НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

___першого (бакалаврського)_____**рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
_____7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: ____Розробка мобільних застосувань для операційної системи iOS
за допомогою мови програмування Swift_____
_____

Виконав (-ла): студент (-ка) __4__ курсу, групи ___ДА-21_____
(шифр групи)

_____Петрішенко Сергій Олександрович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник_____доцент, к.т.н. Безносик О. Ю. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Економічна частина професор, д.е.н. Семенченко Н.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____доцент, к.т.н. Тимощук О. Л. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А._____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Петрішенку Сергію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка мобільних застосувань для операційної системи iOS за допомогою мови програмування Swift
керівник проекту (роботи) Безносик Олександр Юрійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 15.06.2016

3. Вихідні дані до проекту (роботи) _____
Приклади у вигляді програмних продуктів, які:
- наглядно демонструють різні технології розробки для платформи iOS;
- основні особливості мови програмування Swift.

Використання фреймворків iOS SDK (Core Data, MapKit, CloudKit).
Форма реалізації – у вигляді мобільних застосувань.
Середовище розробки – Xcode, мова програмування – Swift.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)
1. Обгрунтувати вибір мови програмування та програмного середовища розробки.

2. Дослідити основні методи та особливості технологій розробки мобільних застосувань для платформи iOS.
3. Розглянути основні фреймворки, використовуючи iOS SDK.
4. Зробити аналіз розроблених продуктів.
5. Зробити оцінку використаних та досліджених підходів.
6. Визначити переваги та недоліки створеного рішення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Архітектура платформи iOS – плакат.
2. Порівняння продуктивності – плакат.
3. Діаграми класів прикладів – плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічна частина	Семенченко Н. В., професор		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Ознайомлення з технічною літературою	15.02.2016	
3	Збір інформації	28.02.2016	
4	Дослідження предметної області та існуючих рішень	10.03.2016	
5	Обгрунтування вибору мови програмування	15.03.2016	
6	Дослідження існуючих технологій	25.03.2016	
7	Виконання програмної реалізації прикладів	25.04.2016	
8	Розробка мобільних застосувань з використанням вибраної технології	30.04.2016	
9	Оформлення дипломної роботи	31.05.2016	
10	Отримання допуску до захисту та подача роботи в ДЕК	15.06.2016	

Студент _____ С. О. Петрішенко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ О. Ю. Безносик
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Петрішенка Сергія Олександровича
на тему «Розробка мобільних застосувань для операційної системи iOS використовуючи мову програмування Swift»

Дана дипломна робота присвячена розробці програмного забезпечення для платформи iOS використовуючи мову Swift. Метою роботи є демонстрація головних особливостей мови програмування Swift на основі прикладів мобільних додатків.

В роботі розглянуто основні засоби для розробки мобільного програмного забезпечення, проведено їх аналіз та визначено найкращий засіб для реалізації поставленої задачі. На основі аналізу існуючих фреймворків було проведено огляд основних технологій розробки додатків під платформу iOS. Розроблені мобільні додатки на основі проведених досліджень як приклад використання Swift. Зроблено оцінку використаних та досліджених підходів. Визначено основні переваги та недоліки створеного рішення.

Загальний обсяг роботи: 98 сторінок, 34 рисунків, 10 таблиці, 19 бібліографічних найменувань.

Ключові слова: фреймворк, мобільна платформа iOS, технологія розробки, архітектура платформи, мобільний додаток, інструменти для розробки.

АННОТАЦИЯ

бакалаврской дипломной работы Петришенка Сергея Александровича
на тему «Разработка мобильных приложений для операционной системы iOS используя язык программирования Swift»

Данная дипломная работа посвящена разработке программного обеспечения для платформы iOS используя язык Swift. Целью работы является демонстрация главных особенностей языка программирования Swift на основе примеров мобильных приложений.

В работе рассмотрены основные средства для разработки мобильного программного обеспечения, проведено их анализ и определено лучшее средство для реализации поставленной задачи. На основе анализа существующих фреймворков был проведен осмотр основных технологий разработки приложений под платформу iOS. Разработаны мобильные приложения на основе проведенных исследований как пример использования Swift. Произведена оценка использованных и исследованных подходов. Определены основные преимущества и недостатки созданного решения.

Общий объем работы: 98 страниц, 34 рисунков, 10 таблицы, 19 библиографических наименований.

Ключевые слова: фреймворк, мобильная платформа iOS, технология разработки, архитектура платформы, мобильное приложение, инструменты для разработки.

ANNOTATION

a bachelor’s degree work of Serhii Petrishenko
entitled "Mobile application development using Swift”

This thesis is devoted to the development of software for the iOS platform using Swift language. The aim is to demonstrate the main features of Swift programming language based on the examples of mobile applications.

The paper describes the main tools for developing mobile software, conduct analysis and determine the best way to accomplish the task. Based on an analysis of existing frameworks was carried out inspection of the main application development technologies for iOS platform. Developed mobile applications on the basis of the research as an example of using Swift. The estimation used and tested approaches. The main advantages and disadvantages created solutions.

Total volume of work: 98 pages, 34 figures, 10 tables, 19 bibliographical references.

Keywords: framework, iOS mobile platform, development technology, platform architecture, mobile application, development tools.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	програмне забезпечення
UI (User Interface)	інтерфейс користувача
Фреймворк	(англ. Framework, каркас, платформа, структура, інфраструктура) – інфраструктура програмних рішень, що полегшує розробку складних систем
IDE (Integrated Development Environment)	комп'ютерна програма, що допомагає програмістові розробляти нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче
GCC (GNU Compiler Collection)	набір компіляторів для різних мов програмування
LLVM (Low Level Virtual Machine)	універсальна система аналізу, трансформації і оптимізації програм, щореалізує віртуальну машину з RISC-подібними інструкціями
Interface Builder	середовище для розробки інтерфейса корситувача
Developer Tools	інструменти для розробки
Юніт-тестування	модульне тестування
RPC (Remote Procedure Call)	протокол, що дозволяє програмі, запущеній на одному комп'ютері,

	бути викликаною на іншому комп'ютері без написання безпосередньо коду для цієї операці
API (Application Programming Interface)	набір визначень взаємодії різнотипного програмного забезпечення
HTTP (HyperText Transfer Protocol)	протокол передачі даних, що використовується в комп'ютерних мережах
SDK (Software Development Kit)	набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи (програмної або програмно-апаратної)
WWDC (Apple Worldwide Developers Conference)	щорічна конференція розробників для платформи Macintosh
Система контролю версій	програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D моделі, текстового документу тощо

ВСТУП

Оцінка сучасного стану проблем

Розробка мобільних додатків відіграє все більш важливу роль для організацій, яким необхідно спілкуватися зі співробітниками або клієнтами за допомогою вбудованих додатків. На сьогоднішній день існує великий вибір мов програмування для розробки мобільних додатків. Це пов'язано з тим, що для різних мобільних пристроїв доводиться використовувати різні мови програмування, що обумовлене тим, що мобільні пристрої мають різні операційні системи (ОС).

Цільова платформа (iOS, Android, Windows Phone) буде мати значний вплив на мову розробки, яка буде використовуватися. Наприклад, можна розробляти рідні додатки для кожної платформи або використовувати сторонній інструмент для оптимізації своїх додатків на різних платформах. Другий підхід може заощадити час і зусилля, хоча це може вплинути на зручність використання. Сучасні мобільні пристрої пропонують широкий спектр варіантів розробки.

Процес розробки програмного забезпечення для мобільних пристроїв вимагає підтримувати певні обмеження щодо специфіки роботи додатків у мобільних операційних системах. Особливості роботи мобільних додатків стосуються перш за все:

- на витрати живлення акумуляторної батареї мобільного пристрою;
- обмеження на кількість даних, що передаються через мережу Інтернет;
- зменшену швидкість передачі пакетів в мобільному Інтернет з'єднанні;
- можливість втрати пакетів при передачі в мобільному Інтернет з'єднанні;
- кількість обмінів даними з зовнішніми пристроями на Bluetooth;
- безпеку даних користувача;
- значну фрагментацію версій операційних систем та фреймворків;
- велику різноманітність розмірів екранів мобільних пристроїв;

- обмеження запитів до системи визначення місцезнаходження абонента;
- обмеження на розмір файлу додатку;
- порівняно невеликий ліміт оперативної пам'яті мобільного пристрою;
- порівняно обмежену швидкість передачі даних в мобільному Інтернет з'єднанні.

Актуальність

Сучасна людина робить все для того щоб досягти максимального комфорту. Сьогодні одним з бажань більшості людей є вихід в Інтернет. Причому вони завжди хочуть залишатися онлайн. Саме тому величезною актуальністю користується така послуга, як розробка мобільних додатків під ОС iOS. Все це стало актуальним разом з появою мобільного Інтернету. Під час поїздок завжди є можливість підключитися до мережі за допомогою телефону, планшета або іншого пристрою. Але відразу ж варто відзначити, що без спеціальних додатків навряд чи б була досягнута необхідна ефективність. Без них не обійтися і при вирішенні таких завдань, як архітектурна 3D візуалізація.

Сьогодні фахівцями в області інформаційних технологій розробляються мобільні додатки, які дозволяють вирішувати безліч завдань, наприклад, створення 3D-анімації. Деякі служать для того щоб встановлювати з'єднання з мережею. Інші допомагають оптимізувати маршрут. Треті призначені для тих, хто шукає найвигідніші магазини. Є й такі, за допомогою яких можна замовити їжу додому. В основу кожної з таких програм лягли певні утиліти, що в результаті дозволяє швидко вирішувати поставлену задачу, економити час і досягати максимально комфортного рівня життя.

Всі мобільні додатки умовно можна поділити на програми для робочих цілей і на розважальні програми. Перші дозволяють бізнесменам і офісним працівникам контролювати бізнес-процеси, складати аналітичну звітність, виконувати такі завдання, як розробка дизайну фірмового стилю. Другі включають в себе різноманітні ігри, програмне забезпечення для перегляду фільмів і прослуховування музики, засоби для спілкування тощо. Кожен з

мобільних застосувань знаходить свого споживача, однак, як зазначають фахівці з цій галузі, найбільшою популярністю користується спеціалізоване програмне забезпечення, наприклад, розробка фірмового стилю, який необхідний компаніям, що працюють в різних напрямках. Також саме на таких програмах можна робити непогані гроші, адже сучасні компанії не шкодують інвестицій в продукти, які могли б в будь-якій мірі оптимізувати або спростити наявні бізнес-процеси.

Протягом останніх років показник, що характеризує рівень попиту на мобільні пристрої, постійно зростає. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна і доцільна. Отже, тільки корисна розробка отримає гідне визнання з боку користувачів.

Мета

Основною метою дипломної роботи є розробка мобільних застосувань для операційної системи iOS за допомогою мови програмування Swift. Відповідно до цього на шляху досягнення мети будуть вирішені такі задачі:

- обгрунтування вибору мови програмування та середовища розробки;
- дослідження існуючих технологій та рішень для створення мобільних застосувань для ОС iOS;
- обгрунтування вибору певної технології, використовуючи iOS SDK;
- аналіз розробленого програмного продукту;
- оцінка використаних та досліджених підходів;
- визначення переваг та недоліків створеного рішення;

Необхідність проведення розробки

Головною ціллю дипломної роботи є використання для розробки мобільних додатків нової мови програмування – Swift. Адже компанія Apple постаралася створити просту і сучасну мову, на яку вони покладають великі надії. Купертіновці планують залучити на свою сторону не тільки досвідчених розробників, які вже вміють працювати з Objective-C і C, а й новачків, які

напевно використовують дану їм можливість і почнуть вивчати Swift. Ринок додатків знову виросте, а значить вплив Apple збільшиться.

1. ЗАСОБИ РОЗРОБКИ

1.1 Огляд середовищ розробки

1.1.1 Середовище розробки Xcode

Xcode – це пакет інструментів для розробки додатків під Mac OS X і iPhone OS, розроблений Apple. Остання версія Xcode 3.2, безкоштовно поставляється на дистрибутивному диску Mac OS X Install DVD разом з операційною системою Mac OS X 10.6, хоча і не встановлюється за умовчанням. Третя версія не підтримується старими версіями Mac OS, для яких XCode також доступний для безкоштовного звантаження через Apple Developer Connection. Оновлення можна безкоштовно скачати на офіційному сайті підтримки. На сьогодні є єдиним засобом написання «універсальних» (Universal Binary) прикладних програм для Mac OS X.

Xcode тісно інтегрований з фреймворком Cocoa. Його використовують і при розробці самої Apple Mac OS X. Цей набір інструментів включає:

- Xcode IDE (для кодування, створення і налагодження додатків) (рис. 1.1) [1]:

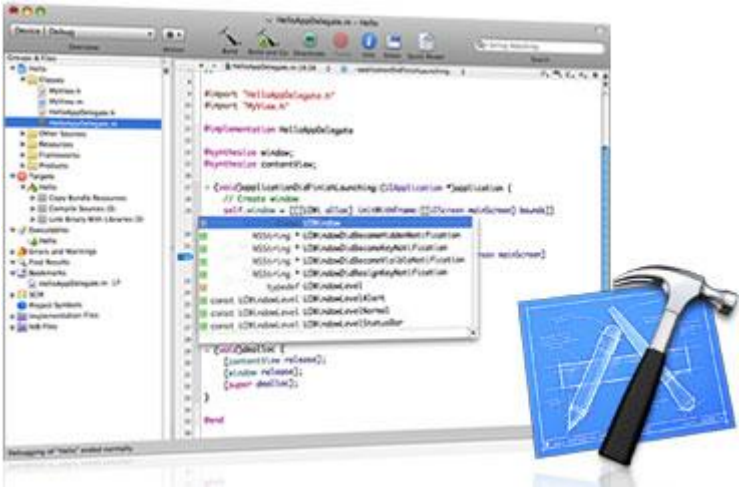

Рисунок 1.1 – Xcode IDE

Xcode IDE надає все, що потрібно для розробки: від професійних редакторів з функцією автозаповнення коду і Cocoa рефакторінга, до налаштування open-source компіляторів. Xcode IDE розроблений з нуля, щоб можна було скористатися всіма можливостями Cocoa і новітніми технологіями Apple.

Xcode включає велику кількість можливостей, щоб полегшити розробки iOS-додатків, включаючи наступні:

- систему управління проектом для визначення програмних продуктів;
- середовище для редагування коду, що включає можливості підсвічування синтаксису, автозаповнення коду та індексацію символів;
- просунуту програму перегляду і пошуку документації Apple;
- контекстно-залежний інспектор для перегляду інформації про виділений в коді символі;
- просунуту систему збирання проекту з перевіркою залежностей і перевіркою правил складання;
- компілятори GCC, що підтримують мови C, C ++, Objective-C, Objective-C ++ та інші;
- підтримка LLVM і Clang для мов C, C ++ і Objective-C;
- вбудоване налагодження вихідного коду з використанням GDB;
- розподілені обчислення, що дозволяють поширювати великі проекти через кілька мережевих пристроїв;
- інтелектуальна компіляція, яка прискорює час компіляції одного файлу;
- просунуті можливості по налагодженню коду;
- просунуті засоби налагодження, що дозволяють робити глобальні зміни в коді без зміни його поведінки;

- підтримка знімків проекту, які надають легше управління вихідним кодом;
- підтримка запуску засобів продуктивності для аналізу програми;
- підтримка вбудованої системи управління вихідним кодом;
- підтримка AppleScript для автоматизації процесу складання;
- підтримка налагоджувальної інформації в форматах DWARF і Stabs (налагоджувальна інформація для всіх проектів за замовчуванням генерується в форматі DWARF).

- Interface Builder (для розробки користувальницького інтерфейсу) (рис. 1.2) [1]:

Рисунок 1.2 – Interface Builder

Interface Builder спрощує створення призначеного для користувача інтерфейсу (UI). З його допомогою можна легко, без написання коду, створити шари з вікон, різні кнопки, повзунки та інші елементи управління. Потім можна перетворити цей прототип UI в реальний додаток, додавши нові можливості. Xcode працює з Interface Builder в режимі реального часу, так що можна спостерігати в графічному інтерфейсі (Interface Builder) те, що розробляється в Xcode.

Використовуючи Interface Builder, можна створювати вікна програми шляхом перетягування в нього попередньо налаштованих компонентів. Компоненти включають стандартні системні елементи управління, такі як перемикачі, текстові поля, кнопки тощо. Компоненти поміщаються вікна, після цього їх можна позиціонувати, перетягуючи по всій величині вікна, налаштовувати атрибути, використовуючи інспектор і встановлювати зв'язки між цими об'єктами і кодом. Вміст вікна зберігається в nib-файл, який є ресурсним файлом особливого формату. Він містить всю інформацію, яка потрібна бібліотеці "UIKit" для створення тих же самих об'єктів у додатку під час виконання. Завантаження nib-файлу створює версії часу виконання всіх об'єктів, збережених у файлі, налаштовує їх в точності, якими вони були в Interface Builder. Також використовується інформація про взаємозв'язок, зазначена розробником для установки зв'язку між заново створеними об'єктами і вже існуючими в додатку. Ці зв'язки забезпечують код вказівниками на об'єкти nib-файлу, а також надають інформацію самим об'єктам, необхідну для передачі дій користувача вихідному коду.

В цілому, використання Interface Builder зберігає величезну кількість часу, що припадає на створення UI. Interface Builder усуває написання коду, необхідного для створення, налаштування і позиціонування об'єктів, які використовуються для розробки інтерфейсу. Тому тут можна побачити, як інтерфейс буде виглядати під час виконання. UI фактично є архівами об'єктів Cocoa, які не вимагають генерації коду. Зміни в інтерфейсі користувача (UI) не вимагають перекомпіляції (перевірки) коду, а зміни в коді не вимагають перекомпіляції UI.

- Інструменти для аналізу поведінки і продуктивності (рис. 1.3) [1]:

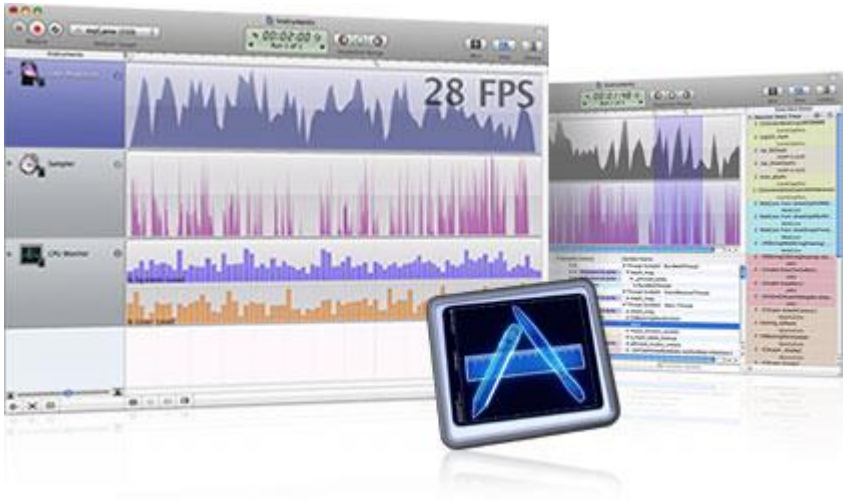

Рисунок 1.3 – Developer Tools

Величезний світ Mac і iPhone застосувань надає користувачеві великий досвід, на який слід спиратися при створенні своєї програми. Додаток має містити в собі елегантний користувальницький інтерфейс і оптимальну продуктивність. Developer Tools включають потужні інструменти оптимізації і аналізу (Instruments and Shark), які дозволяють проаналізувати продуктивність iOS-додатків, поки вони виконуються в емуляторі або на пристрої. Developer Tools збирають дані запущеного додатку і відображають їх у графічному вигляді (тимчасова шкала). Вона дозволяє збирати дані про використання пам'яті, дискової активності, мережеву активність і графічної продуктивності мобільного застосування. Тимчасова шкала може відображати всі типи інформації відразу, дозволяючи співвідносити загальну поведінку додатка, а не тільки поведінку за певним параметром. Все це дозволяє легко визначити проблемні зони додатка, і потім перейти до проблемних рядків коду.

Developer Tools надають засоби для аналізу поведінки програми з плином часу. Наприклад, вікно середовища Developer Tools дозволяє зберігати дані декількох запусків, дозволяючи тим самим бачити, чи поліпшилося поведінка застосування або воно все ще потребує доопрацювання. Також можна зберігати дані цих запусків в документах і відкривати їх в будь-який час.

- Додаткові інструменти (Interactive Playgrounds, Source Control).

1.1.2 Середовище розробки AppCode

AppCode – це інтегроване середовище розробки (IDE) для Swift, Objective-C, C, C ++, JavaScript розробки, побудований на JetBrains' IntelliJ IDEA платформи. Перша версія публічного превью AppCode стала доступна в квітні 2011. Останній стабільний реліз був випущений 6 квітня 2016 року і доступний на офіційному веб-сайті JetBrains [2].

AppCode побудований на платформі IntelliJ IDEA, який написаний на Java. Користувачі можуть розширити свої можливості за рахунок установки плагінів, створених для IntelliJ Platform, а також вони можуть написати свої власні плагіни.

Основні особливості AppCode:

- Автоформатування коду для мови Swift;
- Підсвічування синтаксису і редактора колірних схем для мови Swift;
- Автодоповнення для імен типів, методів і змінних;
- Великий вибір функцій навігації:

AppCode дозволяє:

  - перейти до класу по його імені;
  - перейти до файлу по його імені;
  - перейти до символу по його імені;
  - перейти на оголошення або визначення методу;
  - використовувати для навігації структуру файлу.

- Підсвічування всіх використань символів в межах поточного файлу або пошуку всіх використань поточного символу в проекті;
- Рефакторинг Rename для безпечного перейменування символу в проекті;
- Підтримка юніт-тестування на мові Swift.

1.1.3 Використання Xamarin Studio

Xamarin – це фреймворк для кросплатформенної розробки мобільних додатків (iOS, Android, Windows Phone) з використанням мови C #.

Фреймворк складається з декількох основних частин:

- Xamarin.IOS – бібліотека класів для C #, що надає розробнику доступ до iOS SDK;
- Xamarin.Android – бібліотека класів для C #, що надає розробнику доступ до Android SDK;
- Компілятори для iOS і Android;
- IDE Xamarin Studio;
- Плагін для Visual Studio.

Xamarin Studio - кросплатформенна IDE, яка працює як на Mac OS X, так і на Windows. Вона має простий вигляд, проте за зовнішньою простотою ховається досить потужний інструмент, який включив в себе безліч інструментів, звичних нам в Visual Studio і Resharper [3]:

- Приємне підсвічування синтаксису;
- Автодоповнення коду (включаючи можливість одночасного імпорту namespaces);
- Зручний універсальний пошук за назвами файлів, типам, членам класів тощо;
- Розвинені можливості навігації по проекту: швидкий перехід до опису класу, перехід до базового класу, список місць використання класу тощо;
- Різні механізми рефакторингу і швидка підказка (як alt + Enter в Resharper);
- Досить розвинені механізми налагодження, включаючи стеження, перегляд поточного значення змінної при наведенні, візуалізацію потоків і аналог Immediate window в VS;
- Вбудована інтеграція з системами контролю версій: SVN, Git і TFS (для TFS потрібні сторонні утиліти).

1.1.4 Використання Visual Studio

Xamarin пропонує можливість вести розробку в Visual Studio після установки спеціального плагіна, який доступний в business-ліцензії (999 $) [4]. Після встановлення плагіну для Visual Studio треба налаштувати з'єднання з Mac, яке буде використано при запуску проекту на виконання. Тобто після запуску додаток автоматично пересилається на Mac, де компілюється і завантажується або на симулятор, або на пристрій, при цьому сам процес процес налагодження буде відбуватися в Visual Studio.

Варіантів роботи в Visual Studio кілька. Або використовується віртуальна машина всередині Mac (наприклад, Parallels), куди потрібно ставити Windows і Visual Studio. Або використовуються дві різні фізичні машини, при цьому використовувати один Mac для декількох PC-розробників важко, тому що налагодження вимагає маніпуляцій з симулятором. І останній варіант – використовувати віртуальну машину з Mac OS X (так званий hackintosh). Останній варіант являється найперспективнішим, хоча є деякі обмеження. Наприклад, в Xcode доведеться переміщатися по Storyboard тільки з використанням смуг прокручування, тому що windows-миша не дуже схожа на справжню мишу від Mac з усіма наслідками, що випливають.

1.1.5 Використання Appcelerator Titanium

Appcelerator Titanium – фреймворк з відкритим вихідним кодом для створення мобільних і десктопних кросплатформенних додатків за допомогою мови програмування JavaScript. Основні особливості Appcelerator Titanium включають в себе:

- Крос-платформний API для доступу до власних компонентів користувацького інтерфейсу, такі як панелі навігації, меню і діалогових

вікон, до власної функціональності пристрою, включаючи файлову систему, мережу, геолокації, акселерометр і карти.

- Прозорий доступ до власної функціональності вже не охоплений API.

Весь вихідний код програми розгортається в мобільний пристрій, де він інтерпретується з використанням механізму JavaScript; Rhino від Mozilla використовується на Android і BlackBerry, і JavaScriptCore від Apple використовується на iOS. Програма завантаження займає більше часу, ніж для програм, розроблених з відповідними SDK. Також є проблеми зі стабільністю роботи додатка, а також з управлінням пам’яттю [5].

1.2 Огляд мов програмування для розробки

1.2.1 Swift

Програмування – основа основ комп'ютерної техніки. Корпорація Apple давно відома своїм умінням задавати тон розвитку індустрії на роки вперед, але з огляду на поступове сходження купертіновцев з ринку професійних рішень, надкушене яблуко асоціюється в першу чергу з споживчими товарами. Однак чергове дослідження громадської думки показує, що розробники ПЗ більш ніж задоволені свіжою пропозицією компанії – мовою програмування Swift. Згідно з інформацією ресурсу Stack Overflow, майже 80 відсотків професіоналів із задоволенням працювали або планують працювати з випущеним не так давно інструментом розробки від Apple (рис. 1.4) [6]. Дослідникам вдалося опитати 26 тисяч відвідувачів з більш ніж 150 країн світу. Близько третини постійно зайнятих в сегменті написання мобільного ПЗ респондентів працюють в основному на платформі iOS, а менше половини також займаються створенням додатків для конкуруючої ОС Android. 20 % опитаних не змогли визначитися, швидше за все, сюди входять фахівці, які регулярно розробляють програмне забезпечення для різних платформ.

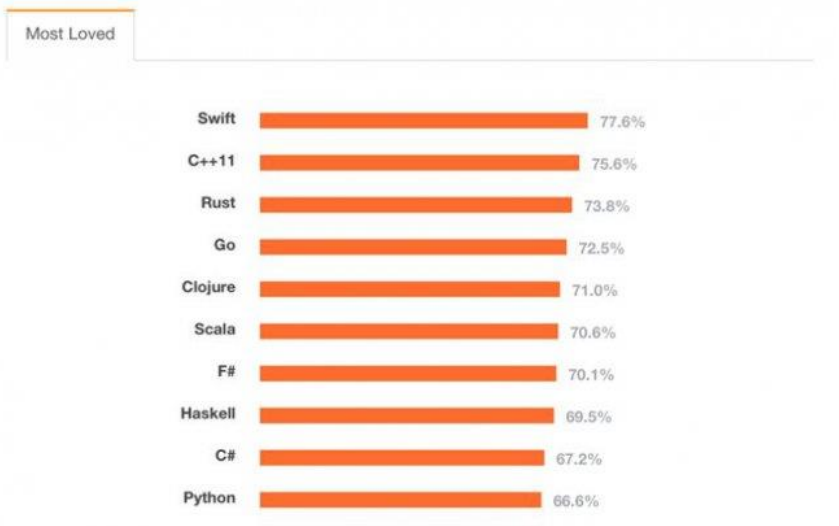

Рисунок 1.4 – Оцінки мов програмування

Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

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.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування.

Основним застосуванням Swift є розробка користувацьких застосунків для MacOS X і Apple iOS з використанням фреймворка Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Вихідний код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті [7].

Swift щільно інтегрований у власницьке середовище розробки Xcode і не може бути використаний відособлено на платформах, відмінних від OS X.

Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

1.2.2 Objective-C

Objective-C – рефлективна, високорівнева об'єктно-орієнтована мова програмування загального призначення, розроблена у вигляді набору розширень стандартної C.

Розроблена компанією Apple, використовується в основному у Mac OS X та GNUStep – середовищах, розроблених на основі стандарту OpenStep, та Cocoa – бібліотеки компонентів для розробки програм. Програму на Objective-C що не використовує цих бібліотек можна скомпілювати для будь-якої платформи, яку підтримує gcc компілятор з підтримкою Objective-C.

Objective-C є розширенням C і тому будь-яку програму на C можна скомпілювати компілятором Objective-C.

ООП в Objective-C включає інтерфейси, класи, категорії. Реалізовано одиничне, невіртуальне спадкування. Немає єдиного базового класу для всіх об'єктів. Всі методи в класі – віртуальні. Категорія – це парадигма, яка дозволяє описувати інтерфейс з методами, які «необов'язково» імплементувати.

Синтакс Objective-C породжений одночасно від C та Smalltalk. Від останньої взято основний семантичний конструкт мови – замість виклику методу об'єктові надсилається повідомлення. Наприклад, якщо клас об'єкта obj

імплементує метод doJob то говориться що об'єкт відкликається на повідомлення doJob. Щоб надіслати повідомлення doJob цьому об'єктові потрібно написати: [obj doJob];

Такий механізм дозволяє надсилати повідомлення навіть до тих об'єктів які не підтримують їх обробки. Такий підхід відрізняється від тих, що використовуються в статично типізованих мовах C++ чи Java.

1.2.3 Основні причини майбутньої популярності Swift

Мови програмування не вмирають швидко, а студії розробники, які чіпляються за згасаючі парадигми, вмирають. В даний момент Swift не тільки витісняє Objective-C, коли мова йде про розробки додатків під OS X, iOS і WatchOS, але і в недалекому майбутньому замінить C для внутрішнього програмування, для платформ Apple [8].

Завдяки кільком ключовим особливостям, Swift має потенціал стати єдиною мовою програмування, для створення захоплюючих, гнучких додатків або програм на багато років.

У Apple великі надії на Swift. Компанія настільки ефективно оптимізувала компілятор і саму мову в принципі, що багато можливостей ще тільки належить розкрити. Якщо вірити офіційній документації по Swift, то можна з впевненістю сказати, що Swift «призначений для зльоту від "hello, world" до цілої операційної системи» [9]. Виділимо основні причини:

- Swift більш читабельна мова, ніж Objective-C:

У Objective-C є всі болячки, які можуть бути у мови, побудованої на C. Для диференціації ключових слів і типів від C типів, Objective-C вводить нові ключові слова, використовуючи символ @. Так як Swift не побудований на C, то він може об'єднати всі ключові слова і видалити численні символи @ перед кожним Objective-C типом або перед пов'язаною з об'єктом ключовим словом.

Swift переглядає загальноприйняті умови успадкування. Таким чином, більше не потрібно ставити кому в кінці рядка або писати круглі дужки, щоб оточити умовні вирази всередині операторів if / else [7]. Найбільшою перевагою Swift перед Objective-C є те, що виклики методу не розташовуються усередині один одного. Для виклику функцій і методів в Swift використовується стандартний, розділений комами, список параметрів в круглих дужках. В результаті отримуємо мову зі спрощеним синтаксисом і граматикою.

Код в Swift дуже нагадує природну англійську мову, на додаток до інших сучасних популярних мов програмування. Читабельність коду в Swift полегшує роботу для програмістів JavaScript, Java, Python, C #, C ++, на відміну від Objective-C.

- Swift легше підтримувати:

Спадщина утримує Objective-C позаду: мова не може розвиватися без розвитку C. C вимагає від програмістів підтримки 2 кодових файлів для поліпшення часу установки і ефективності створення додатка – вимога, яка переноситься на Objective-C.

Swift скасовує вимогу двох-файлів. Xcode і компілятор LLVM може з'ясувати залежність і виконати покрокові зміни автоматично в Swift 1.2. Середовище розробки Xcode і компілятор LLVM (Low Level Virtual Machine) може з'ясувати залежність і виконати покрокові зміни автоматично в Swift. Тому, поділу змісту від тіла стає справою минулого. Swift поєднує в собі заголовок Objective-C (.h) і файли реалізації (.m) в одному файлі коду (.swift).
Двох файлова система Objective-C накладає додаткову роботу на програмістів - і це робота, яка відволікає їх від більш важливих завдань. У Objective-C потрібно вручну синхронізувати імена методів і коментарі між файлами [1].

Xcode і компілятор LLVM можуть працювати непомітно для програміста, знижуючи його навантаження. Використовуючи Swift, програмісти роблять менше допоміжних дій і можуть витратити більше часу на створення логіки

додатка. Swift відмовляється від шаблонної праці і покращує якість коду, підтримування коментарів і функцій [10].

- Swift – безпечніша мова:

Одним з цікавих моментів в Objective-C можна вважати спосіб обробки вказівників, особливо nil (NULL). У Objective-C нічого не трапиться, якщо спробувати викликати метод зі змінною покажчика nil (неініціалізованих) [9]. Вираз або рядок коду стає невиконуваним і може здатися, що вираз спрацює, але насправді буде повно помилок. Нездійсненність призводить до непередбачуваної поведінки, що є ворогом програмістів, які намагаються знайти і виправити випадкові збої або зупинити поведінку, яка відхиляється від норми.

Опціональні типи дають можливість існування в Swift коді nil (опціонального значення), що говорить про можливість створення помилки компілятора при написанні поганого коду. Це створює короткий цикл зворотного зв'язку і дозволяє програмістам писати програми більш впевнено. Проблеми можуть бути усунені при вже написаному коді, що значно зменшує кількість часу і грошей, які витрачаються на виправлення помилок, в порівнянні з передачею вказівника на NSError в Objective-C [9].

Традиційно в Objective-C, якщо значення було повернуто з методу, то програміст був зобов'язаний зафіксувати поведінку вказівника повернутої змінної (з використанням коментарів і найменування методу). В Swift опціональні типи і типи значень дозволяють явно визначити метод, якщо значення існує, або якщо воно може бути опціональним (тобто, значення може існувати або може бути nil).

Для забезпечення передбачуваної поведінки, Swift викликає помилку при виконанні коду, якщо використовується опціональна змінна nil. Ця помилка забезпечує узгоджену поведінку, яке полегшує процес усунення помилок, тому що змушує програміста вирішити проблему відразу. Ця помилка виникне на

тому рядку коду, де була використана опціональна змінна nil. Це означає, що помилка буде виправлена своєчасно або її вдасться уникнути зовсім.

- Незалежне управління пам'яттю:

Swift уніфікований так, як ніколи не був Objective-C. Підтримка автоматичного підрахунку посилань (ARC) є повною з процедурних і об'єктно-орієнтованих шляхах коду. У Objective-C ARC підтримується всередині Cocoa API і об'єктно-орієнтованого коду, але він не доступний для C коду і API, наприклад Core Graphics [3]. Це означає, що програміст повинен взяти на себе управління пам'яттю при роботі з Core Graphics APIs і з іншими старішими API, доступних на iOS. Величезні витоки пам'яті, які програміст може мати в Objective-C, неможливі в Swift.

Програмісту не доводиться думати про пам'ять для кожного цифрового об'єкта, який він створює. Тому що ARC обробляє всі управління пам'яттю під час компіляції, і витрати, які пішли б на управління пам'яттю, тепер можуть бути сфокусовані на основній лозіці додатка і нових можливостях. Це відбувається тому, що ARC в Swift працює і на процесуальному, і на об'єктно-орієнтованому коді, і тепер не потрібно контекстних переходів для програмістів, навіть якщо вони пишуть код, який розрахований на більш старі API. Це є проблемою для поточної версії Objective-C.

Автоматичне і високопродуктивне управління пам'яттю було проблемою, яку Apple змогли вирішити і довели, що це може підвищити продуктивність. З іншого боку і в Objective-C, і в Swift немає залежності від темпу роботи Garbage Collector, який використовується для очищення невикористаної пам'яті, як в Java, Go або C #. Це важливий фактор для будь-якої мови програмування, що використовується для чутливої графіки і наданні входу для користувача, особливо на таких пристроях як iPhone, Apple Watch або IPad (де затримка в роботі сприймається як розчарування і змушує користувача думати, що програма не працює).

- Swift вимагає меншу кількість коду:

Swift зменшує кількість коду, необхідного для повторюваних заяв і рядків (рис. 1.5) [9]. У Objective-C, працюючи з текстовими рядками, для того, щоб об'єднати дві частини інформації, потрібно зробити безліч кроків. Для додавання двох рядків Swift використовує оператор «+». Ця особливість відсутня у Objective-C. Така підтримка для об'єднаних символів і рядків має важливе значення для будь-якої мови програмування, яка використовує відображення тексту для користувача на екрані.

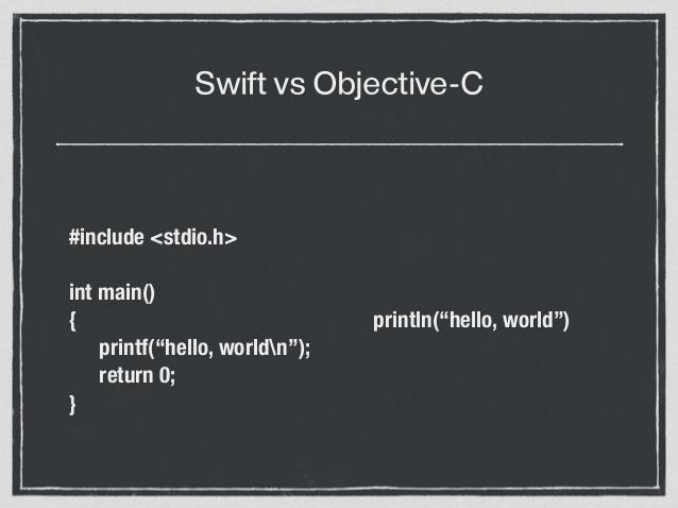

Рисунок 1.5 – Порівняння Swift і Objective-C

Система типів в Swift знижує складність в написанні коду, так як компілятор може з'ясовувати типи. Наприклад: Objective-C вимагає, щоб програмісти запам'ятовували спеціальні маркери рядків (%S, %d, %@) і використовували розділений комами список. Swift підтримує інтерполяцію рядків, що усуває необхідність запам'ятовування символів і дозволяє програмістам вставляти змінні, такі як назва ярлика або кнопки, безпосередньо у вбудований рядок користувача. Тип виведення системи і рядки інтерполяції зменшують кількість помилок, які поширені в Objective-C.

У Objective-C, якщо порушити порядок розташування або використовувати неправильний символ, мобільний додаток працювати не буде. Swift ж знову зменшує кількість написання коду вбудованій підтримці для обробки текстових рядків і даних.

- Swift швидше:

Видалення застарілих конвенцій набагато поліпшило «двигун» Swift. Тестування продуктивності коду в Swift як і раніше вказує на те, що Apple продовжує покращувати швидкість роботи додатків на Swift (рис. 1.6) [8].

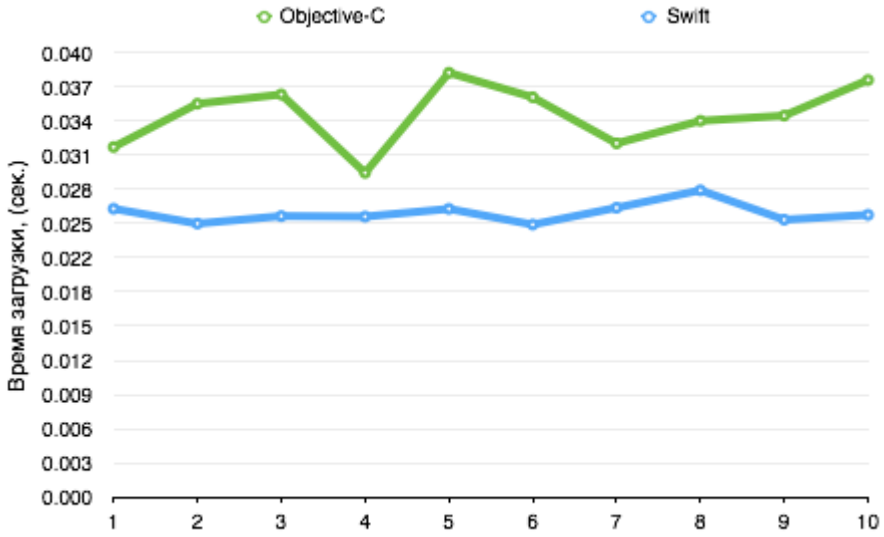

Рисунок 1.6 – Швидкість роботи додатків

Відповідно до даних Primate Labs, за показниками популярного тесту GeekBench (крос-платформний еталонний тест для вимірювання швидкодії процесора і підсистеми пам'яті комп'ютера) Swift наблизився до експлуатаційних характеристик C++ по обмеженню швидкості обчислень з використанням алгоритму Мандельброта (знаходження множини Мандельброта) в грудні 2014 року [11]. У лютому 2015 року Primate Labs виявили, що Xcode 6.3 Beta поліпшив продуктивність алгоритму GEMM в Swift (алгоритм обмеженої пам’яті з послідовним доступом великих масивів (a memory-bound algorithm with sequential access of large arrays)) до коефіцієнта 1,4. Початкова імплементація FFT (алгоритму обмеженої пам'яті з випадковим доступом великих масивів) – поліпшення продуктивності в 2,6 рази.

Swift показав і інші поліпшені показники при подальшому тестуванні: 8,5-кратне підвищенням для FFT алгоритму (залишивши C++ тільки з приростом продуктивності в 1,1 рази). Також, Swift перевершує C++ для алгоритму Мандельброта з коефіцієнтом в 1,03 [12].

Практично, Swift на одному рівні з C++ для FFT і алгоритму Мандельброта. Відповідно до даних Primate Labs алгоритм GEMM показав, що компілятор Swift не може векторизувати код так, як може компілятор C++. У C++ є невелика перевага, але Apple обіцяють все виправити в наступних версіях Swift.

- Менше зіткнень імен з проектами відкритого вихідного коду:

Існує одна проблема, яка переслідує код Objective-C – це відсутність формальної підтримки простору імен (namespaces), які були створені як вирішення проблеми C++ для зіткнень при назві файлів. Коли це зіткнення відбувається в Objective-C, це вважається помилкою лінкера (редактора зв'язків), і додаток не може працювати. Обхідні шляхи існують, але вони мають потенційні підводні камені. Існує загальна згода на використання дво- або трибуквених префіксів, для відмінності написаного коду Objective-C кимось, і від такого ж коду для Facebook.

Swift дає можливість використання неявних просторів імен, що дозволяє одному і тому ж файлу з кодом використовуватися в численних проектах, без ризиків збою і вимог найменувань, таких як NSString (NextStep – компанія Стіва Джобса, після його звільнення з Apple) або CGPoint (Core Graphics). В кінцевому рахунку, ця функція в Swift сприяє більшій продуктивності для програмістів і означає, що їм не доведеться займатися додатковою роботою, яка існує в Objective-C. Також можна замітити, що в Swift є зв'язок з простими іменами, такими як Array, Dictionary і String, замість NSArray, NSDictionary, і NSString, які народилися через відсутність простору імен в Objective-C.

В Swift простори імен засновані на цілях коду. Це означає, що програмісти можуть диференціювати класи або значення, використовуючи ідентифікатор простору імен. Це дуже вагома зміна в Swift. Вона значно полегшує включення проектів з відкритим вихідним кодом, а також додавання рамок і бібліотек в вихідний код. Простори назв дозволяють різним програмним компаніям створювати ті ж імена файлів, не турбуючись про

зіткненнях при інтеграції проектів з відкритим вихідним кодом. Тепер і Facebook, і Apple можуть використовувати файл об'єктного коду FlyingCar.swift без будь-яких помилок або ризиків збою.

- Swift підтримує динамічні бібліотеки:

Найбільше зміна в Swift, яка не отримало достатньої уваги – це перехід від статичних бібліотек, які оновлюються тільки при великих оновлення (iOS8, iOS7 тощо), до динамічних бібліотек. Динамічні бібліотеки – це виконувані шматки коду, які можуть бути приєднані до додатка. Ця функція дозволяє додаткам поточної версії Swift зв'язатися з новішими версіями мови в зв'язку з його постійним розвитком.

Розробник надає додаток разом з бібліотеками, обидві з яких підписані електронним підписом і мають сертифікат розвитку для забезпечення цілісності (hello, NSA). Це означає, що Swift може розвиватися швидше ніж iOS, який в свою чергу висуває вимоги для сучасної мови програмування. Зміни в бібліотеках включаються в останнє оновлення, в додаток на App Store, і все чудово працює.

Динамічні бібліотеки ніколи не підтримувалися iOS до запуску Swift і iOS8, хоча динамічні бібліотеки підтримувалися на Mac протягом дуже довгого часу. Динамічні бібліотеки є зовнішніми для виконання додатками, але включені в пакет програми при завантаженні з App Store. Це зменшує початковий розмір програми, так як воно завантажується в пам'ять, а зовнішній код додається тільки коли використовується.

Можливість відкласти завантаження в мобільному додатку або у вбудованому додатку на Apple Watch поліпшить ефективність відтворення для користувача. Це одна з відмінностей, яке робить екосистему iOS більш гнучкою. Apple була зосереджена на завантаженні тільки аресурсів і тепер компілювання і зв’язування коду відбувається на льоту. Завантаження в ході роботи зменшує кількість часу очікування, поки ресурс не буде відображений на екрані.

Динамічні бібліотеки Swift дозволяють проводити зміни і поліпшення простіше, ніж коли-небудь раніше. Програмістам більше не потрібно чекати iOS релізів, щоб отримати поліпшення Apple для Swift.

- Interactive Playgrounds – спонукання до інтерактивного програмування:

Одним із важливих нововведень Swift, в порівнянні з Objective-C, є можливість писати код і переглядати результати в реальному часі. Ми можемо внести деякі зміни в програмі і відразу побачити результат, так що нам не потрібно перекомпільовувати і перезапускати програму. Це має назву Interactive Playgrounds. Також тут можна використовувати Quick Look, який відображає графіку або список результатів, Timeline Assistant, який допомагає експериментувати з кодом UI (інтерфейсом користувача) або продивлятися повний цикл створення анімацій. Даний код можна перенести в основний проект (рис. 1.7) [1].

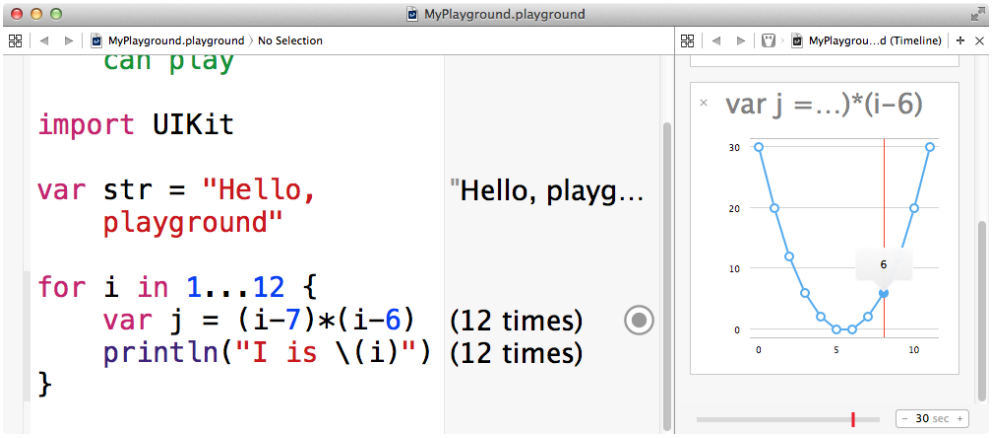

Рисунок 1.7 – Interactive Playgrounds

Apple додала вбудоване виконання коду під час його написання, щоб допомогти програмістам створювати шматки коду або писати алгоритми і зразу ж отримувати результат, що може поліпшити швидкість написання коду, тому що та модель, яка традиційно потрібна програмісту, може бути замінена візуалізацією даних в пісочниці. Програмування – повторюваний процес, і кожен strain може бути зменшений або використаний на додаток до творчого процесу. Це зробить роботу програмістів більш продуктивною і звільнить їх від зайвої роботи, яку додавали програмістам традиційні компілятори.

Можна відмітити, що Interactive Playgrounds не так значимі для новачків, як для досвідчених програмістів. Наприклад, показуючи роботу змінної в Interactive Playgrounds, програмісту-новачку важко зрозуміти необхідність використання змінної Float замість змінної Int. Необхідність стає очевидною, коли цю змінну можна показати в додатку, що скролл запам'ятовує останнє положення в новинах Facebook. Також, починаючи з Xcode 7 можна створювати коментарі до коду, використовуючи форматований текст з жирним шрифтом або курсивом, маркований список, вбудовані зображення і посилання. Таким чином, Interactive Playgrounds розроблені для того, щоб зробити програмування більш інтерактивним і доступним.

- Swift - це майбутнє, на яке може вплинути будь-хто:

Objective-C нікуди не зникне, але і не буде зазнавати великих змін, завдяки появі Swift. Швидше за все, деякі зміни перекочують в Objective-C, але спадщина Objective-C в C означає, що воно лише буде все поглинати.

Swift дає можливість суспільству вплинути на мову, яка буде використовуватися для створення програмного забезпечення, вбудованих систем і інших девайсів, таких як Apple Watch.

Apple спрямована на забезпечення найкращого споживчого досвіду і впроваджує тільки ті функції, які вважає гідними уваги. В Swift 2.2 і релізі Xcode 7, Apple вже усунула тисячі помилок за допомогою популярного Apple Bug Reporter utility. Команда підтримки розвитку та еволюції Swift дуже зацікавлена в тому, щоб мова була покращено.

1.3 Обгрунтування вибору інструментів для розробки

Перед кожним розробником завжди постає питання: які саме інструменти вибрати для роботи. Головними критеріями стають простота у використанні та швидкість. Тому в дипломній роботі будуть використовуватися найновіші продукти від компанії Apple – середовище розробки Xcode і мова

програмування Swift. Адже вони увібрали найкращі особливості універсальних інструментів для розробки мобільного програмного забезпечення.

Порівнюючи Swift і Objective-C можна з впевненістю сказати, що Swift – це сучасні норми синтаксису, ефективне управління пам'яттю, висока швидкість роботи і інтерактивність. У Objective-C цього немає, але зате є надійність, база документації, прикладів, шаблонів і багато досвідчених програмістів. Swift – це майбутнє, яке вже не за горами. Все пізнається в порівнянні і лише відпрацювавши певну кількість проектів, можна стверджувати, чи зручна мова, і чи можна на ній ефективно працювати.

Проаналізувавши особливості кожного з середовищ розробки, Xcode перемагає усіх вищепредставлених учасників, адже працювати в ньому – одне задоволення. Воно гнучке, швидке, потужне і здатне завжди прийти на допомогу. Xcode стає все краще, незважаючи на складні заходи, які вживаються Apple з метою утримання повного контролю над iOS додатками і пристроями. Робочий простір в Xcode тримає розробника зосередженим. Під час написання коду, в реальному часі можна побачити помилки компілятора або проблеми, а також повідомлення з докладним описом корисної інформації. Відладчик працює плавно, а симулятор – швидкий і орієнтований на користувача. Отже, Xcode перевершує всі інші IDE.

1.4 Висновок

Тема розробки мобільних застосувань для платформи iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку. В даному розділі було проаналізовані основні інструменти для розробки мобільного програмного забезпечення під платформу iOS. Також було доведено і обгрунтовано правильність вибору засобів для проектування. На

основі проведеного аналізу встановлено, що найкращим середовищем розробки є Xcode, а мовою програмування – Swift.

2. ОСНОВНІ МЕТОДИ ТА ОСОБЛИВОСТІ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Архітектура iOS

Архітектура iOS схожа з базовою архітектурою операційної системи Mac OS X. На найвищому рівні iOS являє собою проміжний шар між обладнанням пристроя та додатками, які відображаються на екрані (рис. 2.1). Дуже рідко доводиться створювати додатки, які будуть звертатися до обладнання безпосередньо. Замість цього, додатки взаємодіють з обладнанням через набір чітко визначених системних інтерфейсів, які захищають додатки від змін обладнання. Ця абстракція дозволяє дуже легко писати програми, які коректно працюють на пристроях з різними апаратними можливостями.

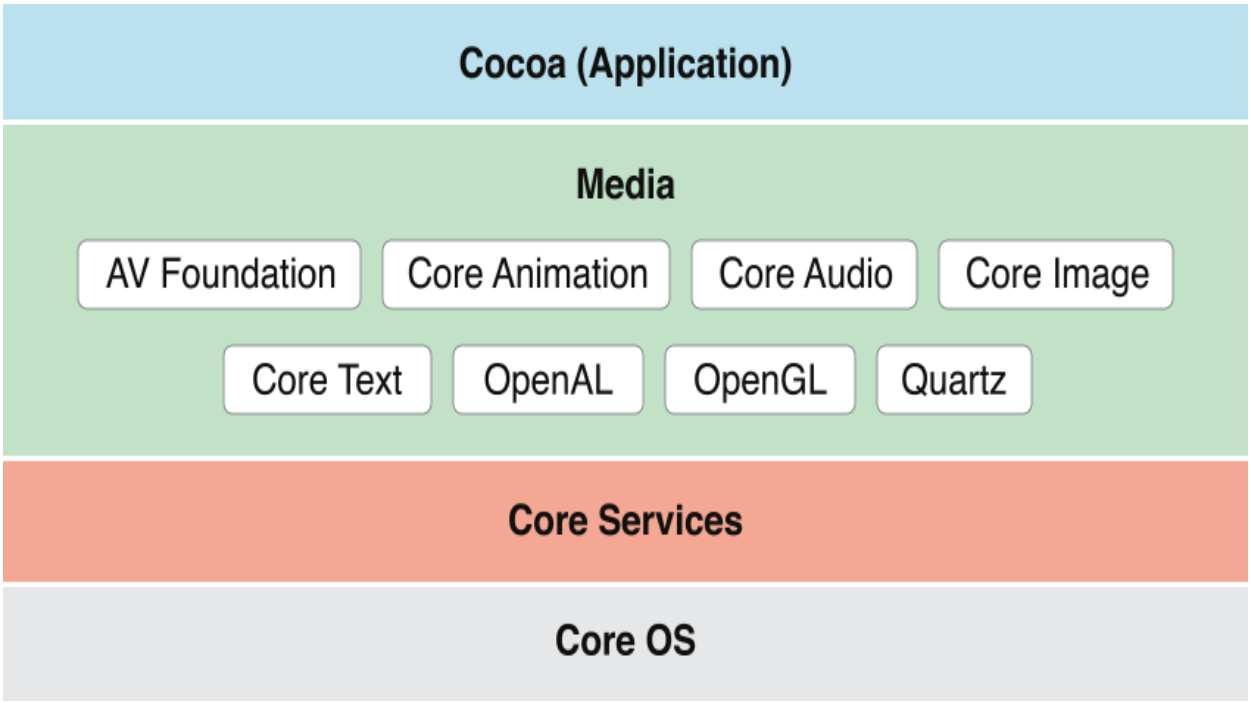

Рисунок 2.1 – Архітектура платформи

Таблиця 2.1 – Рівні архітектури платформи iOS

Рівні	Призначення
Cocoa Touch	Містить ключові бібліотеки для розробки iOS додатків
Media	Містить технології для роботи з графічними, відео і аудіо даними
Core Service	Містить основні системні служби, які використовуються усіма додатками
Core OS	Містить низькорівневі компоненти, на яких побудовані інші технології

Хоча додатки в цілому захищені від змін обладнання, але доводиться враховувати відмінності між пристроями при написанні коду. Наприклад, у деяких пристроїв є камера, а у деяких її немає. Якщо додаток вміє працювати при наявності або відсутності якоїсь функції, то використовуючи інтерфейс відповідної бібліотеки можна визначити, доступна ця функція чи ні.

Додатки, яким потрібна наявність певного обладнання, повинні декларувати цю вимогу в файлі зі списком властивостей додатки (Info.plist). Реалізація технологій iOS може бути представлена у вигляді набору шарів (див. рис. 2.1). На самому нижньому шарі операційної системи знаходяться основні служби та технології, від яких залежать всі додатки; на більш високих рівнях знаходяться складніші служби і технології.

При написанні додатків всюди, де це можливо, рекомендується використовувати бібліотеки вищого рівня, ніж бібліотеки низького рівня. Бібліотеки вищого рівня написані для того, щоб забезпечити об'єктно-орієнтовані абстракції для низькорівневих структур. Ці абстракції істотно полегшують написання коду, тому що вони зменшують обсяг коду, який необхідно написати і приховують досить складні функції, такі як сокети і потоки. І хоча вони приховують низькорівневі функції, ці функції як і раніше доступні для розробників. Розробники, які вважають за краще використовувати

низькорівневі бібліотеки або хочуть скористатися наявними можливостями, яких не надають високорівневі бібліотеки, можуть їх використовувати.

Основні переваги iOS:

- простота роботи;
- висока стабільність;
- безпечність;
- відсутність вірусів;
- відсутність фрагментованості;
- регулярне оновлення;
- зручність розробки додатків;
- широкий вибір програм та ігор.

Основні недоліки:

- прив’язка до апаратного забезпечення одного виробника;
- загальна «закритість» системи.

2.2 Управління пам’яттю (ARC)

ARC (automatic reference counting) використовується для відстеження та управління пам'яттю мобільного застосування. ARC автоматично звільняє пам'ять, яка використовувалася екземпляром класу, коли ці екземпляри більше не потрібні.

Кожен раз, коли створюється екземпляр класу, ARC виділяє шматок пам'яті для зберігання інформації цього екземпляра. Цей шматок пам'яті містить інформацію про тип екземпляра, про його значення і про будь-які властивості, пов'язаних з ним.

Додатково, коли екземпляр більше не потрібен, ARC звільняє пам'ять, використану під цей екземпляр, і направляє цю пам'ять туди, де вона потрібна. Це свого роду гарантія того, що непотрібні екземпляри не будуть займати пам'ять.

Однак, якщо ARC звільнить пам'ять використовуваного екземпляра, то доступ до властивостей або методів цього екземпляра буде неможливий. Якщо спробувати отримати доступ до цього екземпляра, то додаток швидше за все видасть помилку і зупинить свою роботу.

Для того, щоб потрібний екземпляр не пропав, ARC веде облік кількості властивостей, констант, змінних, які посилаються на кожен екземпляр класу. ARC не звільнить екземпляр, якщо є хоча б одне активне посилання.

Для того щоб це було можливо, кожен раз як привласнюється екземпляр властивості, константі або змінній створюється strong reference (сильний зв'язок) з цим екземпляром. Такий зв'язок називається "сильним", так як він міцно тримається за цей екземпляр і не дозволяє йому звільниться до тих пір, поки залишаються сильні зв'язки.

Наприклад, нехай є клас Person, який визначає константну властивість name:

```
class Person {
        let name: String
        init(name: String) {
        self.name = name
        print("\(name) is being initialized")
        }
        deinit {
        print("\(name) is being deinitialized")
        }
}
```

Клас Person має ініціалізатор, який встановлює властивість name екземпляра і виводить повідомлення для відображення того, що йде ініціалізація. Так само клас Person має деініціалізатор, який виводить повідомлення, коли екземпляр класу звільняється.

Наступний шматок коду визначає три змінні класу Person?, який використовується для установки декількох посилань до нового екземпляру Person в наступних шматках коду. Так як ці змінні опціонального типу Person?,

а не Person, вони автоматично ініціалізуються зі значенням nil, і не мають жодних посилань на екземпляр Person:

```
var reference1: Person?
var reference2: Person?
var reference3: Person?
```

Тепер можна створити екземпляр класу Person і привласнити його однією з цих трьох змінних:

```
reference1 = Person(name: "John Appleseed")
// Prints "John Appleseed is being initialized"
```

Повідомлення "John Appleseed is being initialized" виводиться під час того, як викликається ініціалізатор класу Person. Це підтверджує той факт, що відбулася ініціалізація.

Так як новий екземпляр класу Person було присвоєно змінній reference1, значить тепер існує сильне посилання між reference1 і новим екземпляром класу Person. Тепер у цього екземпляра є як мінімум одне сильна посилання, значить ARC тримає під Person пам'ять і не звільняє її.

Якщо надати іншим змінним той же екземпляр Person, то додасться два сильних посилання до цього екземпляра:

```
reference2 = reference1
reference3 = reference1
```

Тепер екземпляр класу Person має три сильні посилання. Якщо знищите два з цих трьох посилань (включаючи і первісне посилання), присвоємо nil двом змінним, то залишиться одне сильне посилання, і екземпляр Person не буде звільнений:

```
reference1 = nil
reference2 = nil
```

ARC не звільнить екземпляр класу Person до тих пір, поки залишається останнє сильне посилання, знищивши яку вказуємо на те, що екземпляр більше не використовується:

```
reference3 = nil
// Prints "John Appleseed is being deinitialized"
```

2.3 Цикл життя UIViewController

Більшість додатків під iOS пишуться з використанням фреймворку UIKit, а значить використовують UIViewController. Відповідно патерну проектування MVC (рис. 2.2) UIViewController являється контроллером, який зв’язує модель даних з візуальним представленням. Тобто, саме View - це просто аркуш паперу, на який можна добавити такі елементи як кнопки, toolbars, text views тощо. Коли натискається кнопка, то повідомлення про це отримує саме UIViewController, і саме він контролює всю діяльність, яка там відбувається. Кожен UIViewController контролює одне View. Також на UIViewController «покладено» обов'язок контролю не тільки над своїм життєвим циклом, але і над циклом життя власного UIView.

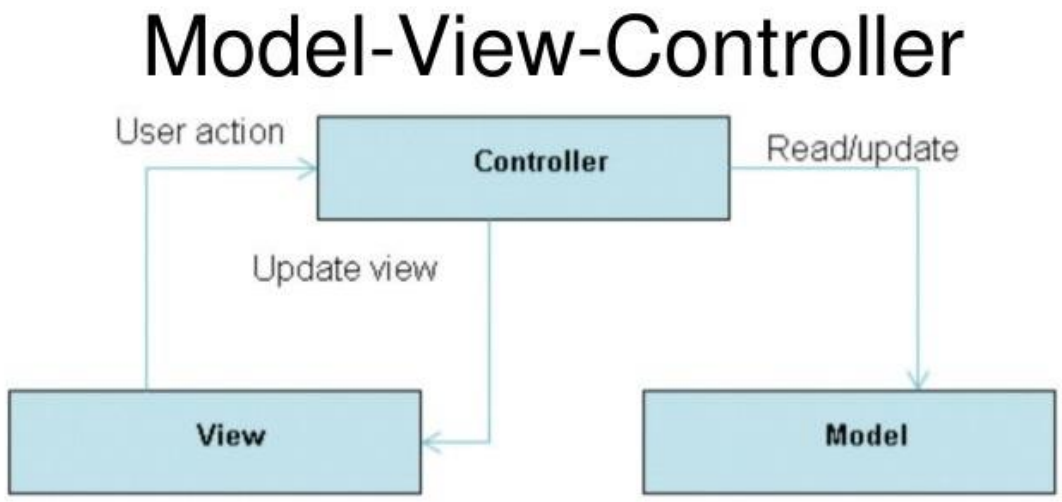

Рисунок 2.2 – Паттерн проектування MVC

Життєвий цикл являє собою наступне (рис. 2.3):

- Створення UIViewController:
  - init
  - initWithNibName:
- Знищення UIViewController:
  - dealloc

- Створення UIView:
  - isViewLoaded
  - loadView
  - viewDidLoad
  - initWith
- Обробка зміни стану UIView:
  - viewDidLoad
  - viewWillAppear
  - viewDidAppear
  - viewWillDisappear
  - viewDidDisappear
  - viewDidUnload
- Знищення UIView:
  - viewDidUnload
- Обробка браку пам'яті
  - didReceiveMemoryWarning

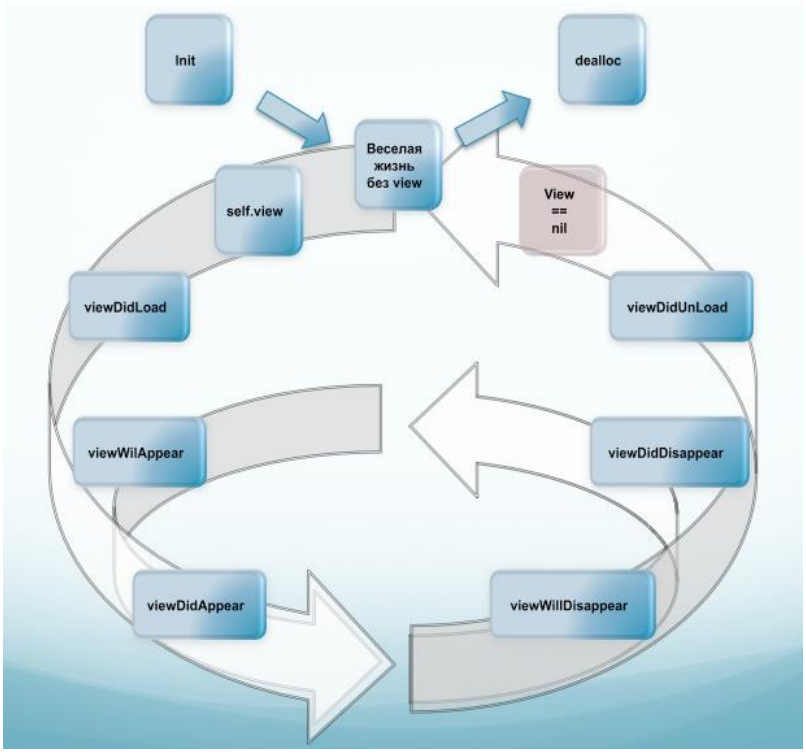

Рисунок 2.3 – Цикл життя UIViewController

Початок «життя»

Незалежно від того, як саме інціалізується контролер (UIViewController) (з підключення .nib файлу або нормально), відбувається виклик методу init, в якому і з'являється значення змінної self, вірніше, її значення перестає бути nil і замінюється вказівником на конкретний блок пам'яті. Ініціалізується тільки сам контролер. Ніякого UIView не існує (хоча він і ініціалізований, тобто звернення до self.view можливо). Якщо використовуються .nib файли, то ніяких зв’язків на цьому етапі ще немає. Відповідно, спроба створення або звернення до візуальних компонентів призведе до помилки. У будь-який момент можна перевірити, завантажений UIView чи ні (відповідно, можна звертатися до візуальних елементів) за допомогою методу isViewLoaded. Головна його особливість у тому, що він не «провокує» завантаження UIView, якщо ще не завантажений.

Створення UIView

У загальному випадку, створення образу (його завантаження в пам'ять) починається в той момент, коли відбудеться будь-яке звернення до self.view. Буде викликаний метод loadView, результатом якого буде або створений порожній UIView або з елементами з .nib файлу. Саме в цей момент котроллер і його UIView стають повністю доступними. По закінченню завантаження буде викликаний метод viewDidLoad. Якщо інтерфейс створюється з використанням коду, то власні елементи інтерфейсу можна створювати , додавати і налаштовувати як в методі loadView, так і в методі viewDidLoad. Якщо використовуються .nib файли, то для настройки елементів інтерфейсу доступний тільки метод viewDidLoad.

Життя UIView

- viewDidLoad – UIView завантажений і повністю готовий до роботи. Тут відбувається так звана «друга частина ініціалізації контролера».

- viewWillAppear – буде викликаний перед тим, як почати відображення UIView. У разі, якщо для відображення використовується анімація – перед початком анімації.
- viewDidAppear – буде викликаний після закінчення відображення UIView. У разі, якщо для відображення використовується анімація – після закінчення анімації відображення.
- viewWillDisappear – теж, що і viewWillAppear, тільки коли UIView припиняє відображення.
- viewDidDisappear – теж, що і viewDidAppear, тільки коли UIView припиняє відображення.
- viewDidUnload – UIView вивантажено з пам'яті але не елементи інтерфейсу, які створювалися. Саме в цьому методі їх потрібно знищити.

Кінець життєвого шляху контролера

Коли контролер буде знищений (вказівник на нього буде nil), ARC очистить виділену для нього пам'ять. Перед цим буде викликаний метод dealloc. Саме в ньому нобхідно виконати очистку і обнулення всіх вказівників, які створювалися для використання всередині контролера.

Обробка браку пам'яті

Метод didReceiveMemoryWarning викликається системою при нестачі пам'яті. За замовчуванням, реалізація цього методу для контролера, який не перебуває у видимій області, викличе звільнення і видалення UIView, що, в свою чергу, призведе до виклику viewDidUnload.

2.4 Storyboard

Storyboard – це середовище для розробки користувацького інтерфейсу мобільного застосування (рис. 2.4).

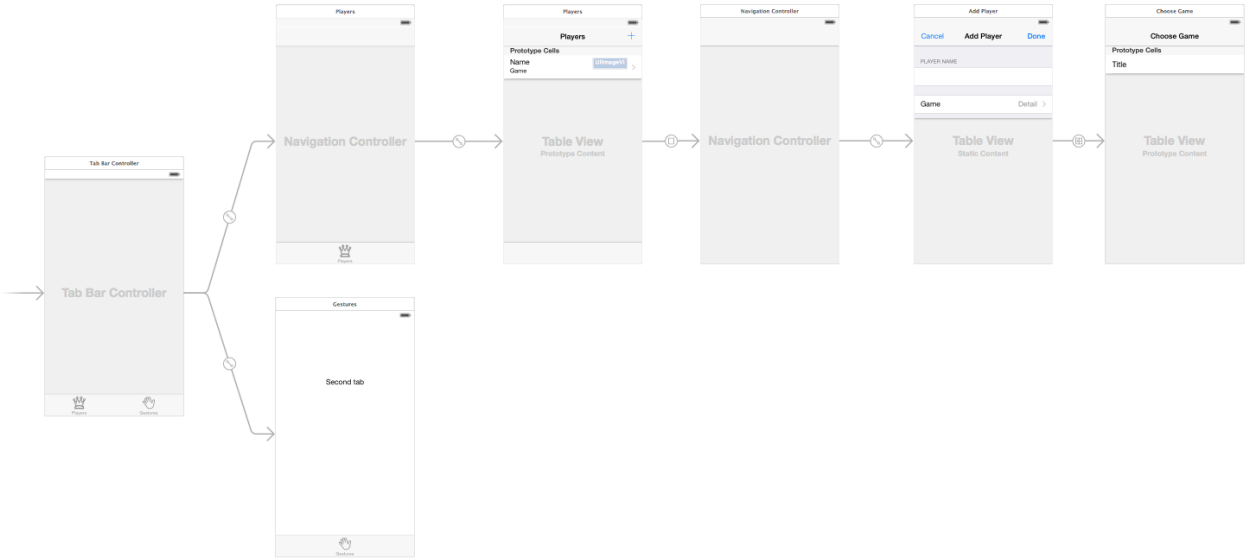

Рисунок 2.4 – Ієрархія зв’язків UIView

Головна перевага Storyboard – не маючи уявлення, як працює дане мобільне застосування, розробник бачить, як виглядають його сторінки і як вони пов’язані між собою. Якщо створюється додаток з великою кількістю різних сторінок, то Storyboard дозволяють істотно зменшити кількість коду для їх з’єднання, щоб забезпечити перехід від однієї сторінки до іншої. Раніше розробникам доводилося створювати окремі файли (файли nib або xib) інтерфейсу для кожного окремого view controller (сторінки). Але тепер додатки використовують єдиний Storyboard, який включає в себе дизайн всіх сторінок і який визначає взаємодію між ними.

Storyboards мають ряд переваг:

- У Storyboard дуже зручно спостерігати загальну картину застосування і взаємозв'язків між його сторінками. У Storyboard можна стежити за будь-чим, тому що загальний дизайн програми міститься в одному єдиному файлі, а не розподілений між декількома файлами nib.
- Storyboards можуть описувати переходи між різними вікнами. Ці переходи називаються “segues”, які створюються шляхом з'єднання двох сторінок прямо в Storyboard. Завдяки цим segues використовується менше коду для користувацького інтерфейсу.

- Storyboards полегшують вашу роботу з table view, з cell. Можна створювати таблиці практично повністю в Storyboards, але ж знову таки це саме те, що зменшує в рази код, який би довелося писати.
- Storyboards спрощують роботу при використання AutoLayout. Storyboards - потужна функція, яка дозволяє визначати математичні взаємозв’язки між елементами, які мають певні розміри і позиції, і так само спрощує роботу по відображенню мобільного застосування на різних пристроях з різними розширеннями екрану.

2.5 Технологія AutoLayout

AutoLayout (автомакет, автокомпановка) – це механізм розташування елементів користувача на екрані. AutoLayout використовується для побудови динамічних призначених для користувача інтерфейсів, масштабованих і адаптованих до різних форматів і розширень екранів пристроїв, а також до їх орієнтацій. AutoLayout прийшов на зміну системі «пружин і розтяжок», які застосовуються в попередніх версіях iOS SDK.

Також AutoLayout робить інтернаціоналізацію більш простим завданням, розміщувати текст змінної довжини на екрані стає простіше, також підтримуються мови з напрямком письма справа наліво, такі як іврит і арабська.

Робота AutoLayout заснована на зв'язках (constraints), які встановлюють геометричні відносини між уявленнями призначеного для користувача інтерфейсу. Наприклад, можна створити зв'язок, який говорить: «текстова мітка повинна бути закріплена на деякій відстані від лівого краю батьківського уявлення і з'єднана з лівим краєм кнопки, з додавання між ними проміжку в 10 px».

AutoLayout бере задані зв'язки і математично обчислює ідеальні позиції і розміри для всіх уявлень. Розробнику не потрібно більше встановлювати

розміри уявлень вручну, задавати їх координати розташування – AutoLayout бере цю роботу на себе.

Створювати зв'язки можна як програмно, так і за допомогою Interface Builder.

2.6 Огляд основних фреймворків

2.6.1 Фреймворк UIKit

UIKit – це платформа, яка визначає компоненти ядра iOS-додатка від ярликів і кнопок до табличних видів і контролерів навігації. У той час, як платформа Foundation визначає класи, протоколи і функції в розробці як під iOS, так і під OS X, платформа UIKit спрямована виключно на iOS-розробку. Це еквівалент середовища розробки Application Kit або AppKit для розробки OS X [13].

Як і Foundation, UIKit визначає класи, протоколи і функції, типи даних і константи. Він також додає додатковий функціонал в різні класи Foundation, начебто NSObject, NSString і NSValue за допомогою використання категорій Objective-C.

UIKit забезпечує ключову інфраструктуру для реалізації графічних, заснованих на подіях iOS-додатків. Кожне iOS-додаток використовує цю бібліотеку для реалізації наступних центральних можливостей [13]:

- управління додатком;
- управління інтерфейсом користувача;
- підтримка графіки і вікон;
- підтримка багатозадачності;
- підтримка друку;
- підтримка обробки подій дотиків і жестів;

- об'єкти для відображення стандартних системних вікон і елементів управління;
- підтримка текстового і web-вмісту;
- підтримка «Вирізання», «Копіювання» та «Вставки»;
- підтримка анімації вмісту інтерфейсу користувача;
- інтеграція з іншими додатками системи за допомогою URL-схем;
- підтримка служби push-повідомлень Apple;
- підтримка додаткових можливостей для користувачів з обмеженими можливостями;
- планування і доставка локальних повідомлень;
- створення PDF-документів;
- підтримка особливих вікон введення інформації, які працюють як системна клавіатура;
- підтримка створення особливих текстових полів, які взаємодіють з системної клавіатурою.

Одним з ключових компонентів середовища розробки UIKit є клас UITableView. Він добре оптимізований для відображення нумерованих списків, як маленьких, так і великих. Table View можна редагувати і адаптувати під широкий діапазон сфер застосування, але основна ідея залишається незмінною - представлення нумерованого списку пунктів.

Клас UITableView відповідає тільки за подання даних у вигляді списку рядків. Відображені дані управляються об'єктом dataSource в table view. Об'єктом dataSource може бути будь-який об'єкт, який відповідає протоколу UITableViewDataSource. Об'єкт-джерело даних в table view часто є оператором уявлення, який керує поданням table view.

Подібним чином, table view відповідає тільки за розпізнавання дотиків в table view. Він не відповідає за реагування на ці події. До того ж до об'єкта dataSource в table view, у table view також є делегований об'єкт. Коли table view

розпізнає подію touch, він сповіщає делегований об'єкт про цю подію, і тоді вже в обов'язки делегованого об'єкта table view входить реагування на цю подію.

Маючи об'єкт, який управляє даними, а також делегований об'єкт, який обробляє події взаємодії з користувачем, table view може зосередити ресурси на представленні даних. В результаті ми отримуємо дуже продуктивний UIKit-компонент з можливість повторного використання, який відмінно можна порівняти з MVC-патерном (Model-View-Controller). Клас UITableView успадковує параметри з UIView, і отже, відповідає за відображення даних програми.

Об'єкт data source трохи схожий, але не ідентичний делегованому об'єкту. Делегованого об'єкту передається контроль над призначеним для користувача інтерфейсом делегованих об'єктом. Об'єкт data source має контроль над даними. Table view запитує у об'єкта data source дані, які потрібно відобразити. Це має на увазі той факт, що об'єкт джерела даних також відповідальний за управління даними, які він передає в table view.

Клас UITableView успадковує параметри з UIScrollView, субкласса UIView, який надає підтримку відображення контенту, за розміром більшого, ніж вікно програми. Екземпляр UITableView складається з рядків, де кожен рядок містить одну клітинку - екземпляр UITableViewCell або субкласів. На відміну від UITableView в OS X, екземпляр NSTableView в UITableView має на одну колонку більше. Вкладені набори даних і ієрархії можна відобразити за допомогою комбінації table view і контролерів навігації (UINavigationController).

2.6.2 Фреймворк MapKit

Фреймворк MapKit надає прокручуваний інтерфейс з картою, який можна вбудувати в існуючу ієрархію вікон мобільного застосування. Ця технологія забезпечує підтримку анотування карти, тобто наносити коментарі на карту,

виконує зворотне геокодування вибірок, щоб визначити placemarks для даного відображення координат.

MapKit дозволяє використовувати цю карту, щоб надати направлення або підсвітити пам'ятки. Додатки можуть програмно задавати атрибути карти або дозволити користувачеві управляти картою на свій розсуд. Також є можливість додавати примітки, картинки тощо до карти.

У iOS 4.0, базове вікно з картою отримало підтримку переміщуваних анотацій і власних шарів [14]. Переміщувані анотації дають можливість перепозиціонувати анотації як програмно, так і за допомогою маніпуляцій користувача, після того, як вони були вже додані на карту. Шари пропонують спосіб створення складних анотацій на карті, які складаються з більш ніж однієї точки. Наприклад, можна використовувати шари, щоб нанести інформацію про маршрути автобусів, вибіркові карти, межі парків або інформацію поверх карти (наприклад, радіолокаційні дані).

2.6.3 Фреймворк CoreData

CoreData – це технологія для управління моделлю даних в додатках MVC. Бібліотека призначена для використання в додатках, в яких модель даних вже високо структурована. Замість визначення структур даних програмно, використовується графічний інструмент Xcode для побудови схеми подання вашої моделі даних. Під час виконання екземпляри елементів моделі даних створюються, керуються і стають доступними за допомогою бібліотеки Core Data.

Керуючи моделлю даних мобільного застосування, бібліотека істотно скорочує кількість коду, яку необхідно написати. Фреймворк Core Data також надає наступні можливості:

- зберігання даних об'єктів в базі даних SQLite для оптимальної продуктивності;

- клас NSFetchedResultsController для управління результатами в полях-таблицях;
- управління скасуванням або повтором при редагуванні звичайного тексту;
- підтримка перевірки значень властивостей;
- підтримка передачі змін і забезпечення узгоджених зв'язків між об'єктами;
- підтримка угруповання, фільтрації і організації даних в пам'яті.

Core Data – гнучкий фреймворк для роботи з збереженими на пристрої даними. Звичайно ж є й інші варіанти зберігання даних, які можуть краще підійти при вирішенні певних завдань, але зараз Core Data дуже добре вписується в Cocoa Touch. Більшість деталей по роботі зі сховищем даних Core Data приховує, дозволяючи сконцентруватися на тому, що дійсно робить мобільний додаток унікальним і зручним у використанні.

Не дивлячись на те, що Core Data може зберігати дані в реляційній базі даних на зразок SQLite, Core Data не є СУБД [15]. Насправді, в якості сховища Core Data може взагалі не використовувати реляційні бази даних. Core Data не є чимось на зразок Hibernate, хоча і надає деякі можливості ORM [15]. Core Data швидше за все є оболонкою для роботи з даними, яка дозволяє працювати з сутностями і їх зв'язками, атрибутами, в тому вигляді, який нагадує роботу з об'єктним графом в звичайному об'єктно-орієнтованому програмуванні.

Core Data був впроваджений лише починаючи з Mac OS X 10.4 Tiger і iPhone SDK 3.0 [15].

2.6.4 Фреймворк CloudKit

CloudKit надає інтерфейси для переміщення даних між додатком і контейнерами iCloud. Даний фреймворк дозволяє зберігати існуючі дані мобільного додатку в хмарі, так що користувач може отримати доступ до нього

тільки на одному пристрої. Також можна зберігати дані в публічних місцях, де всі користувачі можуть отримати доступ до них.

CloudKit не є заміною для існуючих об'єктів даних мобільного застосування. Замість цього, CloudKit надає додаткові послуги для управління передачею даних з серверами iCloud. Він забезпечує мінімальну підтримку офлайнового кешування, CloudKit залежить від присутності контейнерів в мереж. Додатки завжди можуть зберігати дані в публічних місцях, який доступний для перегляду всім користувачам.

Записи знаходяться в центрі всіх транзакцій даних в CloudKit. Запис являє собою словник «ключ-значення», що представляє дані, які потрібно зберегти. CloudKit надає можливість додавати нові ключі і значення для записів в будь-який час, і дозволяє створювати посилання між пов'язаними записами для організації даних. Клас CKRecord визначає інтерфейси для управління вмістом записів. CloudKit також в значній мірі спирається на використання об'єктів NSOperation для управління асинхронною передачею даних [16].

2.6.5 Фреймворк Social

Фреймворк Social дозволяє інтегрувати додаток з підтримуваними сервісами соціальних мереж. Ця система забезпечує шаблон для створення HTTP-запитів. Social використовується тільки для iOS і забезпечує узагальнений інтерфейс для розміщення запитів від імені користувача [17].

Даний фреймворк використовується для:

- створення мережевого сеансу;
- отримання активності каналу користувача;
- створення нового запису у стрічкі новин соціальної мережі;
- зміни властивостей нового запису;
- прикріплення до запису файлів тощо;
- опублікування створеного запису в соціальній мережі користувача.

2.7 Технологія REST, HTTP запити на основі фреймворку Alamofire

REST – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований у 2000 році Роєм Філдінгом (Roy Fielding), одним із творців протоколу HTTP. Найвідомішою системою, побудованою переважно за архітектурою REST, є сучасна Всесвітня павутина.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад HTML, XML, JSON). Мережевий протокол (як і HTTP) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформацію про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабування системи і дозволяє їй еволюціонувати з новими вимогами.

Антиподом REST є підхід, заснований на виклику віддалених процедур (Remote Procedure Call, RPC). Підхід RPC дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом. При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути великою.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text TransferProtocol, протокол передачі гіпер-текстових документів. HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб-сторінками (зображення і застосунки), так і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

Alamofire є мережевою HTTP бібліотекою, яка написана на Swift. Вона використовує NSURLSession і Foundation URL Loading System, щоб забезпечити мережеві можливості першого класу в зручному Swift інтерфейс.

GET – запрошує вміст вказаного ресурсу. Запитаний ресурс може приймати параметри (наприклад, пошукова система може приймати як параметр шуканий рядок). Вони передаються в рядку URI. Згідно зі стандартом HTTP, запити типу GET вважаються ідемпотентними — багатократне повторення одного і того ж запиту GET повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити GET.

POST – передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу. Наприклад, в блогах відвідувачі зазвичай можуть вводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом POST, і він поміщає їх на сторінку. При цьому передані дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту. На відміну від методу GET, метод POST не вважається ідемпотентним, тобто багатократне повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожного відправлення коментаря з'являтиметься одна копія цього коментаря).

Приклад POST запиту:

```
Alamofire.request(.POST, "https://pa.bluebeard24.com/api/UserInfoApi/CheckUser",
parameters: parameters, encoding: .JSON).responseJSON(completionHandler: { response in
        switch response.result {
        case .Success:
            print(response.result.value!)
```

```
case .Failure:
    print(response.error)
```

2.8 Висковок

У розділі було розглянуто особливості архітектури платформи iOS та технологій розробки мобільних застосувань. Було наведено основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій, адже ці інструменти найкраще підходять для розробки мобільних додатків, які в свою чергу повинні мати простий, лаконічний, привабливий користувацький інтерфейс, швидку і зручну навігацію, високу швидкодію.

3. ПРИКЛАДИ МОБІЛЬНИХ ЗАТОСУВАНЬ, РОЗРОБЛЕНИХ НА ДАНИХ ТЕХНОЛОГІЯХ

3.1 Приклад використання AutoLayout

Дана технологія дозволяє розробляти інтерфейси користувача, який динамічно адаптується до різних форматів і розширень екранів пристроїв. Було розроблено мобільний додаток калькулятор «Калькулятор», який наглядно демонструє основні переваги AutoLayout (рис. 3.1):

Рисунок 3.1 – Розмір екранів

3.2 Приклад використання розглянутих фреймворків

Було розроблено мобільний додаток «Ресторани». Він дозволяє створювати список ресторанів, які користувач відвідав, а також планує відвідати, задавати їм певні властивості, такі як:

- ім’я;
- тип;

- номер телефону;
- фото (завантажується із пристрою);
- місце розташування (користувач може відразу побачити даний ресторан на карті).

Всі дані можна зберігати як на пристрої, так і в iCloud. Також користувач може оцінити даний ресторан, видалити його, зробити новий запис в стрічку новин соціальної мережі (Twitter або Facebook), додавши при цьому назву і фото ресторану, перехід у відповідну соціальну мережу. Додаток передбачає пошук серед записів по назві і типу ресторана.

3.2.1 Використання UITableView

Для відображення даних було використано клас UITableView (рис. 3.2). Для цього потрібно переопределити методи даного класу:

- numberOfSectionsInTableView(tableView: UITableView) -> Int – кількість секцій в таблиці;
- tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int – кількість рядків в секції;
- tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell – налаштовує саму комірку в залежності від типу.

Також було використано метод tableView(tableView: UITableView, editActionsForRowAtIndexPath indexPath: NSIndexPath) -> [UITableViewRowAction]?, який дозволяє створювати свайп по рядку таблички. Саме в цьому методі було реалізовано видалення ресторану із списку і створення нового запису у соціальній мережі (рис. 3.3).

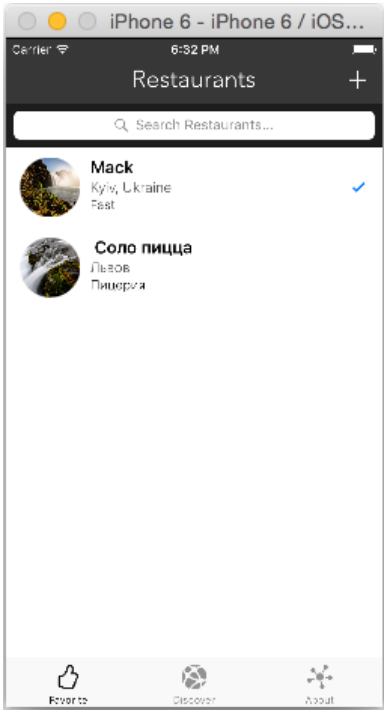

Рисунок 3.2 – Список ресторанів

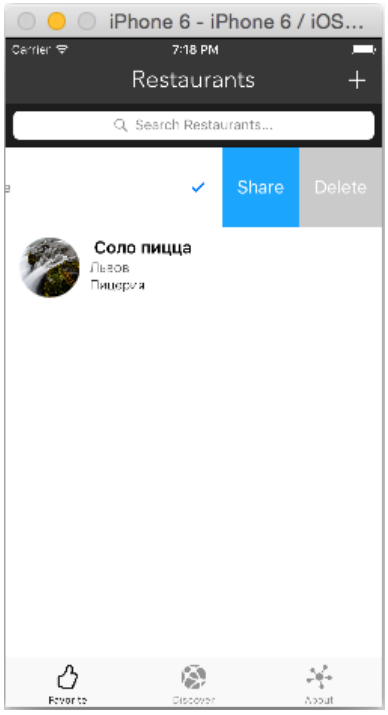

Рисунок 3.3 – Свайп по рядку

3.2.2 Використання CoreData

Він був використаний для додавання (рис. 3.4), відображення (див. рис. 3.2) і видалення ресторанів (див. рис. 3.3) із бази даних:

Рис. 3.4 – Додавання

Для цього потрібно зв’язати дані із бази даних за допомогою NSFetchedResultsController:

```
if let managedObjectContext = (UIApplication.sharedApplication().delegate as?
AppDelegate)?.managedObjectContext {
        fetchResultController = NSFetchedResultsController(fetchRequest: fetchRequest,
managedObjectContext: managedObjectContext, sectionNameKeyPath: nil, cacheName: nil)
        fetchResultController.delegate = self
        do {
           try fetchResultController.performFetch()
           restaurants = fetchResultController.fetchedObjects as! [Restaurant]
        } catch {
           print(error)
        }
    }
```

А також описати контроллер для управління NSFetchedResultsController:

```
func controllerWillChangeContent(controller: NSFetchedResultsController) {
    tableView.beginUpdates()
  }
 func controller(controller: NSFetchedResultsController, didChangeObject anObject: AnyObject,
atIndexPath indexPath: NSIndexPath?, forChangeType type: NSFetchedResultsChangeType,
newIndexPath: NSIndexPath?) {
    switch type {
    case .Insert:
       if let _newIndexPath = newIndexPath {
          tableView.insertRowsAtIndexPaths([_newIndexPath], withRowAnimation: .Fade)
       }
    case .Delete:
       if let _newIndexPath = newIndexPath {
          tableView.deleteRowsAtIndexPaths([_newIndexPath], withRowAnimation: .Fade)
       }
    case .Update:
       if let _newIndexPath = newIndexPath {
```

```
            tableView.reloadRowsAtIndexPaths([_newIndexPath], withRowAnimation: .Fade)
        }
    default:
        tableView.reloadData()
    }
    restaurants = controller.fetchedObjects as! [Restaurant]
}
func controllerDidChangeContent(controller: NSFetchedResultsController) {
    tableView.endUpdates()
}
```

3.2.3 Використання Segue

Segue – це спосіб зміни екранів в iOS. Один із найбільш популярних різновидів – це push (з версії iOS 8 – show). Push segue завжди зміщує поточний вид справа наліво. Головна особливість Segue – можливість передачі даних між двома UIViewController (рис. 3.5). Для цього було використано наступний метод:

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "showDetail" {
        if let indexPath = tableView.indexPathForSelectedRow {
            let destinationController = segue.destinationViewController as! DetailViewController
            destinationController.restaurant = (searchController.active) ?
searchResult[indexPath.row] : restaurants[indexPath.row]
        }
    }
}
```

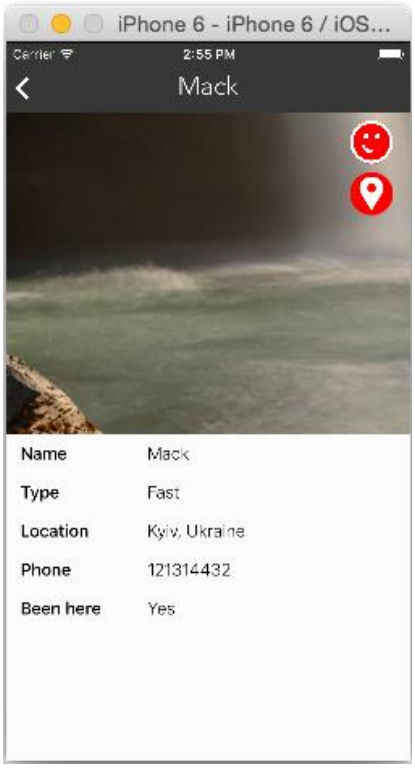

Рисунок 3.5 – Детальна інформація про ресторан

3.2.4 Використання MapKit

Даний фреймворк був використаний для відображення на карті місцезнаходження відповідного ресторану (рис. 3.6). Для цього потрібно задати геокодер, для того, щоб знати певне місце на карті:

```
let geoCoder = CLGeocoder()
    geoCoder.geocodeAddressString(restaurant.location, completionHandler: { placemarks,
error in
        if error != nil {
            print(error)
            return
        }
            if let placemarks = placemarks {
            let placemark = placemarks[0]
            let annotation = MKPointAnnotation()
            annotation.title = self.restaurant.name
            annotation.subtitle = self.restaurant.type
```

```
            if let location = placemark.location {
                annotation.coordinate = location.coordinate
                self.mapView.showAnnotations([annotation], animated: true)
                self.mapView.selectAnnotation(annotation, animated: true)
            }
        }
    })
```

А також задати використати метод mapView(mapView: MKMapView, viewForAnnotation annotation: MKAnnotation) -> MKAnnotationView? Для відображення місцезнаходження даного ресторану.

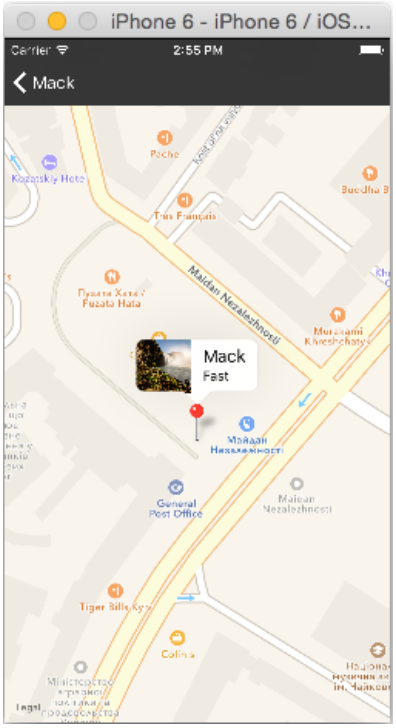

Рисунок 3.6 – Місцезнаходження ресторану

3.2.5 Використання анімації

Користувач може зробити оцінку для певного ресторану. Для відображення елементів оцінки було використано базову анімацію (рис 3.7):

```
override func viewDidAppear(animated: Bool) {
    UIView.animateWithDuration(1.0, delay: 0, options: [], animations: {
        self.stack.transform = CGAffineTransformIdentity
```

```
        }, completion: nil)
    }
```

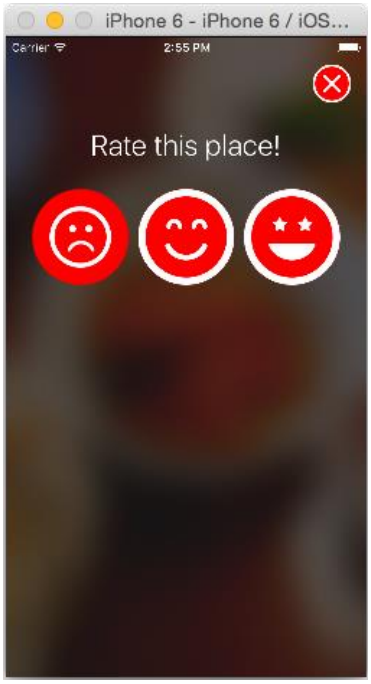

Рисунок 3.7 – Оцінка ресторану

3.2.6 Використання CloudKit

Даний фреймворк дає можливість зв’язувати мобільний додаток з iCloud (рис. 3.8). Для цього потрібно отримати дані з сервера:

```
func getRecordsFromCloud() {
    var newRestaurantsFromCloud:[CKRecord] = []
    let cloudContainer = CKContainer.defaultContainer()
    let publicDatabase = cloudContainer.publicCloudDatabase
    let predicate = NSPredicate(value: true)
    let query = CKQuery(recordType: "Restaurant", predicate: predicate)
    query.sortDescriptors = [NSSortDescriptor(key: "creationDate", ascending: false)]
    let queryOperation = CKQueryOperation(query: query)
    queryOperation.desiredKeys = ["name"]
    queryOperation.queuePriority = .VeryHigh
    queryOperation.resultsLimit = 50
    queryOperation.recordFetchedBlock = { (record: CKRecord!) -> Void in
```

```
            if let restaurantRecord = record {
                newRestaurantsFromCloud.append(restaurantRecord) //Заполняем созданный нами
массив
            }
        }
        queryOperation.queryCompletionBlock = { (cursor: CKQueryCursor?, error: NSError?) ->
Void in
            if error != nil {
                print("Failed to get data from iCloud")
                return
            }
            print("Success")
            NSOperationQueue.mainQueue().addOperationWithBlock() {
                self.restaurants = newRestaurantsFromCloud
                self.tableView.reloadData()
                self.spinner.stopAnimating()
                self.refreshControl?.endRefreshing()
            }
        }
            publicDatabase.addOperation(queryOperation)
    }
```

Для запису в iCloud потрібно:

```
let record = CKRecord(recordType: "Restaurant")
let publicDatabase = CKContainer.defaultContainer().publicCloudDatabase
publicDatabase.saveRecord(record, completionHandler: { (record: CKRecord?, error: NSError?)
-> Void in
            do {
                try NSFileManager.defaultManager().removeItemAtPath(imageFilePath)
            } catch {
                print("saving error")
            }
        })
```

Рисунок 3.8 – Відображення даних із iCloud

Рисунок 3.9 – Pull to refresh

Для відображення картинок по мірі їх завантаження було використано кешування NSCache. Також для оновлення інформації було використано «pull to refresh» (див. рис. 3.9):

```
refreshControl = UIRefreshControl()
    refreshControl?.backgroundColor = UIColor.whiteColor()
    refreshControl?.tintColor = UIColor.grayColor()
    refreshControl?.addTarget(self, action: "getRecordsFromCloud", forControlEvents: UIControlEvents.ValueChanged)
```

3.2.7 Використання Social

Даний фрейсворк надає можливість робити нові записи у стрічкі нових соціальних мереж Twitter і Facebook (рис. 3.10).

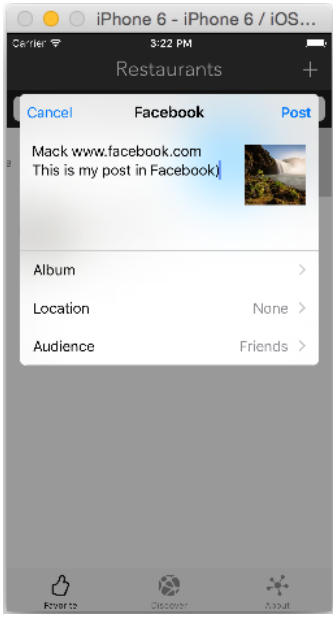

Рисунок 3.10 – Створення запису

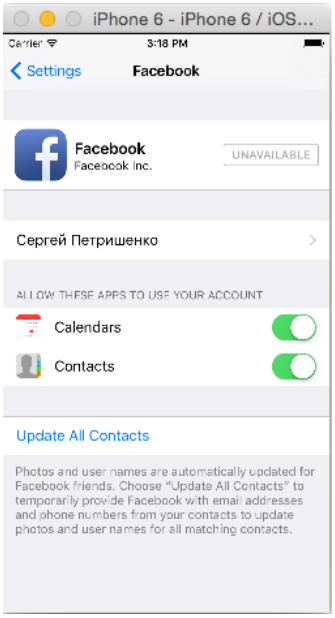

Рисунок 3.11 – Авторизація

Для його використання необхідно авторизуватися у соціальних мережах (див. рис. 3.11). В іншому випадку буде виведено повідомлення про помилку. Для переходу до створення нових записів у соціальних мережах Facebook і Twitter (рис. 3.12) і виводу помилки входу (рис. 3.13) було використано AlertView.

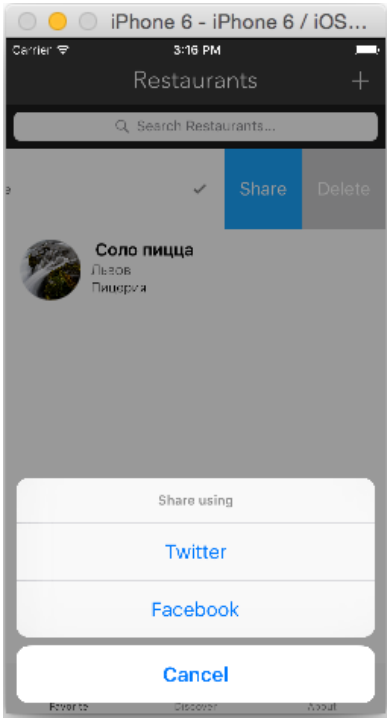

Рисунок 3.12 – Перехід до створення запису

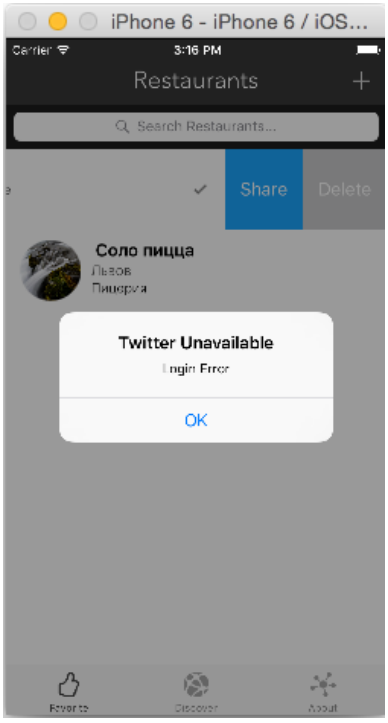

Рисунок 3.13 – Помилка авторизації

Приклад використання для Twitter:

```
let shareMenu = UIAlertController(title: nil, message: "Share using", preferredStyle:
.ActionSheet)
let twitterAction = UIAlertAction(title: "Twitter", style: UIAlertActionStyle.Default,
handler: { (action) -> Void in
    guard SLComposeViewController.isAvailableForServiceType(SLServiceTypeTwitter)
else {
        let alertMessage = UIAlertController(title: "Twitter Unavailable", message: "Login
Error", preferredStyle: .Alert)
        alertMessage.addAction(UIAlertAction(title: "OK", style: .Default, handler: nil))
        self.presentViewController(alertMessage, animated: true, completion: nil)
        return
    }
    let tweetComposer = SLComposeViewController(forServiceType:
SLServiceTypeTwitter)
    tweetComposer.setInitialText(self.restaurantNames[indexPath.row])
    tweetComposer.addImage(UIImage(named: self.restaurantImages[indexPath.row]))
    self.presentViewController(tweetComposer, animated: true, completion: nil)
})
```

3.2.8 Використання веб-переглядів

Розроблений додаток має можливість переходу по ссилці на основні соціальні мережі (рис. 3.14):

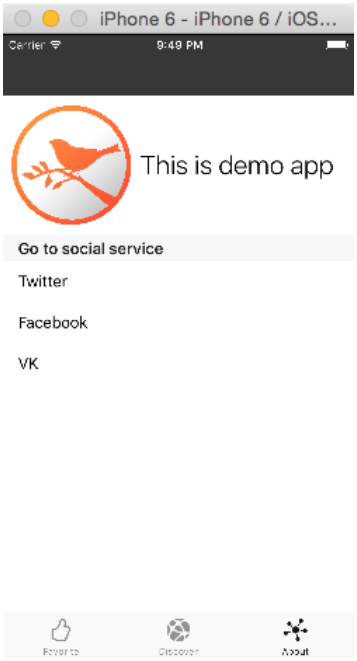

Рисунок 3.14 – Перехід в соціальні мережі

Для веб-перегляду використовується 3 підхода:

- браузер Safari (рис. 3.15):

Для переходу в браузер потрібно викликати наступний метод: UIApplication.sharedApplication().openURL(url).

- UIWebView (рис. 3.16):

Для цього потрібно створити окремий контроллер і виконати перехід по Segue. В цьому контроллері викликати наступний метод:

```
let request = NSURLRequest(URL: url)
webView.loadRequest(request)
```

- SafariViewController (рис. 3.17):

Для цього потрібно викликати наступний метод:

```
let safariView = SFSafariViewController(URL: url, entersReaderIfAvailable: true)
presentViewController(safariView, animated: true, completion: nil)
```

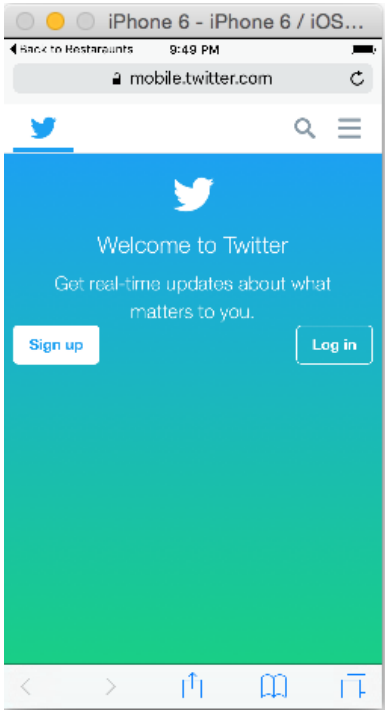

Рисунок 3.15 – Twitter

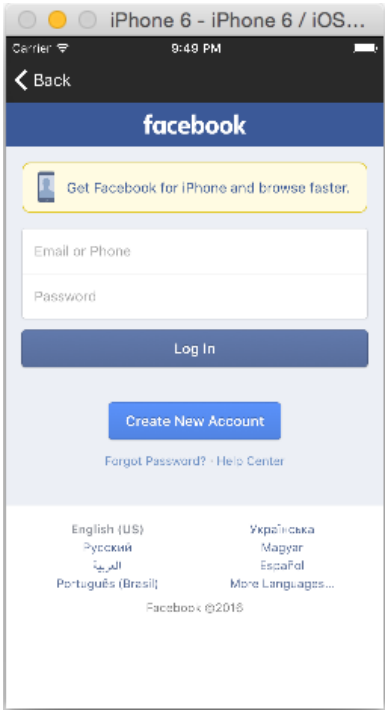

Рисунок 3.16 – Facebook

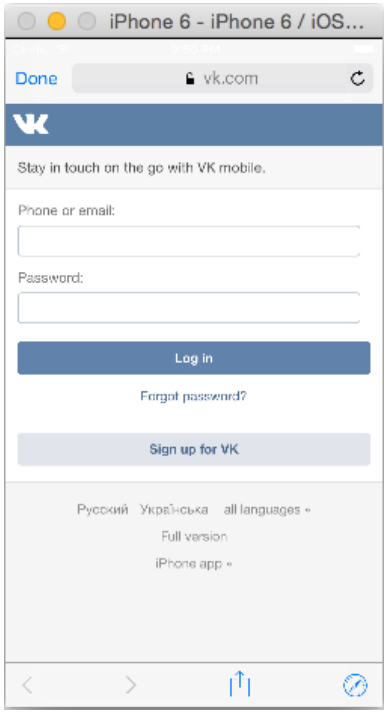

Рисунок 3.17 – Вконтакті

3.3 Приклад використання REST з Alamofire

Даний приклад демонструє роботу клієн-сереверного додатку. Спочатку на сервер відправляється запит типу POST:

```
let parameters = [
        "login": "\(login!)",
        "password": "\(password!)"
    ]
    Alamofire.request(.POST, "https://pa.bluebeard24.com/api/UserInfoApi/CheckUser", parameters: parameters, encoding: .JSON).validate().responseObject(completionHandler: { (response: Response<LoginServerResponse, NSError>) in
        switch response.result {
        case .Success(let name):
            let userKey = NSUserDefaults.standardUserDefaults()
            userKey.setObject(name.idUser, forKey: "userNameId")
            userKey.synchronize()
```

```
            let checkUser = name
            block(result: checkUser)
        case .Failure:
            print("Error")
            block(result: nil)
        }
    })
```

З сервера приходять дані в форматі JSON. Для розпарсування даних було використано фреймворк AlamofireObjectMapper (рис. 3.18). Він перетворює ці дані з формату JSON в об’єкти swift:

```
func mapping(map: Map) {
    avatarUrl <- map["avatarUrl"]
    firstName <- map["firstName"]
    lastName <- map["lastName"]
    userName <- map["userName"]
}
```

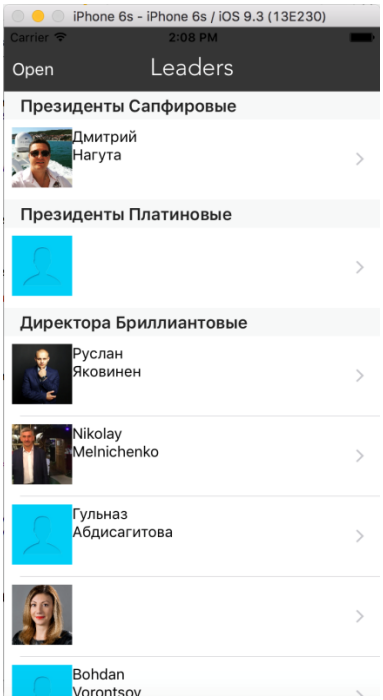

Рисунок 3.18 – Виведення даних

3.4 Аналіз розроблених продуктів

Головною метою дипломної роботи є демонтсрації основних особливостей розробки мобільних застосувань для операційної системи iOS, використовуючи мову програмування Swift. Розроблені приклади повністю відповідають вимогам сучасних мобільних додатків, які розроблені з урахування кращих практик і специфічних архітектурних рішень, які дають можливість забезпечити дуже високу продуктивність мобільного додатку.

3.5 Оцінка використаних та досліджених підходів

В даній роботі були досліджені основні засоби для розробки сучасного мобільного застосування для платформи iOS. Розглянуті технології найкраще підходять для вирішення даної задачі, бо вони відносяться до вищого рівня архітектури iOS і краще забезпечують об'єктно-орієнтовані абстракції для низькорівневих структур. Ці абстракції істотно полегшують написання коду, тому що вони зменшують обсяг коду, який необхідно написати і приховують досить складні функції, такі як сокети і потоки.

3.6 Основні переваги та недоліки створеного рішення

В даній роботі було розроблено приклади мобільних застосувань.

Переваги:

- мають простий, лаконічний, привабливий користувацький інтерфейс;
- мають швидку і зручну навігацію;
- мають високу швидкодію.

Недоліки:

- використання тільки під одну платформу (iOS);

- висока ціна прав розробника при завантаженні на AppStore.

3.7 Висновок

У розділі було розроблено мобільні додатки з урахуванням всіх специфік та можливостей платформи iOS, використовуючи розглянуті технології, проведено аналіз розроблених продуктів, зроблено оцінку використаних та досліджених підходів, визначено основні переваги та недоліки створеного рішення.

4. ЕКОНОМІЧНА ЧАСТИНА

4.1 Функціонально-вартісний аналіз програмного продукту

У даній роботі проводиться оцінка основних характеристик програмного продукту, що полягає в створенні мобільного застосування для операційної системи iOS, використовуючи мову програмування Swift.

Для техніко-економічного аналізу програмного продукту буде використаний метод функціонально-вартісного аналізу (ФВА). Основою ФВА є функціональний підхід, згідно з яким об'єктом аналізу тобто не сам продукт, а функції, які він виконує. ФВА проводиться в два етапи: функціональний аналіз і вартісний аналіз.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом [18]:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на смартфонах Apple Iphone із стандартним набором компонент;

– забезпечувати високу швидкість роботи з сервісами у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

4.3 Обґрунтування функцій програмного продукту

Головною метою програмного продукту є наглядна демонстрація головних переваг і основних можливостей нової мови програмування від компанії Apple – Swift. Ґрунтуючись на цьому, виділимо основні функції:

F1 – вибір найбільш зручної мови програмування;

F2 – вибір технології для створення інтерфейсу користувача;

F3 – вибір середовища розробки;

Кожна з основних функцій може мати кілька варіантів рішення:

для F1:

а) мова програмування C#;

б) мова програмування Ojective-C;

в) мова програмування Swift;

для F2:

а) інтерфейс створений без використання технології Autolayout;

б) інтерфейс створений з використання технології Autolayout;

для F3:

а) середовище розробки Xcode;

б) середовище розробки Xamarin;

в) середовище розробки Appcelerator;

За розглянутими варіантами будуємо морфологічну карту (рис.4.1). На основі цієї карти побудуємо позитивно-негативну матрицю (табл.4.1).

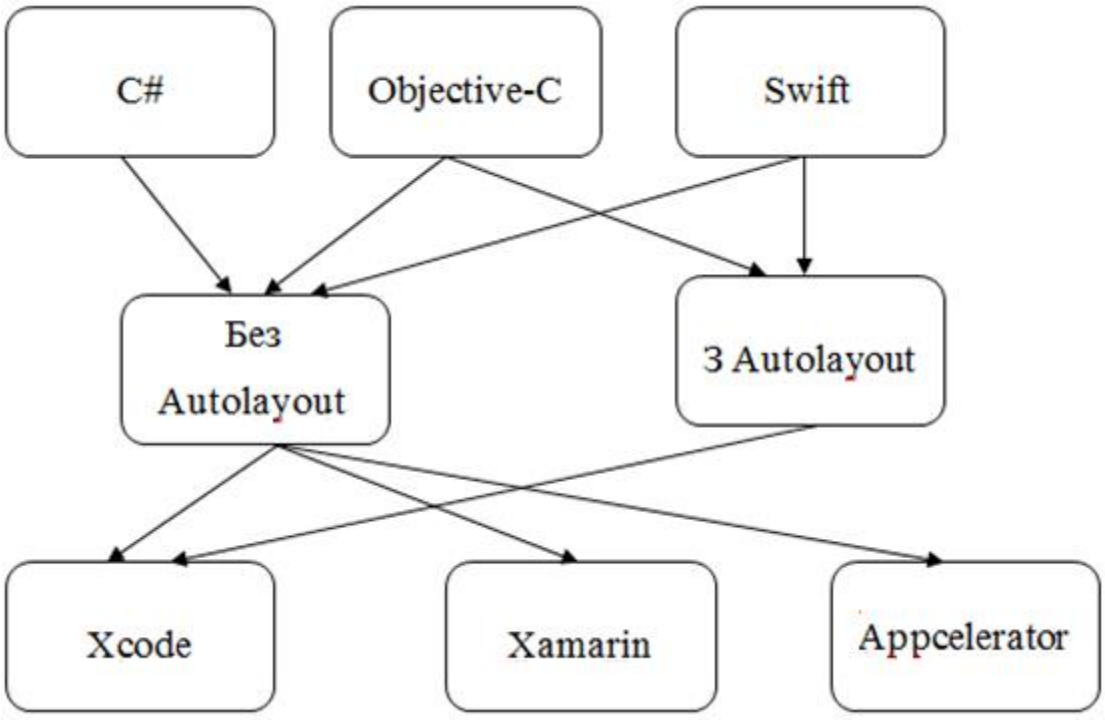

Рисунок 4.1 – Морфологічна карта

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	а)	Багатофункціональна	Низька швидкодія
	б)	Має високу швидкодію	Складність у програмуванні
	в)	Легкість у програмуванні	Відсутня можливість використання на інших платформах
F2	а)	Легкість у використанні	Немає масштабованості
	б)	Швидкодія, масштабованість	Складність у використанні
F3	а)	Висока швидкодія, легкість у використанні	Використання тільки під Mac OS
	б)	Використання під будь-яку платформу	Складність в налаштуванні
	в)	Використання під будь-яку платформу	Складність у використанні

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам.

F1: Оскільки програмний продукт має бути зручним для повторного використання, то використовуємо варіант в) як єдиний можливий.

F2: Вибір технології для створення інтерфейсу користувача не обмежується.

F3: Найкращим середовищем розробки є варіант а), оскільки він є безкоштовним та легким у використанні.

У зв’язку із оглядом основних функцій ПП наведеним вище, будемо розглядати наступні варіанти реалізації:

А: F1в) – F2а) – F3а)

Б: F1в) – F2б) – F3а)

4.4 Обґрунтування системи параметрів програмного продукту

4.4.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

Таблиця 4.2 – Визначення параметрів продукту

X1 – швидкість виконання	*X2* – час виконання	*X3* – ресурсоємність	*X4* – потенційний об’єм програмного коду
Відображає швидкість виконання по часу в залежності від мови програмування	Відображає час виконання	Відображає навантаженість на систему, спричинену роботою програмного продукту	Відображає розмір коду, який необхідно написати розробнику

4.4.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП.

Таблиця 4.3 – Основні параметри продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметру		
			Гірші	Середні	Кращі
Швидкість виконання	X1	мс	1000	400	100
Час виконання	X2	с	10	5	2
Ресурсоємність	X3	%	100	80	20
Об'єм програмного коду	X4	Кількість рядків коду	3000	2000	1000

За даними табл. 4.3 побудуємо графіки залежності бальної оцінки параметра від його основного значення (рис. 4.2–4.5).

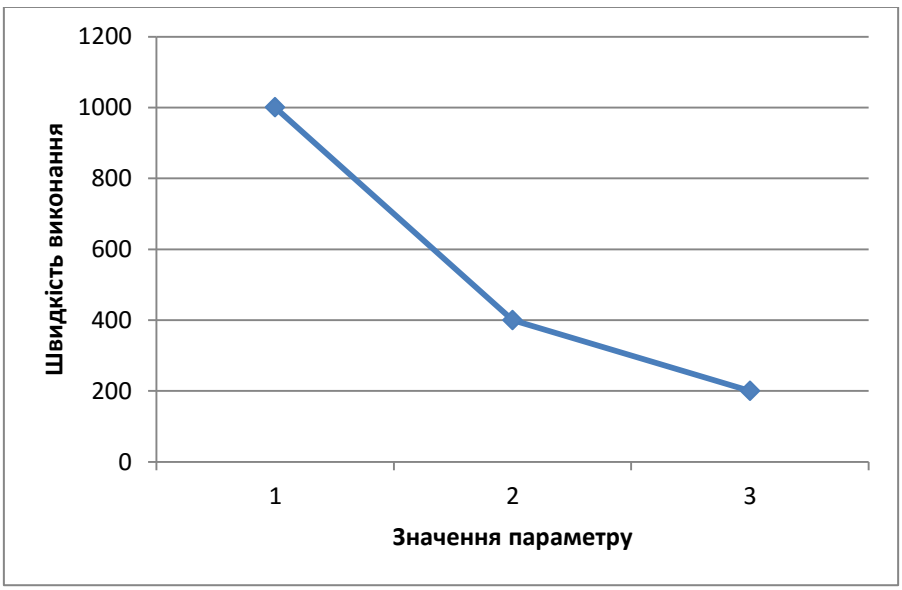

Рисунок 4.2 – Швидкість виконання

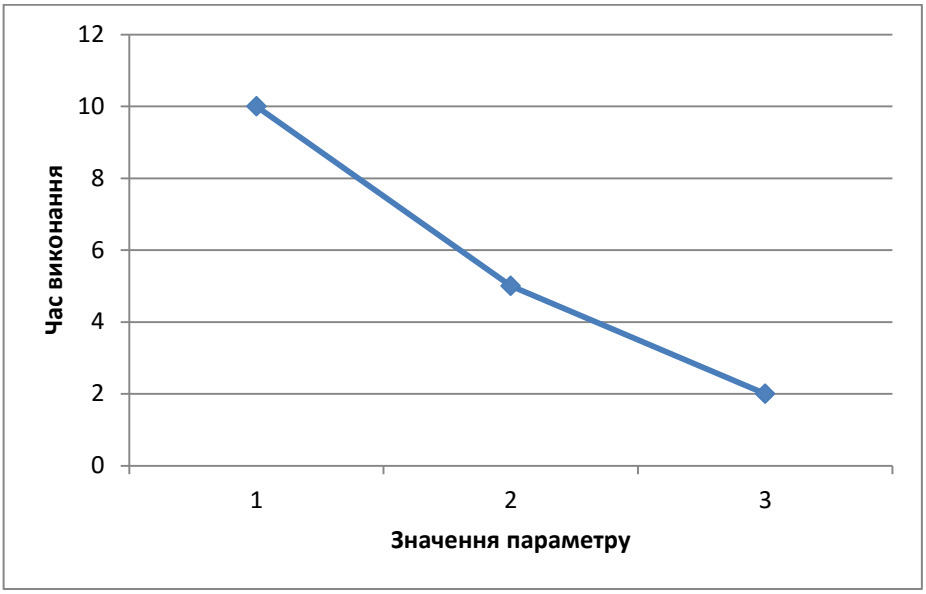

Рисунок 4.3 – Час виконання

Рисунок 4.4 – Ресурсоємність

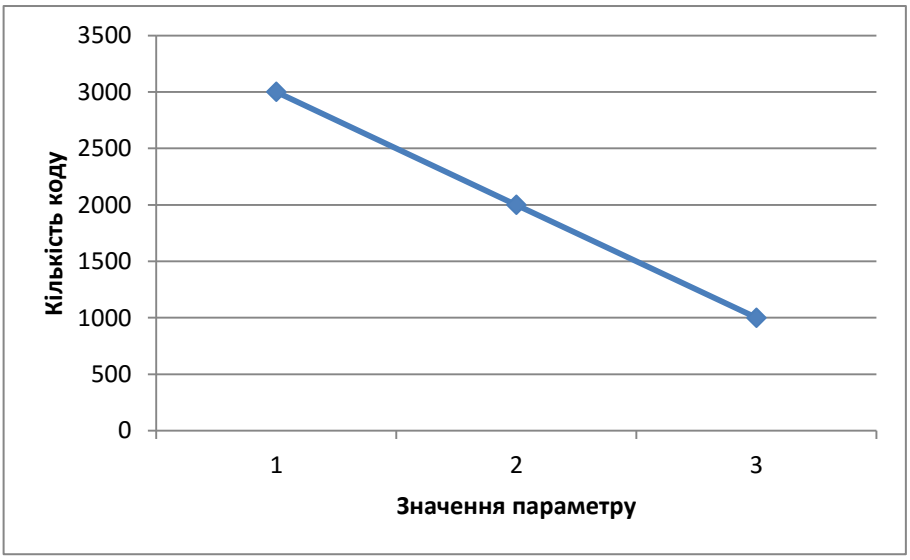

Рисунок 4.5 – Об'єм програмного коду

4.4.3 Аналіз експертного оцінювання параметрів

Важливість кожного параметра в загальній кількості розглянутих під час оцінювання параметрів, знаходять за методом попарного порівняння. Оцінювання проводить експертна комісія із семи осіб. Визначення коефіцієнта важливості передбачує:

- визначення рівня важливості параметра через присвоєння різних рангів;
- перевірити придатність експертних оцінок у подальшому використанні;
- визначити оцінки попарного пріоритету параметрів;
- обробити результати і знайти коефіцієнт важливості.

Після детального обговорення й аналізу кожний експерт оцінює рівень важливості, присвоюючи їм ранг. Результат експертного ранжування наведено в таблицю 4.4.

Таблиця 4.4 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці вимірy	Ранг параметра за оцінкою експерта							Сума рангів Ri	Відхилення Δi	Δi2
			1	2	3	4	5	6	7			
X1	Швидкість виконання	мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Час завантаження і обробки даних	с	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Ресурсоємність	%	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Об’єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
Разом			15	15	15	15	15	15	15	105	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів за формулою (4.1):

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \qquad (4.1)$$

де N – число експертів;

n – кількість параметрів.

б) середня сума рангів за формулою (4.2):

$$T = \frac{1}{n} R_{ij} = 26{,}25. \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів за вормулою (4.3):

$$\Delta_i = R_i - T \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення за формулою (4.4):

$$S = \sum_{i=1}^{N} \Delta_i^2 = 420{,}75. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості за формулою (4.5):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 420{,}75}{7^2(5^3-5)} = 1{,}03 > W_k = 0{,}67 \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.5:

Таблиця 4.5 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі (4.6):

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j, \\ 1{,}0 \text{ при } X_i = X_j, \\ 0{,}5 \text{ при } X_i < X_j. \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за формулою (4.7) :

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{ де } b_i = \sum_{i=1}^{N} a_{ij} \quad (4.7)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за формулою (4.8):

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{ де } b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.8)$$

Як видно з таблиці, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.6 – Розрахунок вагомості параметрів ПП

Параметри Xi	Параметри Xj				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	K_ві	b_i^1	K_ві^1	b_i^2	K_ві^2
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (швидкість виконання) та X4 (кількість строк) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (ресурсоємність) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 20 % або варіанту б) 80%.

Абсолютне значення параметра X2 (час виконання) також обрано на основі експертних знань: а) 2 с; Б) 5с.

4.6 Розрахунок показників якості варіантів реалізації

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою (4.9):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \,, \qquad (4.9)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Розрахунок показників рівня якості варіантів реалізації основних функцій ПП:

Таблиця 4.7 – Рівень якості варіантів реалізації

Параметри	Реалізації функцій	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
X1(F1)	А, Б	400	6.25	0,215	1.344
X2(F3)	А	2	2.8	0,283	0.729
	Б	5	8	0,283	2.264
X3(F3)	А	20%	7.44	0,348	2.589
	Б	80%	1.90	0,348	0.661
X4(F2)	А, Б	2000	6.5	0,154	1.001

За даними з таблиці за формулою (4.10)

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.10)$$

визначаємо рівень якості кожного з варіантів:

А: $K_1 = 1.344 + 0.729 + 2.589 + 1.001 = 5.663$
Б: $K_2 = 1.344 + 2.264 + 0.661 + 1.001 = 5.27$

Як видно з розрахунків, кращим є варіант А, для якого коефіцієнт технічного рівня має найбільше значення.

4.7 Економічний аналіз варіантів розробки програмного продукту

Розрахуємо трудомісткість розробки нашого ПП за різних умов реалізації.

Норми часу беремо відповідно до мов програмування, тому коефіцієнт $К_м = 1$. Якщо для розробки ПП використовують стандартні модулі чи пакети прикладних програм, стандартні програми, норми часу коригуються за допомогою коефіцієнта $К_{ст} =$ 0,6–0,8 [19]. У нашому випадку $К_{ст} = 0,7$. Якщо розробляють стандартний ПП, норму часу потрібно коригувати за допомогою коефіцієнта $К_{ст.м} = 1,2–1,6$. У нашому випадку $К_{ст.м} = 1,6$. У загальному випадку трудомісткість ПП розраховуємо за формулою (4.11):

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.11)$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Таблиця 4.8 – Трудомісткість роботи

Основні функції	Варіант реалізації функції	Рівень складності	Ступінь новизни
F1 (x1)	б	1	Б
F2 (x2)	б	2	В
F3 (x3)	а)	1	Б
	б)	1	А

Таблиця 4.9 – Трудомісткість ПП

Основні функції	Варіант реалізації функції	Загальна норма, $T_{р(б)}$	Коефіцієнт, $К_п$	Коефіцієнт, $К_{сл}$	Коефіцієнт, $К_{ст}$	Коефіцієнт, $К_{ст.м}$	Розрахункова працездатність, T (чол./днів)
F1 (x1)	б	52	1,021	1,00	0,7	1,6	59,46
F2 (x2)	б	52	0,72	1,00	0,7	1,6	65,35
F3 (x3)	а)	52	2,02	1,08	0,7	1,6	127,06
	б)	52	2,84	1,08	0,7	1,6	178,63

На основі даних з таблиці 4.9 визначаємо трудомісткість кожного з варіантів реалізації ПП:

$T_I = (59,46 + 65,35 + 127,06) \cdot 8 = 2014,96$ людино-годин;

$T_{II} = (59,46 + 65,35 + 178,63) \cdot 8 = 2427,55$ людино-годин;

Більш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 8 000 грн. Визначимо зарплату за годину за формулою (4.12):

$$С_ч = \frac{М}{T_m \cdot t} грн., \quad (4.12)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$С_ч = \frac{8000}{21 \cdot 8} = 47{,}62 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою (4.13):

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.13)$$

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 47,62· 2014,96 · 1.2 = 115 142,87 грн.

II. $С_{ЗП}$ = 47,62· 2427,55 · 1.2 = 138 719,92 грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0{,}22$ = 115 142,87 · 0,22 = 25 331,43 грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0{,}22$ = 138 719,92 · 0,22 = 30 518,38 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 8000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З$ = 12 · 8000· 0,2 = 19200 грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З)$ = 19200· (1 + 0.2)=23040 грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0{,}22$= 23040· 0,22= 5 068,8 грн.

Амортизаційні відрахування розраховуємо при амортизації 50% та вартості ЕОМ – 20000 грн за формулою (4.14):

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,50 \cdot 20000 = 11500 \text{ грн.}, \quad (4.14)$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою (4.15):

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 20000 \cdot 0,05 = 1150 \text{ грн.}, \quad (4.15)$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою (4.16):

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = \quad (4.16)$$

$$= (365 – 104 – 8 – 16) \cdot 8 \cdot 0,9 = 1706,4 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою (4.17):

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = \quad (4.17)$$

$$= 1706,4 \cdot 0,5 \cdot 0,2 \cdot 2,02 = 344,69 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13\ 400$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС} = 23040 + 5\ 068,8 + 11500 + 1150 + 344,69 + 13\ 400 = 54503,49$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 54503,49 / 1706,4 = 31,8$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$С_М = С_{М\text{-}Г} \cdot Т$

I. $С_М = 31,8 \cdot 2014,96 = 64\ 075,73$ грн.;

II. $С_М = 31,8 \cdot 2427,55 = 77\ 196,09$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 115\ 142,87 \cdot 0,67 = 77\ 145,72$ грн.;

II. $С_Н = 138\ 719,92 \cdot 0,67 = 92\ 942,35$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП} = 115\ 142,87 + 25\ 331,43 + 64\ 075,73 + 77\ 145,72 = 281\ 695,75$ грн.;

II. $С_{ПП} = 138\ 719,92 + 30\ 518,38 + 77\ 196,09 + 92\ 942,35 = 339\ 376,74$ грн.;

4.8 Вибір кращого варіанта програмного продукту техніко-економічного рівня

Коефіцієнт техніко-економічного рівня розраховують за формулою:

- $К_{тер1} = 5,663 / 281\ 695,75 = 2,01 • 10^{-5}$;
- $К_{тер2} = 5.27 / 339\ 376,74 = 1,55 • 10^{-5}$;

Отже, найбільш ефективним є перший варіант реалізації програми.

4.9 Висновок

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту.

У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 2,01 • 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Мова програмування – Swift;
- Програмне середовище – Xcode;
- Візуалізація інтерфейсу користувача за допомогою технології Autolayout.

ВИСНОВКИ

Тема розробки мобільних застосувань під платформу iOS є досить цікавою і представляє широке поле для подальших досліджень в галузі розробки мобільного ПЗ. Специфіка даного сегмента полягає в тому, що розробка iOS-додатків повинна проводитися з урахуванням особливостей мобільних пристроїв: відмінностями інтерфейсу, іншим розміром екрану, сенсорним управлінням. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку.

В даній роботі було наведено основні інструменти для розробки мобільного програмного забезпечення під платформу iOS.

Розглянуто основні засоби розробки, доведено і обгрунтовано правильність вибору цих засобів для проектування. На основі проведеного аналізу встановлено, що найкращим середовищем розробки є Xcode, а мовою програмування – Swift. Адже вони найбільше підходять для розробки додатків під платформу iOS.

В результаті виконання даної дипломної роботи було досліджено основні особливості мови програмування Swift, на її основі були розроблені мобільні застосування для операційної системи iOS. Було розглянуто основні переваги Swift перед Objective-C такі як простота у використанні, більш простіший у використанні синтаксис, швидкість виконання додатків вища, що призводить до зменшення кількості часу на розробку. Swift доцільно використовувати у тих випадках, коли проект створюється з нуля, адже на даний момент в світі дуже багато додатків, які написані на Objective-C, і їх краще підтримувати саме на цій мові, а не переписувати на Swift.

Також було розглянуто основні особливості архітектури платформи iOS та технології розробки мобільних застосувань, визначено основні переваги та недоліки даної платформи. Було наведено основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій,

адже ці інструменти найкраще підходять для розробки мобільних додатків, використовуючи саме Swift, які в свою чергу повинні мати простий, лаконічний, привабливий користувацький інтерфейс, швидку і зручну навігацію, високу швидкодію. Розглянуті фреймворки і технології відносяться до вищого рівня архітектури, тому вони забезпечують об'єктно-орієнтовані абстракції для низькорівневих структур. Ці абстракції істотно полегшують написання коду, тому що вони зменшують обсяг коду, який необхідно написати і приховують досить складні функції, такі як сокети і потоки.

Результатом даної роботи є підтверджений відповідним аналізом висновок про те, що Swift найкраще підходе для розробки сучасного мобільного ПЗ під платформу iOS. Прикладом використання є додатки, які наглядно демонструють основні переваги Swift.

Також було проведено аналіз розроблених програмних продуктів, зроблено оцінку використаних та досліджених підходів, визначено основні переваги та недоліки створеного рішення.

В цілому ідея розробки мобільного ПЗ є досить перспективною, адже протягом останніх років показник, що характеризує рівень попиту на мобільні пристрої, постійно зростає. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна і доцільна. Головне грамотно оцінити, для кого і навіщо створюється ПЗ. Тільки корисна розробка отримає гідне визнання з боку користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт Apple Xcode. – Режим доступу: https://developer.apple.com/xcode/. – Дата доступу 25.05.2016.
2. Офіційний сайт AppCode. – Режим доступу: https://www.jetbrains.com/objc/. – Дата доступу 21.05.2016.
3. Офіційний сайт Xamarin. – Режим доступу: https://www.xamarin.com/. – Дата доступу 19.05.2016.
4. Офіційний сайт Visual Studio. – Режим доступу: https://www.visualstudio.com/. – Дата доступу 03.05.2016.
5. Офіційний сайт Appcelerator. – Режим доступу: http://www.appcelerator.com/. – Дата доступу 02.06.2016.
6. Офіційний сайт Stack Overflow. – Режим доступу: http://stackoverflow.com/research/developer-survey-2015#tech-super. – Дата доступу 13.03.2016.
7. ANON The Swift Programming Language (Swift 2.1) / ANON – Cupertino: Apple Inc., 2014. – 528 p.
8. Vandad Nahavandipoor iOS 8 Swift Programming Cookbook / Vandad Nahavandipoor – Boston: O'Reilly Media., 2014. – 902 p.
9. Офіційна документація мови програмування Swift. – Режим доступу: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/. – Дата доступу 12.05.2016.
10. Innovations in Science and Technology: the XVI All-Ukrainian R&D Students Conference Proceeding, (Kyiv, April 18, 2016) / National Technical University of Ukraine “Kyiv Polytechnic Institute”. – Part II. – Kyiv, 2016. – 116 p.
11. Офіційний сайт блогу Primate Labs. – Режим доступу: http://www.primatelabs.com/blog/2014/12/swift-performance/. – Дата доступу 20.04.2016.

12. Офіційний сайт блогу Primate Labs. – Режим доступу: http://www.primatelabs.com/blog/2015/02/swift-performance-updated/. – Дата доступу 23.04.2016.
13. Офіційна документація фреймворку UIKit. – Режим доступу: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/. – Дата доступу 02.05.2016.
14. Офіційна документація фреймворку MapKit. – Режим доступу: https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/. – Дата доступу 26.04.2016.
15. Офіційна документація фреймворку CoreData. – Режим доступу: https://developer.apple.com/library/watchos/documentation/Cocoa/Conceptual/CoreData/index.html. – Дата доступу 27.05.2016.
16. Офіційна документація фреймворку CloudKit. – Режим доступу: https://developer.apple.com/library/ios/documentation/CloudKit/Reference/CloudKit_Framework_Reference/. – Дата доступу 14.05.2016.
17. Офіційна документація фреймворку Social. – Режим доступу: https://developer.apple.com/library/ios/documentation/Social/Reference/Social_Framework/. – Дата доступу 27.05.2016.
18. Яловий Г.К. Економіка та організація виробництва / Яловий Г.К., Пашін В.П., Сичов В.С. – К.: "Політехніка", 2004. – 80 с.
19. Богданюк В.Є. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів / Богданюк В.Є., Березовський К.В., Пашін В.П. – К.: НТУУ "КПІ", 1999. – 66 с.