувач бачить його поточну локацію, якщо він буде рухатись, локація також буде змінюватися. Поряд із локацією, користувач бачить кнопку QR, натиснув на яку у користувача ввімкнеться камера, наводячи яку на QR-код, користувача перенаправить на відповідне меню.

Нижче локації та кнопки QR, користувач бачить поле Search, вписав в яке назву або навіть першу букву закладу, йому висвітіться результат пошуку нижче, тобто пошук знайде усі заклади харчування, назви яких розпочинаються на вписану букву (рис. 3.23).

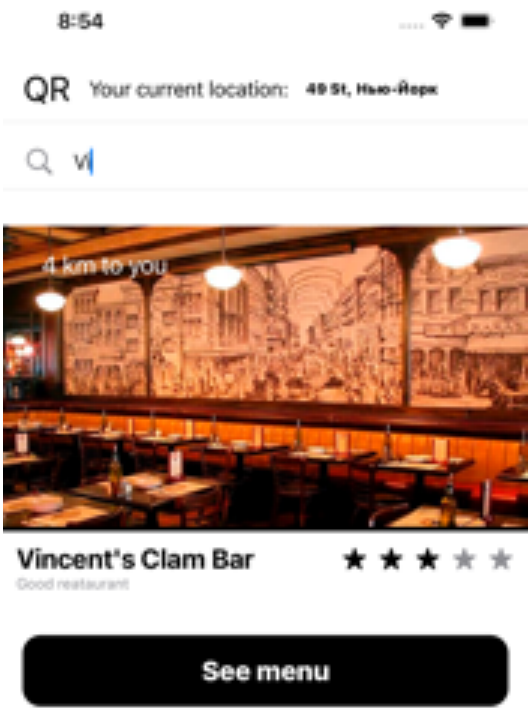

Рисунок 3.23 – Результат пошуку через строку Search

Ще нижче, користувач бачить основний блок ресторанів, в якому у кожного ресторану є своя картка (рис. 3.24). На цій картці користувач по перше бачить фотографію цього закладу, у верхньому лівому куті відстань від його локації до цього закладу харчування, нижче - назва ресторану, його опис, та з права зірочки з рейтингом ресторану, вираженим у зірках, темні зірки – рейтинг, максимальний рейтинг – п’ять зірок. У самому низі картки ресторану є одна кнопка – «See menu», натиснув на яку користувач відкриє меню цього закладу харчування.

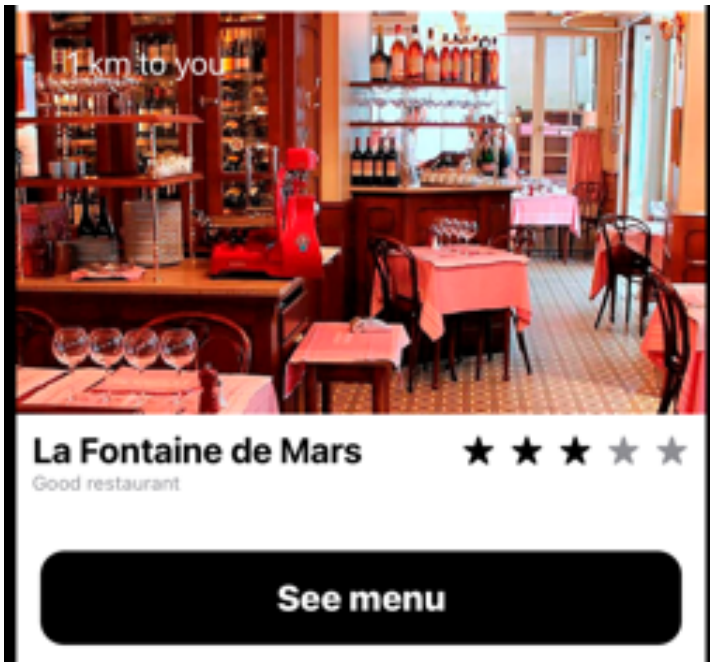

Рисунок 3.24 – Картка закладу харчування

3.4.4 Сторінка з меню

Після натискання на кнопку «See menu», клієнт потрапляє на наступну сторінку – сторінку з меню обраного закладу харчування. Стилізовано, вона виглядить майже так само, як і головна сторінка, але все ж таки є деяка різниця. Перша різниця у верхньому навігаційному меню, тут замість кнопки QR з’являться дві інші кнопки, це кнопка назад та кнопка «бургер» меню. Також, замість локації користувача зверху мається два тексти написані різним шрифтом, перший – це назва ресторану, другий – повна адреса ресторану (рис. 3.25).

Також трохи змінено функціонал картки з стравою (картка виглядає майже так само як і картка ресторану). Замість відстані до закладу (рис. 3.22), на цій картці відображається ціна страви та навпроти ціни маємо кнопку «+», натиснувши на яку страва додається до замовлення, а замість «+» з’являються «√» (рис. 3.26), що означає що страва додана до замовлення.

На цій сторінці так само працює пошук за назвою, як і на сторінці з закладами харчування.

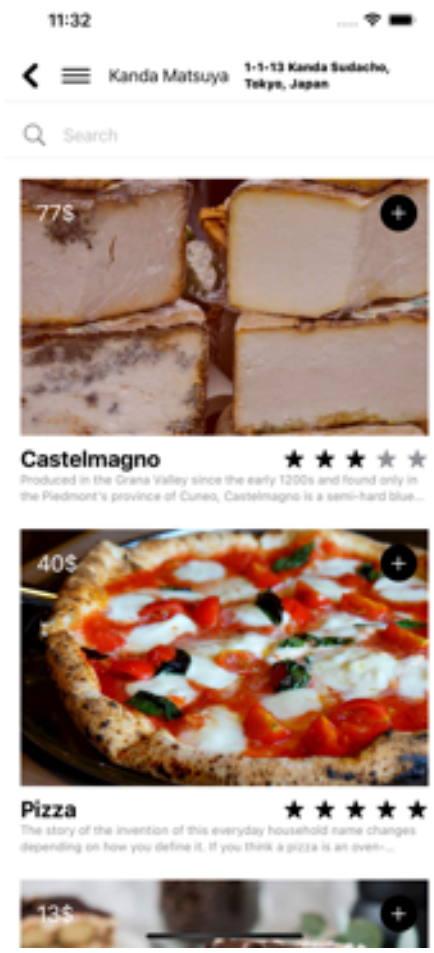

Рисунок 3.25 – Сторінка меню

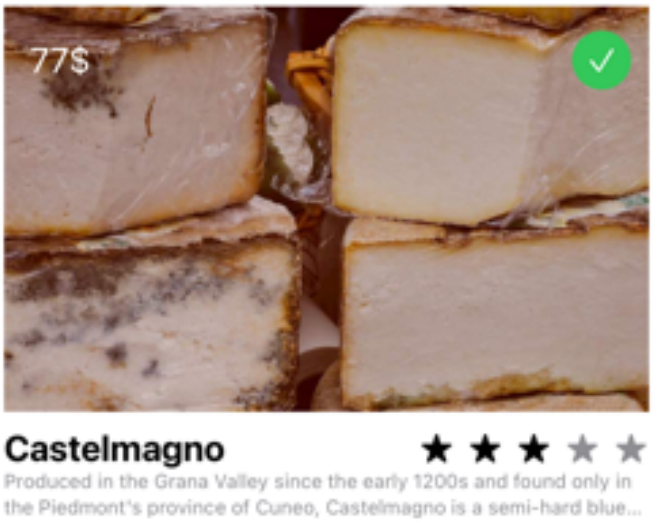

Рисунок 3.26 – Картка доданої страви до замовлення

При натисканні на «бургер» меню з’являться ще одне меню, в якому є кнопка «Cart», натиснувши на яку користувач перейде до свого замовлення та розрахунку (рис. 3.27). Але ця кнопка доступна не завжди, якщо користувач знаходиться не поруч з закладом харчування, ця кнопка буде заблокована.

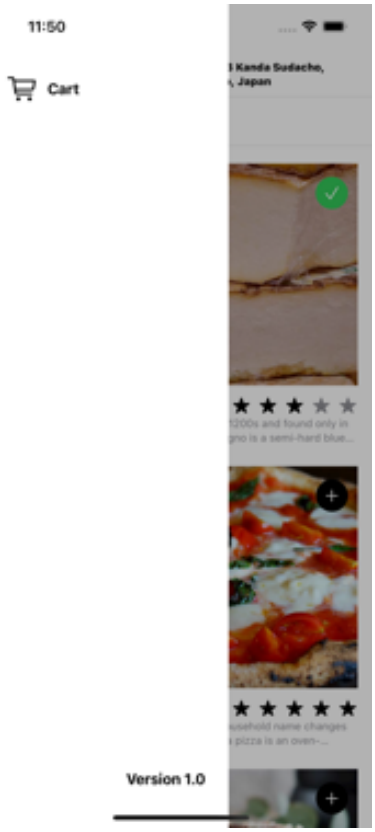

Рисунок 3.27 – «Бургер» меню

3.4.5 Сторінка замовлення

Остання сторінка в цьому застосунку це сторінка замовлення. В верхньому навігаційному меню є лише кнопка «назад» у вигляді стрілочки а також напис «My cart». Нижче в цьому меню відображанні усі страви, які користувач хоче замовити в даному закладі харчування (рис. 3.28). Якщо користувач замовив страву, на цій сторінці з’являється картка, на якій є зображення, опис, вартість та кількість цієї страви у замовленні. Користувач може редагувати кількість однакових страв або видаляти страви. Щоб

збільшити або зменшити кількість страв, потрібно натиснути на «+» або «-» навпроти зображення на картці страви, відображення кількості буде змінюватися і також буде змінюватися загальна вартість замовлення. У самому низу сторінки є кнопка «Check out» натиснути на яку потрібно лише після перевірки свого замовлення. Натиснувши кнопку, з’явиться вікно з Apple Pay на загальну суму замовлення, оплативши яке, клієнту залишиться лише чекати, доки його замовлення приготують та по дадуть його до клієнта.

Рисунок 3.28 – Сторінка з замовленням

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований мобільний додаток для платформ на операційній системі iOS «Food App», який виконує функцію електронного меню для закладів харчування. Програма розроблена на мові програмування Swift, яка призначена для девайсів компанії Apple, у середовищі розробки XCode.

Було виконано дослідження наявних аналогічних або близьких за змістом програми для систематизації проведення лекцій, проаналізовані їх недоліки та запропоновані інтерфейсі та функціональні рішення для покращення зручності користування користувачів.

Розроблений додаток надає змоги:

– легко та зручно переглядати заклади харчування поблизу;

– переглядати запропоноване меню у цих закладах харчування;

– швидкий пошук за назвою страви або закладу харчування;

– швидке сканування QR-коду, щоб моментально перейти до меню саме цього закладу;

– легко робити замовлення у ресторані, не чекаючи на офіціанта;

– швидкий, а головне – сучасний спосіб оплати, завдяки Apple pay;

– користувачу більше не потрібно контактувати з паперовим меню, та з чеками і терміналами для оплати, що збільшує індивідуальний захист від вірусів.

Також в програмі наявне анонімна автентифікація, відстежування локації користувача, завантаження даних з бази та передача даних до бази даних.

За допомогою архітектурних шаблонів було створено легко розширювану систему, за допомогою якої в майбутньому можна буде повністю автоматизувати та покращити процес обслуговування клієнтів [32].

Також було розроблено гнучкий та адаптивний інтерфейс для даної програмної системи. Цей факт забезпечує можливість використовувати застосування на мобільних телефонах iPhone та планшетів iPad з відповідними операційними системами – iOS та iPad Os [32].

Було використано різноманітні засоби серед розробки XCode та конструктора інтерфейсу за допомогою яких розроблено зрозумілий інтерфейс для використання користувачами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Neuburg, M. (2016). IOS 10 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics. " O'Reilly Media, Inc.".

2. Усов, В. А. (2015). Swift. Основы разработки приложений под iOS "Издательский дом"" Питер"""".

3. Марк, Д., Олссон, Ф., & Топли, К. (2015). Swift. Разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK. М.: Вильямс.

4. Fung, B., & Lind III, C. (2015). CP Maps–An interactive Cal Poly Map iOS Application.

5. Roadley, T. (2013). Learning Core Data for IOS: A Hands-On Guide to Building Core Data Applications. Addison-Wesley.

6. Goodwill, J., & Matlock, W. (2015). The swift programming language. In Beginning Swift Games Development for iOS (pp. 219-244). Apress, Berkeley, CA.

7. Sadun, E. (2013). IOS Drawing: Practical UIKit Solutions. Addison-Wesley.

8. Руководство Swift. Swiftbook. URL: https://swiftbook.ru/ (дата звернення 18.04.2022).

9. Мартин, Р. (2018). Чистая архитектура. Искусство разработки программного обеспечения. " Издательский дом"" Питер"".

10. Buckley, T. F., & Jesty, P. H. (1989, June). Programming a VIPER. In Proceedings of the Fourth Annual Conference on Computer Assurance,'Systems Integrity, Software Safety and Process Security (pp. 84-92). IEEE.

11. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

12. Бондаржевская, К. А., & Каблаш, К. Д. (2019). Принципы flat дизайна.

13. Гамма, Э., Хелм, Р., Джонсон, Р., & Влиссидес, Д. (2007). Приемы объектно-ориентированного проектирования. Паттерны проектирования. Учебное пособие.

14. Fowler, M. (2012). Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch. Addison-Wesley.

15. Weisfeld, M. (2013). The Object-Oriented Thought Process: ObjectOr Thought Process_4. Addison-Wesley.

16. Chupikov, A., Kinoshenko, D., Mashtalir, V., & Shcherbinin, K. (2006, July). Image retrieval with segmentation-based query. In International Workshop on Adaptive Multimedia Retrieval (pp. 207-221). Springer, Berlin, Heidelberg.

17. Kussul, N., Skakun, S., Kravchenko, O., Shelestov, A., Gallego, J. F., Kussul, O., ... & Sokolov, G. Архитектурно-структурные особенности средств автоматизации процесса извлечения знаний из естественно-языковых текстов.

18. Kinoshenko, D., Mashtalir, S., Stephan, A., & Vinarski, V. (1993). Neural Network Segmentation Of Video Via Time Series Analysis. INFORMATION THEORIES & APPLICATIONS, 232.

19. Егорова, Е. А., Киношенко, Д. К., Машталир, С. В., & Шляхов, Д. В. (2006). Метрическое сравнение результатов сементации изображений.

20. Mashtalir, V., Shcherbinin, K., Shlyakhov, V., & Yegorova, E. (2011). REGIONS OF SUFFICIENCY FOR METRICAL DATA RETRIEVAL. INFORMATION TECHNOLOGIES & KNOWLEDGE, 31.

21. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

22. Bodyanskiy, Y., Kinoshenko, D., Mashtalir, S., & Mikhnova, O. (2012). On-line video segmentation using methods of fault detection in multidimensional time sequences. International Journal of Electronic Commerce Studies, 3(1), 1-20.

23. Brytik, V., Grebinnik, O., & Kobziev, V. (2016). Research the possibilities of different filters and their application to image recognition problems. ECONTECHMOD: An International Quarterly Journal on Economics of Technology and Modelling Processes, 5(4), 21-27.

24. Андриенко, М. П. (2019). ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ В SWIFT. In *НАУЧНЫЕ ДОСТИЖЕНИЯ И ОТКРЫТИЯ СОВРЕМЕННОЙ МОЛОДЁЖИ* (pp. 97-99).

25. Войтова, Н. А., & Кулев, Е. Г. (2019). UX/UI: ДИЗАЙН ИНТЕРФЕЙСОВ. Вестник образовательного консорциума Среднерусский университет. Информационные технологии, (1), 4-5.

26. Khast, P. (2017). UX/UI Design Process for a Peer to Peer Financial Platform.

27. Podolny, J. M., & Hansen, M. T. (2020). How Apple is organized for innovation. Harvard Business Review, 98(6), 86-95.

28. Horák, J., & Kaisler, D. (2022). Evaluation of development of Apple Inc. stock price time series. In SHS Web of Conferences (Vol. 132, p. 01012). EDP Sciences.

29. Arocha, J. B. (2017). Getting to the Core: A Case Study on the Company Culture of Apple Inc.

30. Сафин, А. М., & Кадыров, К. А. (2021). АРХИТЕКТУРА MVC. *WORLD SCIENCE: PROBLEMS AND INNOVATIONS 3*, 53.

31. Комлев, Н. (2020). Объектно-ориентированное программирование. Litres.

32. Apple developer. Apple. URL: https://developer.apple.com/ (дата звернення 18.04.2022).