МІНІСТЕРСТОВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«____» _____ 21___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: «On-line сервіс релакс-центра для платформи IOS»

Виконавець: студентка УС-412 Бурова Катерина Олексіївна
(студент, група, прізвище, ім'я, по батькові)

Керівник: к. т. н., доцент Климова Асія Сабирівна
(науковій ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П. _____
(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри

А.С. Савченко
“_____”_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Бурова Катерина Олексіївна

(прізвище, ім’я, по батькові)

1.Тема проекту: «On-line сервіс релакс-центра для платформи IOS» затверджена наказом ректора № 636/ст. від «22» квітня 2021 р.

2. Термін виконання роботи: з 10.03.2021 по 14.06.2021 р.

3. Вихідні дані до роботи: мова програмування Swift, дані з релакс-центру, літературні джерела з досліджуваної проблеми.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): вступ, аналітичний огляд і постановка завдання, розгляд завдання розробки мобільного додатку, дослідження технологій та засобів, результати виконаної роботи, висновки.

5. Перелік обов’язкового графічного матеріалу: порівняння кросплатформних фреймворків, результати виконаної роботи.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Ознайомлення з предметною областю.	22.04.2021р. – 24.04.2021р.	
2	Складання змісту пояснювальної записки дипломного проекту.	25.04.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділу.	26.04.2021р.	
4	Аналітичний огляд і постановка задачі.	27.04.2021р. – 30.04.2021р.	
5	Аналіз існуючих фреймворків для створення додатку на платформу IOS.	31.04.2021р. – 05.05.2021р.	
6	Пошук інформації для релакс-центру.	05.05.2021р. – 08.05.2021р.	
6	Розробка додатку релакс-центру для платформи IOS.	08.05.2021р. – 28.05.2021р.	
7	Висновки та робота над розділами пояснювальної записки дипломного проекту.	28.05.2021р. – 03.06.2021р.	
8	Підписання необхідних документів у встановленому порядку.	03.06.2021р. – 08.06.2021р.	
9	Отримання відгуку керівника, рецензії	08.06.2021р. – 09.06.2021р.	
10	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	10.06.2021р. – 13.06.2021р.	

Студент **(*Бурова К.О.*)**

Керівник дипломної роботи **(*Климова А.С.)***

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «On-line сервіс релакс-центра для платформи IOS» містить: 63 сторінки, 36 рисунків, 1 таблицю, 7 літературних джерел.

Об’єкт дослідження: створення мобільного додатку використовуючи мову програмування Swift.

Предмет дослідження: розробка мобільного додатку для релакс-центру.

Мета роботи: розробка мобільного додатку для платформи IOS використовуючи мову програмування Swift та найсучасніший фреймворк SwiftUI.

Методи дослідження, технічні та програмні засоби: розробка програмних бібліотек, порівняльний аналіз, обробка літературних джерел.

Отримані результати та їх новизна: створено мобільний додаток для платформи IOS. Використано сучасний архітектурний патерн MVVM, та найновіший фреймворк SwiftUI.

МОБІЛЬНИЙ ДОДАТОК, ФРЕЙМВОРК, ПАТЕРН.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

IOS	–	мобільна операційна система від Apple
Фреймворк	–	інфраструктура програмних рішень, що полегшує розробку складних систем
JSON	–	інфраструктура програмних рішень, що полегшує розробку складних систем

ВСТУП

У наш час важко уявити компанію без сайту, мобільного додатку або веб-сервісу. Мобільний додаток здатний збільшити продажі компанії, оскільки служить засобом залучення та утримання клієнтів.

Такий додаток допомагає компанії рости завдяки тому, що:

1. Працює навіть на малопотужних пристроях (воно тільки показує результат на девайсі клієнта, а все найскладніше відбувається на сервері).
2. Не має потреби в різних версіях ПО для окремих операційних систем (обійдеться однієї універсальної).
3. Дає можливість контролювати і зберігання даних, і виконання бізнес-процесів.
4. Дозволяє запускати нові онлайн-послуги, обходити конкурентів.

Але разом з тим воно приносить нові труднощі:

1. Ускладняється розробка, яка коштує дорожче і робиться довше, ніж сайт-візитка;
2. Виникають додаткові витрати (на оренду сервера або купівлю власного, зберігання резервних копій, зарплату сисадміну і декільком програмістам);
3. Іде час на випуск оновлень, виправлення помилок.

РОЗДІЛ 1
ПРИНЦИПИ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАТФОРМИ IOS ТА ЇЇ ПЕРЕВАГИ

1.1. Огляд мобільного додатку

Сьогодні дуже важко уявити собі мобільний пристрій, на якому б не було встановлено жодного додатку. Вони стійко увійшли в наше життя майже одночасно з планшетами та смартфонами. Саме тому цей напрямок так швидко розвивається і захоплює ринок. Все більше підприємців усвідомлюють необхідність розробки мобільного додатку.

Мобільний додаток – це спеціальна програма розроблена для мобільних пристроїв або планшетів, яка встановлюється на ту чи іншу платформу та має певний функціонал. Тобто виконує певні дії і вирішує поставлене коло питань.

Ще недавно мобільні додатки представляли собою тільки ігри, та не несли користі. Але дуже швидко підприємці зрозуміли, що додаток може допомогти вести бізнес, а також стати сильним маркетинговим інструментом, за допомогою якого можна підвищити впізнаваність і довіру до свого бренду, проводити рекламні кампанії, спростити зворотний зв'язок з клієнтами.

Мобільні додатки пишуться на різних мовах програмування (Java; PHP / JavaScript; ActionScript; Swift і Objective-C). У процесі створення додаток проходить кілька етапів.

1) Перший етап. Визначити, для чого потрібний додаток і які завдання він виконуватиме.

2) Другий етап. Проектування і дизайн.

3) Третій етап. Процес розробки.

4) Четвертий етап. Тестування.

5) П'ятий етап. Моніторинг.

Також популярність отримали webview-додатки, які представляють собою мобільні версії сайтів, які відображаються в інтерфейсі мобільного додатка. Webview програми доступні на платформах Android і iOS. Такий додаток може відображати сайт, створений за технологією веб-додатка. Для користувача такий додаток буде здаватися нативним і мати всі необхідні функції. Для розробника використання даної технології знижує витрати на написання окремого коду для мобільного застосування, так як сайт за технологією веб-додатка вже має весь необхідний функціонал. Також webview додатки можуть використовувати нативні функції систем iOS і Android, такі як push-повідомлення, оплату через GooglePay або ApplePay, і багато інших.

1.2. Обмеження та можливості мобільного додатку

Під час розробки програми для мобільних пристроїв треба враховувати їх обмеження та можливості. Оскільки смартфони працюють на акумуляторі, мають менш потужні процесори, ніж персональні комп'ютери, а також мають функції місцезнаходження та камери, тому розробникам доводиться враховувати широкий спектр розмірів дисплею, різні технічні характеристики та конфігурації обладнання через сильну конкуренцію мобільного ПО та зміни в кожній платформі.

Розробка таких програм вимагає використання спеціалізованих інтегрованих середовищ розробки(IDE). Мобільні додатки спершу проходять тестування в середовищі розробки, використовуючи емулятори, а потім перевіряються на місцях. Емулятори дають змогу недорого протестувати програми на смартфонах, до яких розробники не мають фізичного доступу.

Важливу роль у розробці застосунку відіграє дизайн мобільного інтерфейсу користувача. Користувач часто фокусується на взаємодії з пристроєм, а інтерфейс тягне за собою компоненти як апаратного, так і програмного забезпечення. Вхід користувача дозволяє користувачам маніпулювати системою, а випуск пристрою дозволяє системі вказувати ефекти маніпулювання користувачами. Обмеження дизайну мобільного інтерфейсу включають в себе обмежені увагу та форм-фактори, такі як розмір екрана мобільного пристрою для користувача. Контексти мобільного інтерфейсу сигналізують сигнали про активність користувача, такі як розташування та планування, які можуть відображатися в результаті взаємодії користувачів у мобільній програмі. В цілому, мета дизайну мобільного інтерфейсу в першу чергу — для зрозумілого, зручного інтерфейсу.

Мобільні інтерфейсні системи або front-end покладається на мобільні резервні копії (back-end) для підтримки доступу до корпоративних систем. Мобільний зворотний зв'язок полегшує маршрутизацію даних, безпеку, автентифікацію, авторизацію, роботу поза мережею. Ця функціональність підтримується сумішшю компонентів проміжних програм, включаючи сервери мобільних застосунків, Mobile Backend як службу (MBaaS) та інфраструктуру SOA.

Розмовні інтерфейси відображають комп'ютерний інтерфейс і надають взаємодію через текст, а не графічні елементи. Вони імітують розмови з справжніми людьми. Є два основних типи розмовних інтерфейсів: голосові помічники (як Amazon Echo) та chatbots.

1.3. Види мобільних додатків та їх переваги і недоліки

На сьогоднішній день існують такі види мобільних додатків:

- Нативні додатки;
- Web-додатки;
- Крос-платформні або гібридні додатки;
- Мобільні сайти.

Нативні додатки – це додатки, які знаходяться на самому пристрої, і доступ до них можна отримати, натиснувши на іконку програми. Такі додатки встановлюються через магазин додатків (таких як Play Market на Android, App Store на iOS і ін.). Вони розроблені спеціально для конкретної платформи і можуть використовувати всі можливості пристрою - камеру, GPS-датчик, акселерометр, компас, список контактів і все інше. Також вони можуть розпізнавати жести (стандартні жести, встановлені операційною системою або зовсім нові жести, які використовуються в конкретному додатку). Нативні додатки можуть отримати доступ до системи оповіщення пристрою, а також продовжувати свою роботу в режимі оффлайн.

Такі додатки кодуються на тій же мові що і сам телефон. Наприклад мова Objective C використовуються для iOS, а Java для операційних систем Android. Додатки на рідних мовах забезпечують швидку продуктивність і високу ступінь надійності.

Тим не менш, цей тип додатків найдорожчий, бо він прив'язаний до одного типу операційної системи. Іншими словами під кожну платформу необхідно буде писати нову копію цього додатка на іншій мові. Що збільшує вартість в два рази. Більшість ігор в мобільних додатках написані нативними мовами.

Мобільні web-додатки насправді не є додатками як такими. Це веб-сайти, які багато в чому виглядають і відчуваються як нативні додатки, проте все ж не можуть повністю замінити їх. Вони запускаються за допомогою браузера і, як правило, написані на мові HMTL5. Запускаючи мобільні веб-додатки, користувач виконує всі ті дії, які він виконує при переході на будь-який веб-сайт, а також отримує можливість «встановити» їх на свій робочий стіл, створивши закладку сторінки веб-сайту.

Веб-додатки стали широко популярні в той час, коли почав розвиватися HMTL5 і люди усвідомили, що можуть отримати доступ до безлічі функцій нативних додатків, просто зайшовши на веб-сайт через звичайний браузер. На сьогоднішній день складно сказати, де саме розташовується чітка межа між веб-додатками і звичайними веб-сторінками, оскільки функціонал HTML5 зростає з кожним днем і все більше і більше сайтів використовують HTML5.

Такі додатки запускаються через браузер і не вимагають установки. На відміну від звичайних веб-сайтів, які повинні звертатися до сервера кожного разу, коли користувач виконує дію. Кращим прикладом веб-додатку є поштовий сервіс Google «Gmail».

Гібридні додатки – симбіоз нативних додатків і веб-додатків. Так само, як і нативні додатки, їх можна завантажити в магазині додатків, а самі гібридні програми можуть використовувати безліч функцій пристрою, на якому вони встановлені. Так само, як і веб-додатки, основою їх платформи є HTML5. Вони обробляються через браузер, який вбудований в сам додаток.

Найчастіше компанії випускають такі гібридні програми на додачу до вже наявних веб-сайтів в якості оболонки, щоб їх застосування було в списку магазину додатків. Це дозволяє не витрачати величезні суми на розробку окремих мобільних додатків і займає набагато менше часу.

Також однією з причин високої популярності гібридних додатків є той факт, що всі вони написані однією мовою. Це дозволяє випускати крос-платформні додатки, які будуть однаково добре працювати на будь-якій з існуючих систем.

Вони встановлюються на телефон, але написані за допомогою веб-технологій (HTML5, CSS і JavaScript). Перш за все це дозволяє отримати доступ до можливостей телефону, які недоступні в мобільних веб-додатках. Наприклад акселерометр, камера і локальне сховище.

Однак з дизайном додатка можуть виникнути проблеми, оскільки різні платформи мають різні стандарти UI / UX. Наприклад, дизайн для iPhone відрізняється від дизайну для Android.

Мобільні сайти – не явна категорія, але заслуговує на увагу. Мобільні сайти дають можливість без істотних витрат протестувати ринок, затвердити ідею або побудувати MVP. Тим самим зробити бізнес більш доступним на мобільних пристроях.

Використовуючи адаптивні CSS-фреймворки, такі як Bootstrap, можна створювати веб-додаток або веб-сайт, який адаптуються під різні телефони.

Користувачі будуть використовувати один і той же додаток / веб-сайт, але дизайн зміниться відповідно до дозволу екрану, який вони використовують.

1.4. Операційна система IOS

iOS – пропріетарна операційна система від компанії Apple, що випускається виключно для пристроїв даної компанії. Вперше представлена Стівом Джобсом 9 січня 2007 року одночасно з першою моделлю iPhone. Пізніше була розроблена версія для iPad, Apple TV, Apple Watch.

В основі iOS лежить POSIX-сумісна ОС Darwin, а основа Darwin – це ядро XNU, що з'явилося на світ в результаті злиття мікроядра Mach і компонентів ядра FreeBSD. Єдина підтримувана архітектура – ARM.

Операційна система iPhone OS була представлена 9 січня 2007 року спільно з мобільним телефоном iPhone особисто Стівом Джобсом на виставці-конференції Macworld Conference & Expo і випущена в червні того ж року. Apple не припускала окремої назви для операційної системи, тому початковий слоган звучав так: «iPhone працює на OS X». [1]

Умовно начинку OS X / iOS можна розділити на три логічних рівня (рис. 1.1): ядро XNU, шар сумісності зі стандартом POSIX (плюс різні системні демони / сервіси) і шар NeXTSTEP, який реалізує графічний стек, фреймворк і API додатків. Darwin включає в себе перші два шари і поширюється вільно, але тільки у версії для OS X. iOS-варіант, портований на архітектуру ARM і включає в себе деякі доопрацювання, повністю закритий і поширюється тільки в складі прошивок для айдевайсов (судячи з усього, це захист від портатирування iOS на інші пристрої).

За своєю суттю Darwin – це «гола» UNIX-подібна ОС, яка включає в себе POSIX API, шелл, набір команд і сервісів, мінімально необхідних для роботи системи в режимі консолі і запуску UNIX-софта. У цьому сенсі вона схожа на базову систему FreeBSD або мінімальну установку якогось Arch Linux, які дозволяють запустити консольний UNIX-софт, але не мають ні графічної оболонки, ні всього необхідного для запуску серйозних графічних додатків з середовищ GNOME або KDE.

Ключовий компонент Darwin – гібридне ядро XNU, засноване, як уже було сказано вище, на ядрі Mach і компонентах ядра FreeBSD, таких як планувальник процесів, мережевий стек і віртуальна файлова система (шар VFS). На відміну від Mach і FreeBSD, ядро OS X використовує власний API драйверів, названий I / O Kit і дозволяє писати драйвери на C ++, використовуючи об'єктно-орієнтований підхід, сильно спрощує розробку.

iOS використовує кілька змінену версію XNU, проте в силу того, що ядро iOS закрито, сказати, що саме змінила Apple, важко. Відомо тільки, що воно зібрано з іншими опціями компілятора і модифікованим менеджером пам'яті, який враховує невеликі обсяги оперативки в мобільних пристроях. У всьому іншому це все той же XNU, який можна знайти у вигляді зашифрованого кеша в каталозі /System/Library/Caches/com.apple.kernelcaches/kernelcache на самому пристрої.

Рівнем вище ядра в Darwin розташовується шар UNIX / BSD, що включає в себе набір стандартних бібліотек мови Ci (libc, libmatch, libpthread і так далі), а також інструменти командного рядка, набір Шелл (bash, tcsh і ksh) і демонів, таких як launchd і стандартний SSH-сервер. Останній, до речі, можна активувати шляхом правки файлу /System/Library/LaunchDaemons/ssh.plist.

На цьому відкрита частина ОС під назвою Darwin закінчується, і починається шар фреймворків, які як раз і утворюють те, що ми звикли вважати OS X / iOS.

Darwin реалізує лише базову частину Mac OS / iOS, яка відповідає тільки за низькорівневі функції (драйвери, запуск або зупинка системи, управління мережею, ізоляція додатків і так далі). Та частина системи, яку видно користувачеві і додаткам, в його склад не входить і реалізована в так званих фреймворках - наборах бібліотек і сервісів, які відповідають в тому числі за формування графічного оточення і високорівнева API для сторонніх і стічних додатків.

Як і в багатьох інших ОС, API Mac OS і iOS розділений на публічний і приватний. Стороннім додаткам доступний виключно публічний і сильно урізаний API, проте jailbreak-додатки можуть використовувати і приватний.

У стандартному постачанні Mac OS і iOS можна знайти десятки різних фреймворків, які відповідають за доступ до різних функцій ОС – від реалізації

адресної книги (фреймворк AddressBook) до бібліотеки OpenGL (GLKit). Набір базових фреймворків для розробки графічних додатків об'єднаний в так званий Cocoa API, свого роду метафреймворк, що дозволяє отримати доступ до основних можливостей ОС. В iOS він носить ім'я Cocoa Touch і відрізняється від настільної версії орієнтацією на сенсорні дисплеї.

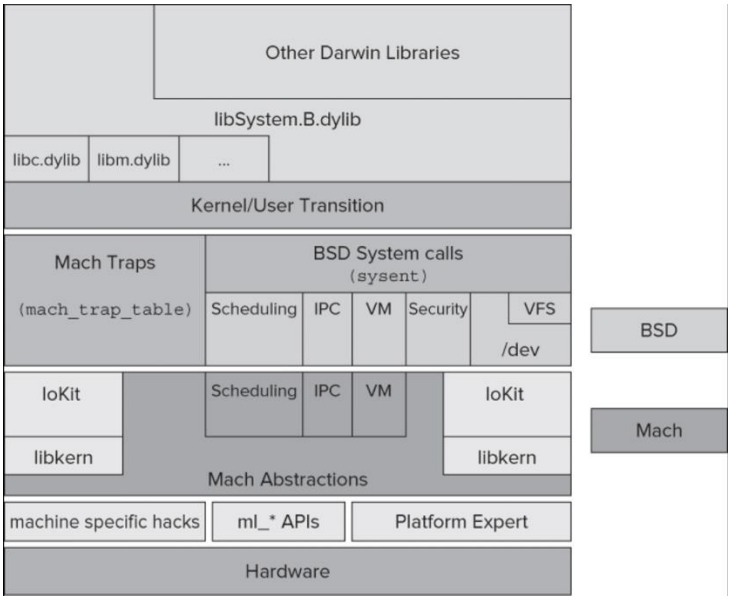

Рис. 1.1. Пристрій IOS

Далеко не всі фреймворки доступні в обох ОС. Багато з них специфічні тільки для iOS. Як приклади можна привести AssetsLibrary, який відповідає за роботу з фотографіями та відео, або iAd, призначений для виведення рекламних оголошень в додатках. Інші фреймворки існують тільки в настільній версії системи, однак час від часу Apple переносить ті чи інші частини iOS в Mac OS або назад, як, наприклад, сталося з фреймворком CoreMedia, який спочатку був доступний тільки в iOS.

Одна з цікавих особливостей фреймворків - їх версійність. Один фреймворк може мати відразу кілька різних версій, тому додаток, розроблений для застарілих версій системи, буде продовжувати працювати, навіть незважаючи на зміни, внесені в нові версії ОС. Саме так реалізований механізм запуску старих iOS-додатків в iOS 7 і вище. Додаток, розроблений для iOS 6, буде виглядати і працювати саме так, як якщо б він був запущений в iOS 6.

Рівнем вище знаходяться додатки, системні і ті, які встановлюються з магазину додатків. Центральне місце серед них займає, звичайно ж, SpringBoard (тільки в iOS), що реалізує домашній екран (робочий стіл). Саме він запускається першим після старту системних демонів, завантаження в пам'ять фреймворків і старту дисплейного сервера, що відповідає за виведення зображення на екран.

SpringBoard – це сполучна ланка між операційною системою і її користувачем, графічний інтерфейс, що дозволяє запускати додатки, перемикатися між ними, переглядати повідомлення та керувати деякими налаштуваннями системи (починаючи з iOS 7). Але також це і обробник подій, таких як дотик екрану або переворот пристрою. На відміну від Mac OS X, яка використовує різні додатки і демони-агенти для реалізації компонентів інтерфейсу (Finder, Dashboard, LaunchPad і інші), в iOS майже всі базові можливості інтерфейсу користувача, в тому числі екран блокування і «шторка», укладені в одному SpringBoard.

На відміну від інших стічних додатків iOS, які розташовуються в каталозі / Applications, SpringBoard нарівні з дисплейним сервером вважається частиною фреймворків і розташовується в каталозі / System / Library / CoreServices /. Для виконання багатьох завдань він використовує плагіни, які лежать в / System / Library / SpringBoardPlugins /. Крім усього іншого, там можна знайти, наприклад, NowPlayingArtLockScreen.lockboundle, що відповідає за відображення інформації про програвання композиції на екрані блокування, або IncomingCall.serviceboundle, відповідальний за обробку вхідного дзвінка.

Починаючи з iOS 6 SpringBoard розділений на дві частини: сам робочий стіл і сервіс BackBoard, відповідальний за комунікації з низькорівневою частиною ОС, що працює з обладнанням (рівень HAL). BackBoard відповідає за обробку таких подій, як торкання екрану, натискання клавіш, отримання свідчення акселерометра, датчика положення і датчика освітленості, а також управляє запуском, припиненням і завершенням додатків.

SpringBoard і BackBoard мають настільки велике значення для iOS, що, якщо будь-яким чином їх зупинити, вся система застигне на місці і навіть запущена в даний момент програма не буде реагувати на торкання екрану. Це відрізняє їх від

домашнього екрану Android, який є всього лише стандартним додатком, який можна зупинити, замінити або взагалі видалити з системи (в цьому випадку на екрані залишаться цілком робочі кнопки навігації і рядок стану з «шторкою»).

1.5. Переваги та недоліки платформи IOS

Зазвичай платформу IOS порівнюють із її найпопулярнішим конкурентом – платформою Android. ***Але все ж таки IOS має такі переваги:***

1) Якість додатків.

Додатки практично завжди виглядають привабливіше, зручніше на iPhone і iPad. Просто порівняйте в магазині, запустивши однакові додатки. Доходить до комічного: порівняйте, як виглядають іконки однакових додатків, які, здавалося б, повинні виглядати однаково: на Android іконки будуть виглядати непривабливо.

2) Швидке оновлення.

Користувачам iPhone і iPad не потрібно чекати, коли виробники спроможуться підготувати оновлення для їх влаштування після релізу нової версії iOS. Більшість користувачів до цих пір не отримали Android 5.0, яку Google анонсувала в минулому році. Оновлення для iOS виходять регулярно і в один момент стають доступні для всіх сумісних пристроїв.

3) Тривала підтримка старих пристроїв.

Період підтримки мобільних телефонів Apple становить 48 місяців. Доступ до операційної системи iOS 8 отримали навіть користувачі смартфона iPhone 4s, представленого в 2011 році. Звичайно, на пристрої доступні не всі можливості ОС, однак користувачі можуть використовувати ряд важливих функцій мобільної платформи і запускати додатки з App Store, сумісні тільки з iOS 8. Те ж саме відбулося в 2012 році з iPhone 3GS. У користувачів була можливість перейти на iOS 6 через 46 місяців після початку продажів смартфона.

4) Кращі програми доступні першими.

Більшість розробників вирішують спочатку випустити додаток на iPhone і iPad і тільки через деякий час запустити його на Android. Це пов'язано з добре опрацьованим інструментарієм розробки для iOS. Наприклад, Instagram, одна з найбільших соціальних мереж світу, більше року була доступна тільки на iPhone. І тільки потім її запустили для Android.

Найактивніші користувачі на iOS, ось чому розробники всіх популярних і успішних програм роблять свій продукт в першу чергу для цієї операційної системи і лише потім допрацьовують його для платформ конкурентів.

5) Екосистема Apple.

Сьогодні при виборі смартфона або планшета головними не є різні технічні характеристики на кшталт ємності батареї, дозволу камери і т.д. Вони більш-менш схожі на всіх сучасних пристроях. Головне для користувача - це мобільна екосистема. І на iOS вона набагато більш розвинена. У недавніх промо-матеріалах Apple представила свою екосистему в вигляді чотирьох ключових продуктів - Apple Watch, MacBook, iPhone і iPad. Сюди ще можна додати Apple TV, роутери AirPort та навушники AirPods.

Але у системи компанії Apple є ряд істотних недоліків:

1) Нове оновлення системи дуже часто уповільнює пристрій. І це робить сама Apple, щоб ви задумалися і купили собі новий телефон.

2) Операційна система є закритою. Не можна подивитися список файлів операційної системи і використовувати пристрій як флешку. Це є одночасно і гідністю. iOS – найбільш захищена система в світі.

3) Дорожнеча телефонів і планшетів на даній операційній системі.

4) Деякі аксесуари може випускати тільки Apple. Наприклад, краще користуватися кабелями зарядки, які створив Apple. У них вшитий чіп. Якщо ви купите китайський кабель для зарядки, то можливо він не буде працювати або ж ваш пристрій вийде з ладу через підроблену продукцію.

5) Свій додаток у вигляді файлу встановити не можна або дуже важко. Додатки можна брати тільки з AppStore.

1.6. Нативний додаток для платформи IOS

Розробка нативних додатків є однією з найчастіше використовуваних розробниками платформ, завдяки широким можливостям, дружньому функціоналу як до користувачів, так і до розробників. Додатки, створені з використанням вбудованого програмного забезпечення, забезпечують максимальний призначений для користувача досвід в порівнянні з гібридним. Нативний підхід зберігає ресурси програми в пам'яті пристрою і забезпечує максимальне використання функцій ОС.

Нативний додаток забезпечує:

1) Високу якість.

Розробник, який вузько спеціалізується на нативних додатках, напише вам чистий, унікальний код. Багаторічний досвід розробки і чіткі стандарти нативних iOS & Android додатків допоможуть зробити якісний продукт з широким функціоналом і знизити ризик появи помилок практично до мінімуму.

2) Низьку вірогідність відмови в розміщенні в App & Play Stores.

Оскільки нативний додаток спочатку відповідає стандартним вимогам певної платформи, малоймовірно, що ви зіткнетеся з якими-небудь проблемами при запуску вашого застосунку в офіційних магазинах App Store і Play Store.

3) 100% використання UX дизайну.

Сучасні користувачі розпещені яскравими, детально-опрацьованими інтерфейсами, і прості, стандартизовані додатки навряд чи зацікавлять їх. Саме в нативній розробці UX дизайн використовується на всі 100%, що дозволяє зробити якісний і цікавий додаток. У гібридному додатку ви отримаєте стандартизований для двох платформ інтерфейс.

4) Різноманітність інструментів для розробки.

Завдяки багаторічному досвіду розробки нативних додатків з'явилася величезна кількість різних фреймворків, шаблонів та інших перевірених інструментів, які дозволять зробити ваш додаток унікальним, індивідуальним і стабільним.

5) Велике співтовариство розробників.

Ну і звичайно ж, розробляючи нативний додаток, ви навряд чи зіткнетеся з

проблемою, яку ніхто не вирішував до вас. А це означає, що вам не доведеться витрачати зайвий час на пошук відповідного рішення, а можна буде звернутися до досвіду інших програмістів.

Але у той же час вартість такого додатку дуже висока. Оригінальний додаток – це унікальний, якісний продукт, для створення якого потрібно чимало часу і, звичайно ж, висококваліфікований розробник з багаторічним досвідом. Тому і коштує такий додаток відповідно.

При розробці нативного додатка у вас є величезна різноманітність інструментів, що входять в SDK тієї чи іншої платформи. Тобто все, що вам потрібно – використовувати ці інструменти в своєму нативному додатку. У випадку з гібридом – вам залишається сподіватися, що на той чи інший нативний інструмент існує адаптація на базі фреймворка, обраного для гібридної розробки. Якщо ж такого інструменту немає – вам доведеться або чекати його появи, або розглядати альтернативні фреймворки, тобто мороки з гібридом набагато більше. Виходячи з цього, стає зрозуміло, що, створити один нативний iOS додаток дешевше, ніж один гібридний iOS додаток. Якщо ж порівнювати розробку гібридного додатка і двох нативних, то ціна гібрида буде нижче, як і очікувалося, адже в гібридному додатку backend і frontend підходять для двох платформ відразу. У нативному додатку потрібно розробляти два окремих frontend'a, що відповідають загальноприйнятим стандартам кожної з платформ.

1.7. Мови програмування для створення додатку на IOS

Для того, щоб стати розробником програм, які зможуть працювати на пристроях під управлінням iOS, iPadOS, tvOS, macOS, watchOS, вам потрібно освоїти Objective-C і / або Swift. Саме на цих мовах програмування і пишуться додатки під iOS. Обидві ці мови є об'єктно-орієнтованими, і дозволяють групувати в процесі написання коду схожі завдання, що в свою чергу спрощує завдання розробників і прискорює їх роботу. Objective-C є більш старим, і вперше був представлений ще в 80-х роках минулого століття. Поступово він допрацьовувався і став основним для

Apple, тому за допомогою Objective-C можна створити додатки під будь-які ОС пристроїв Apple корпорації. До переваг даної мови можна віднести високу ступінь підтримки коду, величезну базу навчальних матеріалів і велика спільнота, схожість з сімейством мов C, сумісність з більш «молодою» мовою Swift.

У 2014 році корпорація Apple представила нову мову програмування, що отримала назву Swift. За словами розробників, дана мова увібрала в себе все найкраще від популярних Objective-C і C, і при цьому отримала більш потужний і зручний функціонал, більш сучасний інструментарій.

До головних переваг даної мови можна віднести високу швидкість розробки програм, зменшення коду, кращу чіткість, підтримку динамічних бібліотек, поліпшену безпеку. Swift і Objective-C сумісні, тому їх можна використовувати навіть в рамках одного проекту.

Мова Swift продовжує активно розвиватися, і все більша кількість розробників переходять на неї або починають свою трудову діяльність саме з неї. Сама Apple робить ставку на Swift, але і повністю відмовлятися від Objective-C точно не варто.

Найпопулярнішим інтегрованим середовищем розробки (IDE) є безкоштовний продукт XCode, створений самою компанією Apple.

XCode – досить зручний додаток із чималим набором корисних інструментів, істотно прискорює і спрощує процес написання програм. В одному середовищі можна написати додаток, провести тестування і оптимізацію, і зібрати відразу на потрібному iOS-пристрої.

Інтерфейс єдиного вікна істотно спрощує роботу розробнику, причому в процесі написання коду програма вкаже програмісту на допущену помилку, якщо така з'явиться. У XCode інтегрований додаток IB (Interface Builder), що дозволяє розробляти графічні інтерфейси, налаштовувати стилі і шрифти.

І це лише основні переваги якими володіє дане середовище розробки iOS додатків. Головним конкурентом XCode є розробка JetBrains – AppCode з відмінною роботою автодоповнення, хорошою інтеграцією з issue-трекера, докладним описом помилок. Але дану IDE в якості основної і єдиної використовувати навряд чи вийде, і в більшості випадків розробники повертаються до XCode.

1.8. Кросплатформні мови програмування

Ми розглянемо п'ять фреймворків – React Native, Flutter, Ionic, Xamarin, PhoneGap, і розберемося, в яких випадках вигідно їх використовувати. У таблиці 2.1 наведено порівняння усіх цих фреймворків.

1) React Native.

Інструмент від Facebook. Його мета – зробити кросплатформні додатки такими ж продуктивними, як нативні.

React Native використовую мову програмування JavaScript та бібліотеку React. Він досить популярний, оскільки його вже застосовують технологічні гіганти. Серед них Instagram, Facebook, Walmart, Tesla, Pinterest, UberEats і інші.

З моменту запуску React Native пройшло близько 5 років, тому його підтримують майже всі провідні IDE. Вивчати React Native і писати код на ньому досить просто завдяки використанню JavaScript (зрозуміло, якщо ви знаєте JavaScript).

«Learn once, write anywhere», що можна трактувати як «навчися один раз, використовуй всюди» – головний принцип React Native, який має на увазі застосування одного і того ж коду для різних платформ. Також в Native є функція Hot Reloading, що дозволяє додавати новий код і вносити правки прямо під час виконання – це дуже корисно, коли ви налаштовуєте призначений для користувача інтерфейс. Середа поставляється з великим набором готових компонентів, однак вони не завжди адаптуються під різні платформи, що вимагає додаткових коригувань в коді. Завдяки великій підтримці спільноти також є багатий вибір сторонніх бібліотек.

Так як React Native націлений на результат, який можна порівняти з нативною розробкою, в гонитві за продуктивністю найчастіше віддають перевагу саме цьому фреймворку. Native також дозволяє розробникам використовувати налаштовувані модулі на мовах для нативної розробки, але їх доведеться писати окремо для кожної платформи.

2) Flutter.

Flutter – дітище Google, вже завоювало хорошу репутацію в кроссплатформній розробці. Його принцип – створення додатків з єдиною кодової базою для мобільних платформ, веба і робочого столу.

Flutter використовує мову програмування Dart, це об'єктно-орієнтована мова, розроблена Google.

Flutter з'явився на ринку не так давно, але його популярність зросла за дуже короткий час. Додатки на ньому можна побачити у Alibaba, Hamilton Musical, Greentea, Google Ads.

Flutter підтримується Android Studio / IntelliJ і Visual Studio Code. Що стосується програмування на Dart – якщо ви знаєте C ++ або Java, вам буде простіше його освоїти. Однак вивчення будь-якого нового мови вимагає часу.

Flutter використовує один і той же код для всіх платформ. На ньому легко створювати красиві інтерфейси. Але якщо вам потрібні різні стилі для різних ОС, доведеться трохи попрацювати, оскільки автоматична адаптація для цих цілей не передбачена. Це пов'язано з тим, що замість нативних компонентів Flutter застосовує свій графічний движок. Однак він не відстає від React Native і включає автоматичний Hot Reloading (додавання нового коду без повторного складання) і великий набір готових віджетів. Зате з Flutter можна випускати додатки для різних версій Android і iOS без додаткових рухів тіла: програми спокійно запускаються навіть на таких старих версіях, як Android Jelly Bean і iOS 8.

Можна сказати, що Flutter перевершує конкурентів і демонструє найвищу продуктивність завдяки сучасній мові Dart і власному движку рендеринга.

3) Ionic

З Ionic можна створювати кросплатформні гібридні програми. Він тісно взаємодіє з фреймворком Apache Cordova, який перетворює веб-додатки в мобільні програми.

Ionic використовує мови програмування JavaScript + HTML, CSS.

Додатки на Ionic можна побачити у MarketWatch, Pacifica, Sworkit, Nationwide.

Ionic завоював визнання серед розробників мобільних додатків, тому що з ним просто працювати. Фреймворк побудований на ECMAScript 6 і TypeScript, тому його

можна використовувати в будь-якій IDE, що підтримує ці мови, наприклад в Visual Studio Code, Atom або Angular IDE. До речі, якщо ви вже знайомі з Angular, React або Vue, то з освоєнням Ionic не виникне особливих труднощів.

Ionic, як і React Native і Flutter, пропонує концепцію єдиного коду для різних платформ, але на новому рівні. Всі його компоненти автоматично адаптуються до платформи, на якій запускається додаток – а значить, розробка стає швидше. Також з Ionic ви можете вільно використовувати JavaScript, Angular, React або Vue.

А ось з продуктивністю Ionic програє і сильно відстає від React Native і Flutter, оскільки для візуалізації додатків він використовує веб-технології та зовсім не застосовує нативні компоненти. Такий підхід значно знижує швидкість.

Але з боку розробки є і плюси: Ionic дозволяє проводити швидке тестування, яке можна запустити прямо в браузері.

4) Xamarin.

Xamarin – платформа для створення мобільних додатків від Microsoft, яка також підтримує розробку для Windows.

Цей фреймворк використовує мови програмування C# та .NET.

На Xamarin зроблені додатки Olo, The World Bank, Storyo і інші.

Як IDE можна використовувати, наприклад, Visual Studio 2019 або Rider. C# досить поширений, тому з написанням коду і освоєнням Xamarin проблем виникати не повинно.

У Xamarin є два основні інструменти: Xamarin.Android/iOS і Xamarin.Forms. По частині кросплатформної розробки Xamarin пропонує використовувати єдиний API Xamarin.Essentials.

Xamarin.Android і Xamarin.iOS наділяють додаток тими ж можливостями і інтерфейсом, які є у нативних рішень. У разі Xamarin.iOS програма компілюється безпосередньо в машинний код (AOT-компіляція), тоді як в Xamarin.Android спочатку відбувається компіляція в байт-код, який потім інтерпретується віртуальною машиною (JIT-компіляція).

Якщо ж потрібно прискорити процес написання коду, краще використовувати Xamarin.Forms – більш простий інструмент, в якому майже всі елементи повністю

сумісні з будь-якими платформами.

Продуктивність Xamarin також вважається близькою до нативної, але залежить від того, використовуєте ви Xamarin.Android, Xamarin.iOS або Xamarin.Forms. У Xamarin.Android/iOS хороша оптимізація завдяки нативним компонентів. Xamarin.Forms же заснований на 100% спільному використанні коду, що в цілому знижує його продуктивність в порівнянні з Xamarin.Android/iOS.

5) PhoneGap.

Як і Ionic, PhoneGap дозволяє використовувати веб-технології в мобільній розробці. Він є дистрибутивом Apache Cordova.

PhoneGap використовує мови програмування JavaScript разом із HTML та CSS.

Серед прикладів додатків на PhoneGap: Logitech Squeezebox Controller, Localeur, Untappd, HealthTap.

Для більш комфортної кроссплатформенной розробки та тестування можна використовувати Adobe Dreamweaver (версії 5.5 і вище), MyEclipse 2013, Tiggzi, ApplicationCraft. Розробка на JavaScript не повинна викликати труднощів, особливо якщо раніше ви вже писали на ньому.

Додаток PhoneGap, по суті, являє собою набір HTML-сторінок, обгорнутих в нативну оболонку. Сторінки зберігаються в локальному каталозі або в хмарі, а під час запуску на смартфоні вони отримують доступ до функцій пристрою через плагіни. Це робить додатки PhoneGap досить легкими, але вони виглядають менш природно, а якість призначеного для користувача інтерфейсу буде більшою мірою залежати від веб-уявлення конкретної ОС.

PhoneGap відрізняється невисокою продуктивністю в порівнянні з нативними інструментами – і в цьому знову винні веб-технології.

Кожен інструмент має свої особливості, і робити вибір слід виходячи з конкретного завдання. Завжди варто обговорювати проект з досвідченою командою розробників, яка розгляне різні підходи і запропонує найкращий варіант.

Таблиця 2.1

Порівняння кросплатформних фреймворків

	React Native	**Flutter**	**Ionic**	**Xamarin**	**PhoneGap**
Мова програмування	JavaScript + React	Dart	JavaScript, HTML, CSS + Angular, React	C# + .NET	JavaScript, HTML, CSS
Додатки	Крос-мні	Крос-мні	Крос-мні, гібридні	Крос-мні	Крос-мні, гібридні
Перший реліз	2015	2017	2013	2011	2009
Хто розроблює	Facebook + спільнота	Google + спільнота	Drifty Co.	Microsoft	Adobe
Спільнота	Дуже велика	Невелика, але активно розвивається	Велика	Велика	Велика
Платформи	Android, IOS, UWP	Android, IOS, Google, Fuchsia, Web, Desktop	Android, IOS, Web	Android, IOS, UWP	Android, IOS, Windows Phone 8
Open source	Так	Так	Так + платні пакети	Так + платні пакети	Так
Інструменти фронтенда	Компоненти Native + Declarative UI	Вбудовані віджети	HTML, CSS + віджети	Xamarin.IOS/Android або Xamarin.Forms	HTML, CSS
Продуктивність	Висока, близька до нативної	Дуже висока	Середня	IOS/Android: висока, близька до нативної Forms: середня	Середня

Висновки

Отже, нативні додатки здаються набагато кращими з точки зору продуктивності і користувальницького досвіду. Однак не будемо забувати, що вибір дійсно залежить від вашої програми. Прості програми, такі як ігри та програми для поширення контенту, зазвичай розробляються як кросплатформний додаток, в той час як додатки

зі складними функціями як нативні.

Кросплатформність також краща для додатків B2B, де час розгортання має першорядне значення. Багато малі підприємства також вибирають кросплатформність через свого обмеженого бюджету. Проте, зниження продуктивності і зручності користувачів заради економії часто призводить до зворотних результатів.

Важливо вибрати платформу, яка відповідає вашим потребам, вимогам, а також вашої цільової аудиторії.

РОЗДІЛ 2
МОВА ПРОГРАМУВАННЯ SWIFT, ОСОБЛИВОСТІ ФРЕЙМВОРКІВ SWIFTUI ТА UIKIT

2.1. Мова програмування Objective-C

Objective-C – це об’єктно-орієнтована мова програмування загального призначення, яка додає обмін повідомленнями в стилі Smalltalk до мови програмування C. Це основна мова програмування, що використовується Apple для операційних систем OS X та iOS та їх відповідних API, Cocoa та Cocoa Touch. Таким чином, спочатку Objective-C сприймали як надбудова над C. У якомусь розумінні це робиться так і до цих пір: можна написати програму на чистому C, а після додати до неї трохи конструкцій з Objective-C (за необхідності), або же навпаки, вільно користуватися C у програмах на Objective-C. Крім того, все це стосується і програм на C++. У 1988 році NeXT (а в наслідку Apple) ліцензував Objective-C і написав для нього компілятор та стандартну бібліотеку (по суті SDK). У 1992 р. до удосконалення мови та компілятора були підключені розробники проекту GNU в рамках проекту OpenStep. З тих пір GCC підтримує Objective-C. Після покупки NeXT, Apple взяла їх SDK (компілятор, бібліотеки, IDE) за основу для своїх подальших розробників. IDE для коду назвав XCode, а для графічного інтерфейсу – Interface Builder. Фреймворк Cocoa для розробки графічного інтерфейсу (і не тільки) на сьогодні є найбільш значущою середою розробки програм на Objective-C. [2]

Файли модулів на Objective-C мають розширення “.m” (якщо використовувалася суміш C ++ та Objective-C, то розширення “.mm”). Заголовочні файли – “.h”. Всі, створені в Objective-C об'єкти класів, повинні розміщуватися в динамічній пам'яті. Тому особове значення набуває тип ідентифікатора, який вказує на об'єкт будь-якого класу (по суті void*). Нульовий покажчик іменується константою нуль. Таким чином, вказівник на будь-який клас можна внести до типу id. Виникає проблема: як дізнатися до якого класу відноситься об'єкт, що криється під id? Це робиться завдяки інваріанту, який присутній у будь-якому об'єкті класу, вивчаючи спеціальний базовий клас NSObject (приставка NS, що визначає NeXT Step). Інваріант isa відноситься до зарезервованого типу Class. Об'єкт такого типу дозволяє визначити ім'я свого та базового класу, набір інваріантів класу, а також прототипи всіх методів, які реалізували цей об'єкт та їх адресу (за допомогою локального списку селекторів). Усі зарезервовані слова Objective-C, які відрізняються від зарезервованих слів мови C, починаються з символу @ (наприклад @protocol, selector, interface). Зазвичай імена інваріантів класів з обмеженою областю видимості (@private, protected) починаються з символів підкреслення. Для рядків у Cocoa є дуже зручний клас NSString. Рядкова константа такого класу записується як @”Hello world”, а не як звичайна для C рядкова константа “Hello world”. Тип BOOL (по суті unsigned char) може приймати постійні значення ТАК або НІ. Усі особливі для Objective-C зарезервовані слова (які відрізняються від мови C і знаходяться в загальнодоступному файлі objc / objc.h) наведені нижче:

- @interface: починає оголошення класу або категорій (категорія – розширення класу додатковими методами без досліджень);
- @implementation: починає визначення класу або категорій;
- @protocol: починає оголошення протоколу (аналог класу C++, який складається з чисто віртуальних функцій);
- @end: завершує оголошення \ визначення будь-якого класу, категорії чи протоколу;
- @private: обмежує область видимостей інваріантів класу методами класу (аналогічно C ++)

– @private: стоїть за замовчуванням. Обмежує область видимості інваріантів класу методів класу та методів виробничих класів (аналогічно C ++);
– @public: видаляє обмеження на область видимості (аналогічно C ++);
– @try: визначає блок з можливою генерацією винятків (аналогічно C ++);
– @throw: генерує об'єкт-вимкнення (аналогічно C ++);
– @catch(): оброблює вийнятки, сгенеровані у попередньому блоці try (аналогічно C ++);
– @finally: визначає блок після блоку try, в якому передається управління незалежно від того, було або ні сгенеровано вимкнення;
– @class: скорочена форма оголошень класу (тільки ім'я (аналогічно C ++));
– @selector(method_name): повертає зкомпільований селектор для імені методу method_name;
– @protocol(protocol_name): повертає екземпляр классу-протоколу з ім'ям protocol_name;
– @encode(type_spec): ініціалізує рядок символів, який буде використан для шифрування даних типу type_spec;
– @synchronized(): визначає блок коду, який виконується лише однією ниткою у будь-який визначений момент часу.

2.2. Мова програмування Swift, її переваги та недоліки

Swift – це фантастичний спосіб писати програми для телефонів, для десктопних комп'ютерів, серверів, та й чого-небудь ще, що запускається і працює за допомогою коду. Swift – безпечна, швидка і інтерактивна мова програмування. Swift увібрала в себе кращі ідеї сучасних мов з мудрістю інженерної культури Apple. Компілятор оптимізований для продуктивності, а мова оптимізована для розробки, без компромісів з однієї чи іншої сторони. [3]

Swift доброзичливий по відношенню до новачків в програмуванні. Це перша мова програмування промислової якості, яка також зрозуміла і цікава, як скриптова

мова. Написання коду в Playground дозволяє експериментувати з кодом Swift і бачити результат миттєво, без необхідності компілювати і запускати додаток.

Swift виключає великий пласт поширених програмних помилок за допомогою застосування сучасних програмних патернів:

- змінні завжди ініціалізовані до того, як будуть використані;
- індекси масивів завжди перевіряються на out-of-bounds помилки;
- цілі числа перевіряються на переповнення;
- опціонали гарантують, що значення nil будуть явно оброблені;
- автоматичне управління пам'яттю;
- обробка помилок дозволяє здійснювати контрольоване відновлення від непередбачених помилок.

Код на Swift скомпільований і оптимізований, щоб отримувати максимальну віддачу від сучасного обладнання. Синтаксис стандартної бібліотеки спроектований ґрунтуючись на тому, що є найочевидніший і найпростіший спосіб написання коду є найкращим варіантом. Комбінація безпеки і швидкості робить Swift кращим кандидатом для написання програм від рівня "Hello, World!" і до цілої операційної системи.

Swift поєднує виведення типів і патерн-матчінг з сучасним простим синтаксисом, дозволяючи складним ідеям бути вираженим просто і коротко. І як результат не тільки стає простіше писати код, але і читати його і підтримувати так само стає просто.

Swift вже має за плечима роки розвитку, і він продовжує розвиватися, включаючи в себе все нові і нові можливості.

Swift є компільованою мовою програмування. Тобто розробник пише вихідний код і потім, використовуючи компілятор, компілює цей код в керуючу програму. Потім цей файл програми можна завантажити в AppStore і поширювати серед інших користувачів.

Для розробки під IOS в першу чергу необхідна відповідна операційна система Mac OS 10.12 Yosemite або вище. Без Mac OS практично неможливо скомпілювати програму. Дана обставина сильно обмежує можливості розробки, враховуючи той

факт, що Mac OS може гарантовано працювати лише на комп'ютерах самої компанії Apple (iMac, MacBook, MacBook Air, MacBook Pro), а також з огляду на високу вартість цих самих комп'ютерів. Однак на звичайному PC під управлінням ОС Windows або ОС на базі Linux створювати додатки під iOS і Mac OS практично неможливо.

Існують також варіанти з віртуальними машинами, на які встановлено Mac OS, або використання Хакінтош, проте працездатність подібних варіантів не гарантована.

Є і ще один варіант – написання коду в будь-якій доступній операційній системі і компіляція його за допомогою спеціальних сервісів за певну плату або безкоштовно. Але, ясна річ, що комфортабельність подібного підходу дуже низька.

Безпосередньо для самої розробки нам будуть потрібні інструменти мови Swift, текстовий редактор для написання коду, симулятори iPhone і iPad для налагодження програми. Для всіх цих та багатьох інших функцій розробки Apple надає безкоштовну середу розробки XCode.

Чи потрібні реальні пристрої iPhone або iPad для тестування розробляються? За великим рахунком немає, так як XCode надає симулятори для тестування, проте в деяких окремих випадках краще тестувати на реальному смартфоні.

2.3. Інтегроване середовище розробки XCode

Xcode (рис. 2.1) – інтегроване середовище розробки (IDE) компанії Apple, яка надає розробникам інструменти для створення додатків під iPhone, iPad, Mac, Apple Watch і Apple TV. Остання версія – Xcode 12 – доступна для завантаження безкоштовно. Xcode запускається тільки на комп'ютерах з OS X (iMac, Macbook і Mac Mini). Річна ліцензія розробника для публікації додатка в iTunes або Mac OS X App Store коштує $ 99. [4]

Рис. 2.1

Середовище розробки Xcode забезпечує ефективність роботи як невеликих, так і великих девелоперських команд. У Xcode IDE використовується схема поділу даних програми Model-View-Controller (Модель-Представлення-Контролер або MVC) для сегментації кожного шару додатку. Так простіше вносити зміни в код. Наприклад, шар UI розділений інструментами, такими як новий Interface Builder, з його допомогою можна поміщати на екран засоби візуального контролю. Auto Layout дозволяє динамічно управляти презентацією об'єктів для екранів різних розмірів; за допомогою Storyboard зручно розташовуються екрани додатка; режим Preview швидко покаже, як виглядають екрани додатка. Жоден з цих інструментів не зачіпає програмний код, який ви створюєте.

Спершу коди в писалися мовою Objective-C. У червні 2014 Apple представила Swift, нову мову для створення мобільних додатків. Це самий швидко освоюваний мову в порівнянні з іншими мовами програмування. Людям з Apple було потрібно досить багато часу, щоб розробити Swift. Як підсумок всіх зусиль, з'явилася мова, який розробникам освоїти набагато простіше, ніж той же Objective-C. До того ж допускається присутність в одному проекті як Swift, так і Objective-C.

Xcode 12 - це радикально швидка версія, в ній міститься практично все, що потрібно для розробки додатків під всі пристрої Apple. Зокрема, нові редакторські розширення. Опція Runtime Issues сповіщає про дефекти, які автоматично виявляє Xcode. Thread Sanitizer відстежує зміну даних та інші баги. Перевірку інтерфейсу здійснює View Debugger - оновлюється інструмент з високою візуальною точністю. Memory Debugger сповіщає про «витоки пам'яті» і приховані баги.

З Xcode можуть працювати індивідуальні розробники. Програмний код перевіряється в репозиторії Git, після чого їм можна ділитися з іншими. Підтримується концепт безперервної інтеграції та інструменти тестування. У поточній версії Xcode також присутній інструмент Test Assistants – забезпечує коректність коду і тестів; інструмент для тестування Test Navigator; підтримка для ботів в Xcode Server, які запускаються після перевірки коду в елементі, є засоби перевірки продуктивності, асинхронності і UI-тестів.

Щоб розмістити додаток, створений в Xcode IDE, в iTunes App Store, знадобиться ліцензія розробника, яка надасть доступ в iTunes Connect, інструмент для розміщення додатків. Для корпоративних додатків iTunes Connect не потрібен, але знадобиться сертифікат, щоб зареєструвати кожен додаток перед публікацією в вашому особистому магазині.

Інтерфейс Xcode інтегрує редагування коду, проект користувацького інтерфейсу, управління активами, тестування і налагодження в єдиному вікні робочої області. Наприклад, виберіть файл в одній області, і відповідний редактор відкривається в іншій області. Виберіть символ або об'єкт призначеного для користувача інтерфейсу, і його документація з'являється в сусідній області (рис. 2.2).

Можна фокусуватися на завданні шляхом відображення тільки того, в чому Ви потребуєте, такий як тільки Ваш вихідний код або тільки Ваш макет інтерфейсу користувача. Або можна працювати з кодом і розташуванням UI поруч. Можна далі налаштувати середовище шляхом відкриття багаторазових вікон і багаторазових вкладок на вікно.

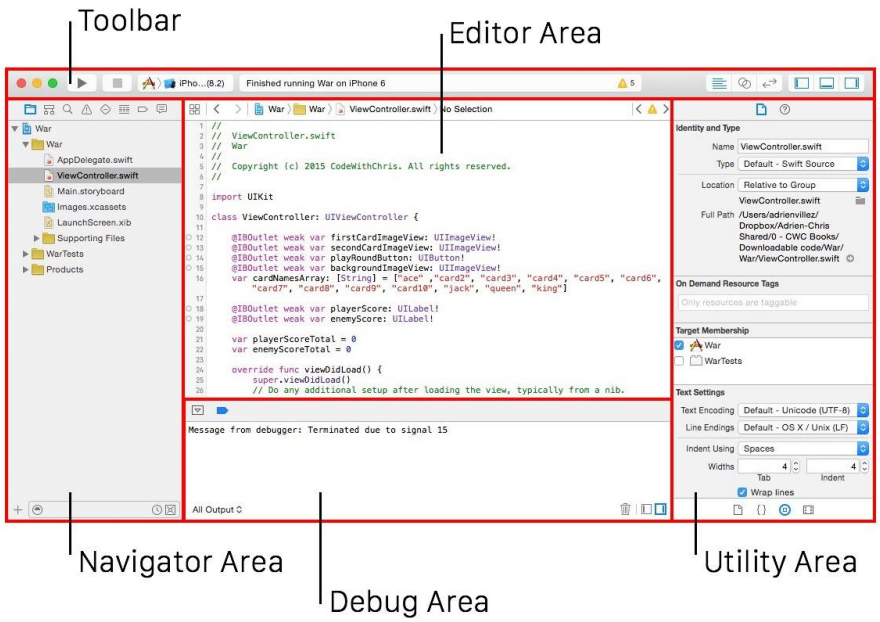

Рис. 2.2

2.4. UI-фреймворки

Cocoa та Cocoa Touch – це середовища розробки додатків для OS X та iOS відповідно. Обидві Cocoa і Cocoa Touch включають виконання Objective-C і два основних фреймворка:

– Cocoa, що включає основи Foundation і AppKit, використовується для розробки програм, що працюють на OS X.

– Cocoa Touch, що включає основи Foundation та UIKit, використовується для розробки програм, що працюють на iOS.

Фреймворк Foundation реалізує кореневий клас NSObject, який визначає базову поведінку об’єкта. Він реалізує класи, що представляють примітивні типи (наприклад, рядки та числа) та колекції (наприклад, масиви та словники). Foundation також забезпечує можливості для інтернаціоналізації, збереження об'єктів, управління файлами та обробки XML. Ви можете використовувати його класи для доступу до основних системних сутностей та служб, таких як порти, потоки, блокування та процеси. Foundation заснована на основі Core Foundation, яка публікує процедурний (ANSI C) інтерфейс.

Фреймворки AppKit та UIKit використовуються для розробки користувальницького інтерфейсу програми. Ці два фреймворки еквівалентні за призначенням, але специфічні для платформи. Вони включають класи для обробки подій, малювання, обробки зображень, обробки тексту, типографіки та передачі даних за допомогою взаємодії. Вони також включають елементи інтерфейсу користувача, такі як подання таблиці, повзунки, кнопки, текстові поля та діалогові вікна попереджень.

Objective-C - рідна, основна мова для розробки додатків Cocoa та Cocoa Touch. Однак проекти для додатків Cocoa та Cocoa Touch можуть включати код C ++ та ANSI C. Крім того, ви можете розробляти додатки какао, використовуючи мови сценаріїв, які прив'язані до середовища виконання Objective-C, такі як PyObjC та RubyCocoa.

Основні особливості Cocoa Touch включають:

- Core Animation: допомагає створити багатий досвід користувачів, забезпечуючи плавне переміщення візуальних елементів. Він також заповнює проміжні кадри анімації з автоматичним синхронізацією та регулюванням;
- Core Audio;
- Core Data: надає об’єктно-орієнтоване рішення для управління даними та допомагає у визначенні моделі даних програми логічним та графічним способом.

Cocoa Touch складається з декількох каркасів, але ключовими є:

Audio and Video, Core Audio, OpenAL, Media Library, AV Foundation, Data Management, Core Data, SQLite, Graphics and Animation, Core Animation, OpenGL ES, Quartz 2D, Networking and Internet, Bonjour, WebKit, BSD Sockets, User Applications, Address Book, Core Location, Map Kit, Store Kit**.**

Cocoa Touch надає основні фреймворки для розробки додатків на пристроях під управлінням iOS. Деякі з них:

- iAd Framework – бібліотека для реалізації сервісів контекстної реклами iAd всередині додатків;

– Game Kit Framework – бібліотека для взаємодії з сервісом Game Center;
– UIKit Framework ґрунтується на Application Kit - бібліотека, яка містить специфічні для iOS GUI-класи;
– Foundation Kit Framework – основна бібліотека, яка містить класи з префіксом NS;
– MapKit Framework – бібліотека, що здійснює взаємодію з картами і навігаційними можливостями iOS-пристроїв.

2.5. Особливості фреймворку UIKit

Фреймворк UIKit забезпечує необхідну інфраструктуру для ваших програм iOS або tvOS. Він надає архітектуру вікна та перегляду для реалізації вашого інтерфейсу, інфраструктуру обробки подій для доставки Multi-Touch та інших типів вхідних даних у ваш додаток, а також основний цикл запуску, необхідний для управління взаємодіями між користувачем, системою та вашим додатком. Інші функції, пропоновані фреймворком, включають підтримку анімації, підтримку документів, підтримку малювання та друку, інформацію про поточний пристрій, керування та відображення тексту, підтримку пошуку, підтримку доступності, підтримку розширень програми та управління ресурсами. [5]

Фреймворк UIKit надає основні об’єкти, необхідні для створення додатків для iOS та tvOS. Ви використовуєте ці об'єкти для відображення вмісту на екрані, взаємодії з цим вмістом та керування взаємодією з системою. Додатки покладаються на UIKit для своєї основної поведінки, і UIKit пропонує безліч способів налаштувати цю поведінку відповідно до ваших конкретних потреб.

Кожна програма UIKit повинна мати такі ресурси:

– іконка програми;
– Launch screen storyboard.

Система відображає значок програми на головному екрані, в налаштуваннях, і в будь-якому місці вона повинна диференціювати свій додаток з інших додатків. Оскільки він використовується в кількох місцях та на кількох пристроях, ви надаєте

кілька версій іконок програми в об’єкті зображення AppIcon вашого проекту Xcode. Іконка вашого додатка повинна відрізнятися, щоб допомогти користувачеві швидко ідентифікувати вашу програму на головному екрані. Однак ви можете змінювати деталі іконки відповідно до різних розмірів зображення, які ви повинні надати.

Файл LaunchScreen.storyboard містить початковий користувальницький інтерфейс вашої програми, і це може бути початковий екран або спрощена версія вашого фактичного інтерфейсу. Коли користувач натискає на іконку вашого додатка, система негайно відображає екран запуску, повідомляючи користувачеві, що ваш додаток зараз запускається. Екран запуску також надає обкладинку вашої програми під час її ініціалізації. Коли ваш додаток готовий, система приховує екран запуску та показує фактичний інтерфейс вашої програми.

Інформацію про конфігурацію та можливості вашого додатка система отримує із файлу списку властивостей інформації (Info.plist, що наведено на рис. 2.3) у вашому наборі програм. Xcode надає попередньо налаштовану версію цього файлу з кожним новим шаблоном проекту, але вам, ймовірно, доведеться змінити цей файл в якийсь момент. Наприклад, якщо ваша програма покладається на конкретне обладнання або використовує певні системні фреймворки, можливо, вам доведеться додати до цього файлу інформацію, пов’язану з цими функціями.

Main storyboard file base name	String	Main
▼ Required device capabilities	Array	(4 items)
Item 0	String	armv7
Item 1	String	front-facing-camera
Item 2	String	location-services
Item 3	String	metal
▶ Supported interface orientations	Array	(3 items)

Рис. 2.3

Однією із загальних модифікацій, яку ви можете внести у файл Info.plist, є декларування вимог до обладнання та програмного забезпечення вашого додатка. Ці вимоги полягають у тому, як ви повідомляєте системі, що потрібно для запуску вашого додатка. Наприклад, навігаційному додатку може знадобитися наявність

обладнання GPS для надання покрокових вказівок. App Store не дозволяє встановлювати програму на пристрій, який не відповідає вимогам вашого додатка.

UIKit надає багато основних об’єктів вашого додатка, включаючи ті, які взаємодіють із системою, запускають основний цикл подій програми та відображають ваш вміст на екрані. Більшість із цих об’єктів ви використовуєте як є або лише з незначними змінами. Знання об’єктів, які потрібно змінити, і коли їх модифікувати, має вирішальне значення для реалізації вашого додатка.

Структура програм UIKit базується на шаблоні дизайну Model-View-Controller (MVC), в якому об’єкти поділяються за призначенням. Модельні об’єкти керують даними програми та бізнес-логікою. Об'єкти перегляду забезпечують візуальне представлення ваших даних. Об'єкти контролера виконують роль моста між вашою моделлю та об'єктами перегляду, переміщуючи дані між ними у відповідний час.

На рис. 2.4 представлена досить типова структура програми UIKit. Ви надаєте об’єкти моделі, які представляють структури даних вашого додатка. UIKit забезпечує більшість об'єктів перегляду, хоча ви можете за потреби визначити власні перегляди для своїх даних. Координація обміну даними між об’єктами даних та поданнями UIKit - це ваші контролери подання та об’єкт-делегат програми.

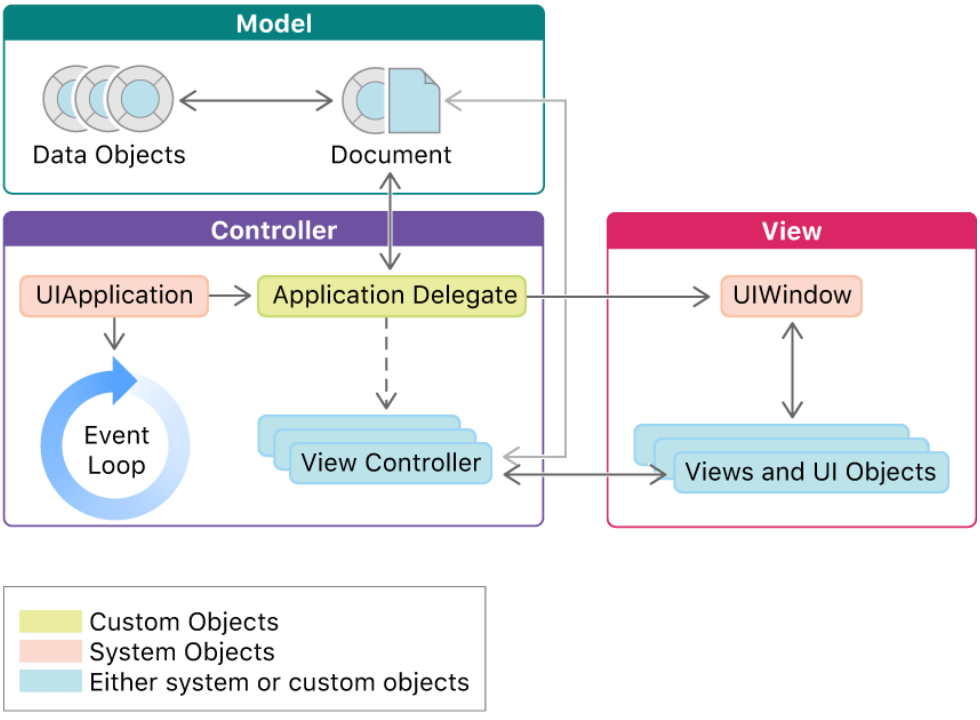

Рис. 2.4

Фреймворки UIKit та Foundation надають багато основних типів, які ви використовуєте для визначення об’єктів моделі вашого додатка. UIKit надає об'єкт UIDocument для організації структур даних, що належать до дискового файлу. Структура Foundation визначає основні об’єкти, що представляють рядки, числа, масиви та інші типи даних. Стандартна бібліотека Swift пропонує багато однакових типів, доступних у рамках Foundation.

UIKit забезпечує більшість об’єктів у контролері та шарах перегляду вашої програми. Зокрема, UIKit визначає клас UIView, який зазвичай відповідає за відображення вашого вмісту на екрані. (Ви також можете візуалізувати вміст безпосередньо на екрані, використовуючи Metal та інші системні фреймворки.) Об’єкт UIApplication запускає основний цикл подій вашого додатка та керує загальним життєвим циклом програми.

2.5. Новий та ефективний фреймворк SwiftUI, його особливості

SwiftUI - це інноваційний, надзвичайно простий спосіб побудови користувацьких інтерфейсів на всіх платформах Apple за допомогою потужності Swift. Створюйте інтерфейси користувача для будь-якого пристрою Apple, використовуючи лише один набір інструментів та API. Завдяки декларативному синтаксису Swift, який легко читати та писати природно, SwiftUI безперебійно працює з новими інструментами проектування Xcode, щоб тримати ваш код та дизайн ідеально синхронізованими. Автоматична підтримка динамічного типу, темного режиму, локалізації та доступності означає, що ваш перший рядок коду SwiftUI - це вже найпотужніший код інтерфейсу, який ви коли-небудь писали. [6]

SwiftUI використовує декларативний синтаксис, тому ви можете просто вказати, що повинен робити ваш користувальницький інтерфейс. Наприклад, ви можете написати, що вам потрібен список елементів, що складається з текстових полів, а потім описати вирівнювання, шрифт і колір для кожного поля (рис. 2.5). Ваш

код простіший і легший для читання, ніж будь-коли раніше, що економить ваш час та обслуговування.

```
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```

Рис. 2.5

Цей декларативний стиль застосовується навіть до складних понять, таких як анімація. Легко додайте анімацію майже до будь-якого елемента керування та виберіть колекцію готових до використання ефектів лише з кількома рядками коду. Під час виконання система обробляє всі кроки, необхідні для створення плавного руху, і навіть має справу з перериваннями, щоб підтримувати стабільність програми. За допомогою цієї простої анімації ви будете шукати нові способи оживити свою програму.

Xcode включає інтуїтивно зрозумілі засоби проектування, які роблять побудову інтерфейсів за допомогою SwiftUI таким простим, як перетягування та скидання. Під час роботи в дизайнерському полотні все, що ви редагуєте, повністю синхронізується з кодом у сусідньому редакторі(рис. 2.6). Код миттєво видно як попередній перегляд під час введення тексту, і будь-які зміни, які ви внесете до цього попереднього перегляду, негайно з’являються у вашому коді. Xcode миттєво перекомпілює ваші зміни та вставляє їх у запущену версію програми, видиму та постійно редаговану.

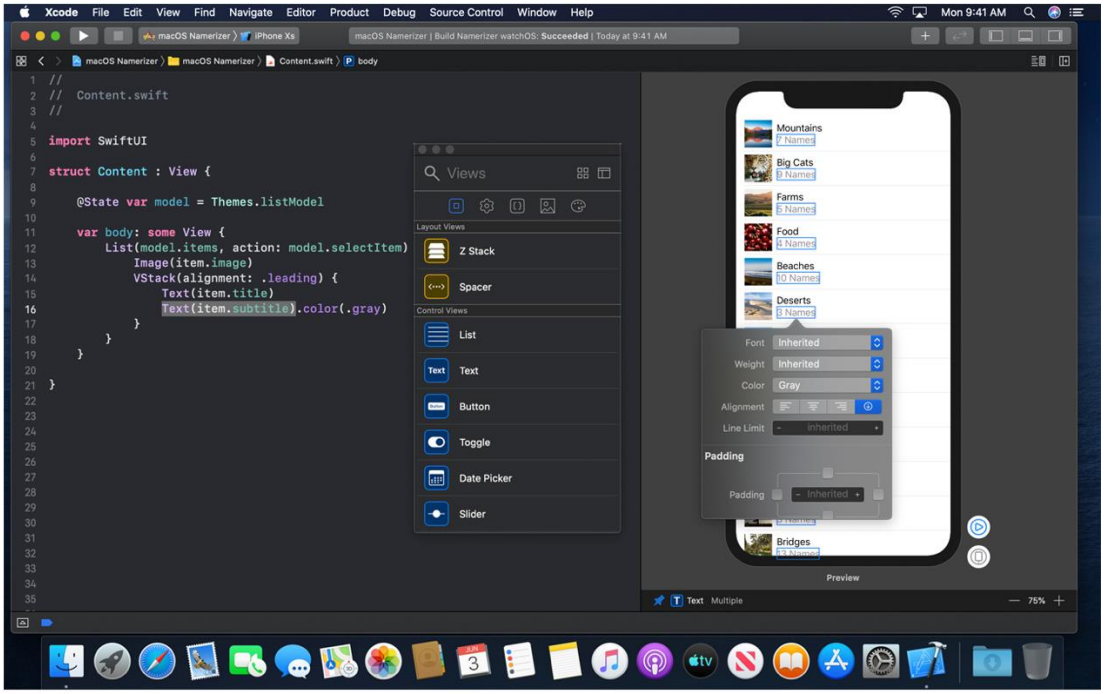

Рис. 2.6

Drag and drop. Розташуйте компоненти у своєму інтерфейсі, просто перетягуючи елементи керування на полотні. Клацніть, щоб відкрити інспектор, щоб вибрати шрифт, колір, вирівнювання та інші варіанти дизайну та легко перевпорядкувати елементи керування за допомогою курсору. Багато з цих візуальних редакторів також доступні в редакторі коду, тому ви можете використовувати інспектори для виявлення нових модифікаторів для кожного елемента керування, навіть якщо ви віддаєте перевагу ручному кодуванню частин вашого інтерфейсу. Ви також можете перетягнути елементи керування з бібліотеки та опустити їх на полотно дизайну або безпосередньо на код.

Dynamic replacement. Компілятор Swift і час роботи повністю вбудовані в Xcode, тому ваш додаток постійно створюється та запускається. Полотно дизайну, яке ви бачите, – це не просто наближення вашого інтерфейсу користувача – це ваш додаток, який працює в режимі реального часу. А Xcode може міняти відредагований код безпосередньо у вашому додатку в режимі реального часу за допомогою "динамічної заміни", нової функції в Swift.

Preview. Тепер ви можете створити один або кілька попередніх переглядів будь-яких подань SwiftUI, щоб отримати зразкові дані, і налаштувати майже все, що

можуть бачити ваші користувачі, наприклад великі шрифти, локалізації чи темний режим. Попередній перегляд також може відображати ваш інтерфейс на будь-якому пристрої та будь-якій орієнтації.

SwiftUI був побудований на багаторічному досвіді створення найбільш інноваційних та інтуїтивно зрозумілих користувацьких інтерфейсів у світі. Все, що сподобалось користувачам в екосистемах Apple, наприклад, елементи керування та досвід роботи на платформі, чудово представлено у вашому коді. SwiftUI справді нативний, тому ваші програми безпосередньо отримують доступ до перевірених технологій кожної платформи(рис. 2.7) за допомогою невеликої кількості коду та інтерактивного дизайну.

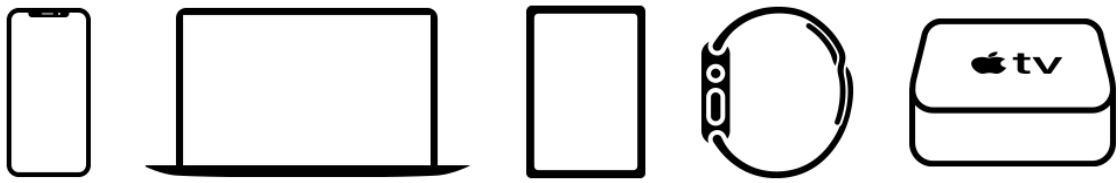

Рис. 2.7

Висновки

Swift – це надійна і інтуїтивно зрозуміла мова програмування від Apple, за допомогою якої можна створювати додатки для iOS, Mac, Apple TV і Apple Watch. Вона надає розробникам небувалу свободу творчості. Завдяки цьому простому і зручному мови з відкритим кодом вам досить просто цікавої ідеї, щоб створити щось неймовірне.

Xcode – це додаток для Mac, призначений для розробки інших додатків для Mac і iOS. У Xcode є всі інструменти, необхідні для створення приголомшливих додатків. Його можна завантажити безкоштовно з Mac App Store.

SwiftUI – це абсолютно новий фреймворк, який дозволяє проектувати і розробляти користувальницькі інтерфейси декларативно та з меншим кодом. На відміну від UIKit, який зазвичай використовується спільно з storyboard, SwiftUI

повністю заснований на програмному забезпеченні. Однак синтаксис SwiftUI дуже легко зрозуміти, і проект SwiftUI можна швидко переглянути за допомогою автоматичного попереднього перегляду(Automatic Preview).

SwiftUI був створений для використання на різних платформах для створення додатків з меншим кодом, ніж UIKit, але з такою ж складністю.

Недоліки SwiftUI:

- він підтримує лише iOS 13 та Xcode 11. Переключившись на них, ви відмовляєтесь від користувачів старих версій iOS, що є радикальним кроком, позбавленим занепокоєння користувача. Але оскільки Apple щороку оновлює свій список підтримуваних версій iOS, я думаю, що SwiftUI буде використовуватися більше протягом наступних двох років, коли користувачі встановлюватимуть останню версію iOS;
- він ще дуже молодий, тому даних про переповнення стеку небагато. Це означає, що ви не можете допомогти вирішити складні проблеми;
- він не дозволяє досліджувати ієрархію подань у попередньому перегляді Xcode.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ РЕЛАКС-ЦЕНТРУ ВИКОРИСТОВУЮЧИ ФРЕЙМВОРК SWIFTUI

3.1. Патерн MVVM та його компоненти

MVVM (Model-View-ViewModel) – це шаблон, який з'явився для обходу обмежень патернів MVC і MVP, і об'єднує деякі з їх сильних сторін. Ця модель вперше з'явилася в складі фреймворка Small Talk в 80-х, і була пізніше покращена з урахуванням оновленої моделі презентацій (MVP). [7]

Патерн MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатки від візуальної частини (подання). Даний патерн є архітектурним, тобто він задає загальну архітектуру програми.

Даний патерн був представлений Джоном Госсманом (John Gossman) в 2005 році як модифікація шаблону Presentation Model і був спочатку націлений на розробку додатків в WPF. І хоча зараз цей патерн вийшов за межі WPF і застосовується в самих різних технологіях, в тому числі при розробці під Android, iOS, проте WPF є досить показовою технологією, яка розкриває можливості даного патерну.

MVVM складається з трьох компонентів: моделі (Model), моделі подання (ViewModel) і уявлення (View), що представлено на рис. 3.1.

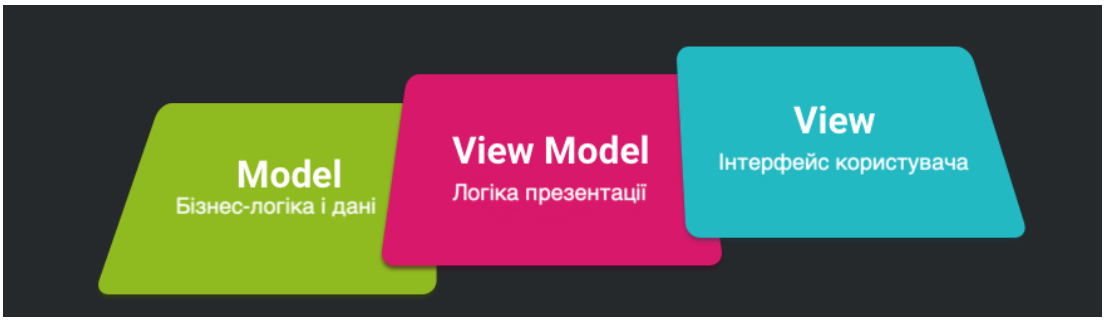

Рис. 3.1

Model описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізує інтерфейси INotifyPropertyChanged або INotifyCollectionChanged, які дозволяють повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча знову ж пряму взаємодію між моделлю і представленням відсутня.

View або подання визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF уявлення – це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас Window) в WPF може містити як інтерфейс в xaml, так і прив'язаний до нього код C #, проте в ідеалі код C # не повинен містити якийсь логіки, крім хіба що конструктора, який викликає метод InitializeComponent і виконує початкову ініціалізацію вікна . Вся ж основна логіка додатки виноситься в компонент ViewModel.

Однак іноді в файлі пов'язаного коду все може знаходитися певна логіка, яку важко реалізувати в рамках патерну MVVM у ViewModel.

Подання і не виконує жодних події за рідкісним винятком, а виконує дії в основному за допомогою команд.

ViewModel або модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані.

ViewModel також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. І також VewModel визначає логіку по оновленню даних в моделі.

Оскільки елементи уявлення, тобто візуальні компоненти типу кнопок, не використовують події, то уявлення взаємодіє з ViewModel за допомогою команд.

Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду під ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

3.2. Розробка основних View та реалізація бізнес-логіки

3.2.1. Розробка головної сторінки

View – це тип, який представляє частину інтерфейсу користувача вашої програми та містить модифікатори, які ви використовуєте для налаштування View.

Щоб створити власне View, треба оголосити типи, що відповідають протоколу View. Потрібно реалізувати тіло(body) – обчислювану властивість для забезпечення вмісту View.

На рис. 3.2 ми бачимо, що тіло приймає «some View». SwiftUI покладається дуже сильно на функцію живлення Swift під назвою «opaque return types», які можна

побачити в дії кожного разу, коли ми пишемо «some View». Це означає "один конкретний тип, який відповідає протоколу View".

```
struct MyView: View {
    var body: some View {
        Text("Hello, World!")
    }
}
```

Рис. 3.2

Повернення «some View» має дві важливі відмінності порівняно з просто поверненням «View»:

– ми завжди повинні повертати однаковий тип View;

– хоча ми не знаємо, який тип View повертається назад, компілятор знає.

Перша різниця важлива для продуктивності: SwiftUI повинен мати можливість переглядати представлені нами View та розуміти, як вони змінюються, щоб він міг правильно оновити користувальницький інтерфейс. Якби нам дозволили випадково міняти View, SwiftUI б дуже повільно з’ясовував, що саме змінилося – йому б майже потрібно було відмовитися від усього і почати знову після кожної невеликої зміни.

SwiftUI пропонує ряд елементів інтерфейсу, які також раніше були доступні в UIKit. Однак у SwiftUI вони мають різну реалізацію та взагалі різні назви. Наприклад, UITableView замінено на List, а UILabel перейнято Text. І список можна продовжувати.

Ці елементи найчастіше використовуються у щоденній розробці:

– Text;

– Image;

– HStack;

– VStack;

– List.

Для реалізації даного проекту я обрала архітектурний патерн MVVM, саме тому я маю створити Model, де буде модель потрібних даних, ViewModel, де буде

відтворюватись бізнес-логіка проекту, а також View, тобто користувацький інтерфейс.

На рис. 3.3 показана модель даних для відео-програвача на головній сторінці.

```
7
8  import Foundation
9
10 struct VideoModel: Identifiable, Codable {
11     var id : String
12     var title: String
13     var URL: String
14 }
15
```

Рис. 3.3

На рис. 3.4 представлена модель даних для тексту на головній сторінці.

```
8  import Foundation
9
10 struct InfoModel: Codable {
11     var title: String
12     var description: String
13 }
14
```

Рис. 3.4

Такі моделі даних я використала для отримання даних із JSON, що знаходиться на сервері. Ці данні я отримала за допомогою URLSession.

URLSession – це об'єкт, який координує групу пов'язаних мережевих завдань передачі даних.

Клас URLSession та пов’язані з ним класи надають API для завантаження даних та завантаження даних у кінцеві точки, зазначені URL-адресами. Додаток також може використовувати цей API для виконання фонових завантажень, коли програма не запущена або в iOS, коли вона призупинена. Можна використовувати пов’язані URLSessionDelegate та URLSessionTaskDelegate для підтримки автентифікації та отримання таких подій, як перенаправлення та завершення завдання.

Додаток створює один або кілька екземплярів URLSession, кожен з яких координує групу пов'язаних завдань передачі даних. Наприклад, якщо ви створюєте веб-браузер, програма може створити один сеанс на вкладку чи вікно, або один сеанс для інтерактивного використання, а інший для фонових завантажень. Протягом кожного сеансу ваш додаток додає ряд завдань, кожне з яких представляє запит на певну URL-адресу (за необхідністю перенаправляючи HTTP).

JSON (JavaScript Object Notation) – це загальний формат для представлення значень і об'єктів. Його опис задокументовано в стандарті RFC 4627. Спочатку він був створений для JavaScript, але багато інших мов також мають бібліотеки, які можуть працювати з ним. Таким чином, JSON легко використовувати для обміну даними, коли клієнт використовує JavaScript, а сервер написаний на Ruby / PHP / Java або будь-якою іншою мовою.

На рис. 3.5 зображено приклад одного із моїх JSON–файлів, що знаходиться на сервері.


```
[
  {
    "title": "Welcome to NATURA salon!",
    "description": "The NATURA herbal massage salon opens the door to the world of amazing massage techniques and spa rituals, invites you to make an amazing journey inside yourself, discovering new facets and sensations of your body, soul and consciousness.\n\nHerbal massage salon NATURA is the only massage salon in Kiev that uses only natural ingredients in massage and spa treatments, all mixtures and formulations are prepared by hand according to old recipes.\n\nEach person who crosses the threshold of our institution is a long-awaited and very dear Guest for us, who is guaranteed to receive an individual massage or an individually designed massage course. We are a team of specialists that is constantly developing and improving our professionalism, for us massage is not a Job, but the whole Life..."
  }
]
```


Рис. 3.5

У цьому проекті для розробки головної сторінки, яка має назву «Main», для створення його інтерфейсу я реалізувала 4 відео-програвача, кожен із них має зверху підпис(Text), а також додала привітальний текст та короткий опис спа-центру.

Для створення привабливого фону я обрала неяскраві кольори. За допомогою ZStack я розташувала 2 кола, та поставила їх прозорість рівною 0.5.

ZStack – це View, який накладає своїх дочірніх View, вирівнюючи їх по обох осях.

ZStack призначає кожному наступному дочірньому View значення осі z вище, ніж попереднє, що означає, що пізніші View з’являються «поверх» попередніх.

Оскільки головна сторінка повинна мати змогу прокрутки, то усі елементи я розташувала у Scroll View. У режимі прокрутки вміст відображається в області прокручуваного вмісту. Коли користувач виконує жести прокрутки, відповідні платформи, View прокрутки регулює, яку частину основного вмісту видно. ScrollView може прокручувати горизонтально, вертикально або і те, і інше, але не забезпечує функцію масштабування.

За допомогою циклу я вивела відео-програвачі на екран.

3.2.2. Робота з відео-програвачем

За допомогою WKWebKit мені вдалося створити відео-програвач.

Об'єкт WKWebView – це власна платформа, яка використовується для безперебійного включення веб-вмісту в інтерфейс вашого додатка. Web View підтримує повний перегляд веб-сторінок і представляє вміст HTML, CSS та JavaScript поряд із View програми. Його можна використовувати, коли веб-технології задовольняють вимоги до оформлення та стилю вашого додатка легше, ніж View. Наприклад, ви можете використовувати його, коли вміст програми часто змінюється.

Web View пропонує контроль над навігацією та взаємодією з користувачем за допомогою об'єктів-делегатів. Делегат навігації потрібен для того, щоб реагувати, коли користувач натискає посилання у веб-вмісті або взаємодіє із вмістом таким чином, що впливає на навігацію. Наприклад, ви можете заборонити користувачеві переходити до нового вмісту, якщо не виконуються певні умови. Використовуйте делегат інтерфейсу користувача, щоб представити власні елементи інтерфейсу, такі як попередження або контекстні меню, у відповідь на взаємодію з вашим веб-вмістом.

Фреймворк WebKit можна використовувати, щоб інтегрувати насичено оформлений веб-вміст у власний вміст програми. WebKit пропонує повний досвід перегляду вашого вмісту, пропонуючи власний перегляд платформи та підтримуючи класи для:

– прояву багатого веб-вмісту за допомогою HTML, CSS та JavaScript
– обробки поступового завантаження вмісту сторінки

– Відображення декількох типів MIME та складених елементів кадру
– переходу між сторінками вмісту
– керування переліченим списком нещодавно відвіданих сторінок

Представте об’єкт WKWebView із ваших ієрархій власного подання та завантажте вміст, який потрібно відобразити. Використовуйте допоміжні об’єкти для управління файлами cookie, оцінки сценаріїв, керування навігацією, створення знімків та виконання текстових пошуків.

3.2.3. Розробка сторінки меню та отримання даних з JSON

Для створення сторінки меню мені знадобився масив даних. Для цього мені потрібно дві моделі даних. На рис.3.6 зображена модель даних для секцій. А на рис. 3.7 зображена модель даних для комірок, що знаходяться у секціях.

```
8  import Foundation
9
10 struct SectionModel: Codable {
11     var title: String
12     var items: [SectionItemModel]
13 }
14
```

Рис. 3.6

```
8  import Foundation
9
10 struct SectionItemModel: Codable, Equatable, Identifiable {
11     let id: Int
12     var massage: String
13     var price: Int
14     var description: String
15 }
16
```

Рис. 3.7

Після отримання даних з JSON через URLSession, я за допомогою List вивела дані на екран та у кожну комірку додала Button з картинкою додавання(рис. 3.8).

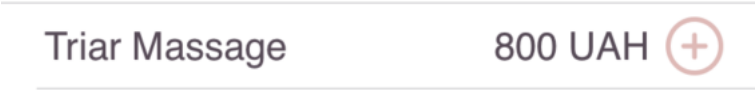

Рис. 3.8

По натисканню на цей Button дані цієї комірки записуються у новий масив, а картинка змінюється на іншу(рис. 3.9). Цей новий масив потрібен для фільтрування даних, які були додані у кошик. Також кожна комірка – це Button, по натисканню на яку виводиться детальна інформація щодо обраної послуги. Ця інформація знаходиться на новому View, що має назву AboutView. По натисканню на кнопку комірки передаються дані з масиву та виводяться на екран.

Рис. 3.9

Також кожна секція має можливість розкриватись. За допомогою DisclosureGroup у SwiftUI це досить легко реалізувати. На рис. 3.10 зображена перша секція, справа є значок, по натисканню на який показуються комірки цієї секції.

Рис. 3.10

Disclosure Group – View, який показує або приховує інший View вмісту, заснований на стані контролю за розкриттям.

Представлення даних групи розкриття складається з мітки для ідентифікації вмісту та елемента керування для показу та приховування вмісту. Показ вмісту

перетворює групу розкриття інформації в «розгорнутий» стан, і приховування їх робить групу розкриття інформації «згортанням».

3.2.4. Розробка сторінки кошику

Сторінка кошику виглядає досить схоже на сторінку меню. Але якщо масив доданих комірок порожній, то на цій сторінці буде відображено текст, що кошик порожній(рис. 3.11).

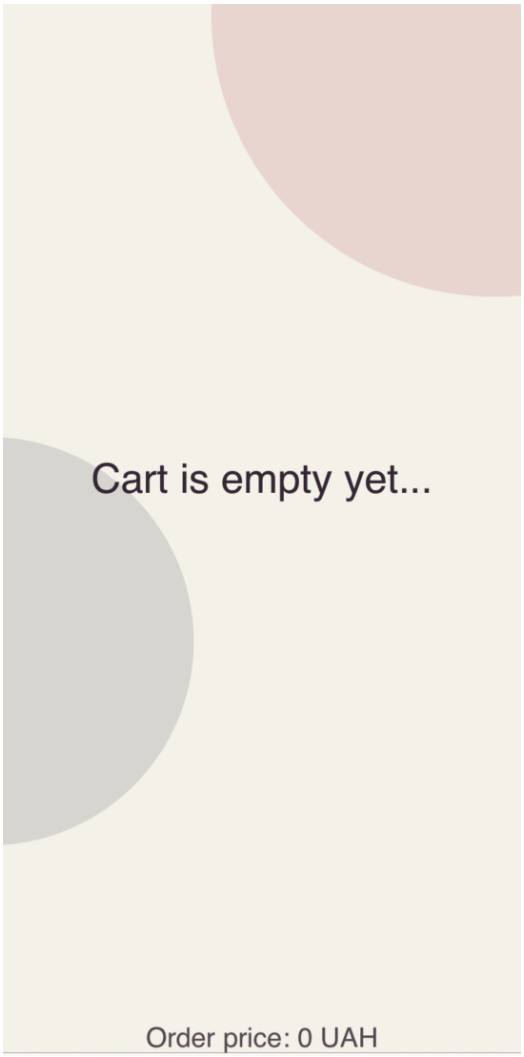

Рис. 3.11

Після додавання комірок у кошик замість відображення тексту буде відображено список доданих комірок. Знизу екрану розташована сума замовлення та

кнопка, натискаючи на яку користувач переходить на нове вікно, де має змогу залишити свої дані для оформлення замовлення.

Такий інтерфейс зручний у використанні та не дуже складний у реалізації.

Модель даних для кошика така ж сама, як і для меню. Видалити обрану послугу можна і у меню, і у кошику, оскільки використовується один масив для цих даних.

3.2.5. Розробка сторінки контактів

Сторінка контактів складається з трьох блоків: телефон, адреса і розклад. Натискаючи на номер телефону(рис. 3.12), на екрані з’являється запит на дзвінок. Після чого можна або подзвонити за номером, або відмінити цю дію.

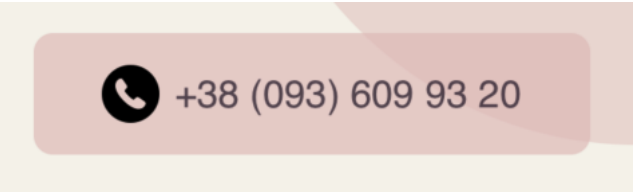

Рис. 3.12

Більше ніякого функціоналу на екрані немає, це лише статичний привабливий текст.

3.2.6. Створення інтерфейсу для введення даних клієнта та оформлення замовлення

Як вже було сказано раніше, після натискання на кнопку знизу на екрані кошика користувач переходить на новий екран, де розташовані два поля вводу для отримання даних користувача, а також кнопка підтвердження. Ці текстові поля мають коментар(рис. 3.13), щоб можна було зрозуміти, які саме дані вводити.

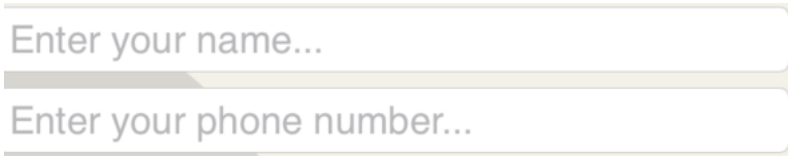

Рис. 3.13

TextField (текстове поле) – елемент керування, який відображає редагований текстовий інтерфейс.

Ви створюєте текстове поле з міткою та прив'язкою до значення. Якщо значенням є рядок, текстове поле постійно оновлює це значення, коли користувач вводить текст, або іншим чином редагує текст у полі. Для нестрокових типів він оновлює значення, коли користувач робить свої зміни, наприклад, натисканням клавіші повернення.

Текстове поле також дозволяє забезпечити два закриття, які налаштовують його поведінку. Властивість onEditingChanged інформує вашу програму про те, коли користувач починає чи закінчує редагування тексту. Властивість onCommit виконується, коли користувач робить свої зміни.

Також, коли користувач обирає текстове поле, де має вписати номер телефону, з’являється клавіатура з цифрами та символами(рис. 3.14), таким чином я не даю змогу вписати некоректні дані.

Рис. 3.14

Після натискання кнопки підтвердження, з’являється Alert(рис.3.15), де сказано, що клієнту зателефонують для підтвердження замовлення, та кнопка «Done», після натискання якої, клієнт автоматично повертається на сторінку кошика, текстові поля стають знов порожніми, масив доданих комірок також стає порожнім. Тобто програма повертається до свого початкового стану.

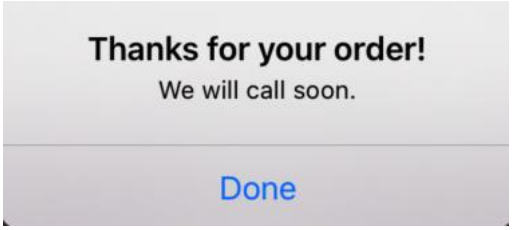

Рис. 3.15

3.2.7. Створення Tab Bar для переходу між вікнами

Для зручного переміщення між екранами я створила Tab Bar.

TabView – це View, який перемикається між кількома дочірніми View за допомогою інтерактивних елементів інтерфейсу користувача.

Щоб створити користувальницький інтерфейс з вкладками, потрібно розмістити View в TabView та застосувати модифікатор tabItem (_ :) до вмісту кожної вкладки(рис. 3.16). У моєму випадку є чотири основних View, тому і tabItem теж чотири. Для привабливого вигляду я додала не тільки текст, а і картинки із бібліотеки.

```
TabView {
    MainView(mainVM: mainVM)
        .tabItem {
            Image(systemName: "house.fill")
            Text("Main")
        }

    ServicesView(servicesVM: servicesVM, cartViewModel: cartViewModel)
        .tabItem {
            Image(systemName: "list.bullet.rectangle")
            Text("Services")
        }

    CartView(cartViewModel: cartViewModel)
        .tabItem {
            Image(systemName: "cart.fill")
            Text("Cart")
        }

    ContactsView(contactsViewModel: contactsViewModel)
        .tabItem {
            Image(systemName: "phone.fill")
            Text("Contacts")
        }
```

Рис. 3.16

В результаті я отримала привабливий Tab Bar(рис. 3.17), завдяки якому користувач має змогу перемикатися між екранами.

Рис. 3.17

3.3. Результати виконаної роботи

У результаті виконаної роботи я розробила свій власний додаток спа-центру з різними користувацькими елементами. Додаток складається з чотирьох основних вікон: MainView, ServicesView, CartView та ContactView, а також двох додаткових вікон: AboutView і CheckoutView.

На рис. 3.18 представлено MainView, де знаходяться відео-програвачі та інформація-опис спа-центру. На екрані є можливість прокрутки, тому продовження екрану зображено на рис. 3.19.

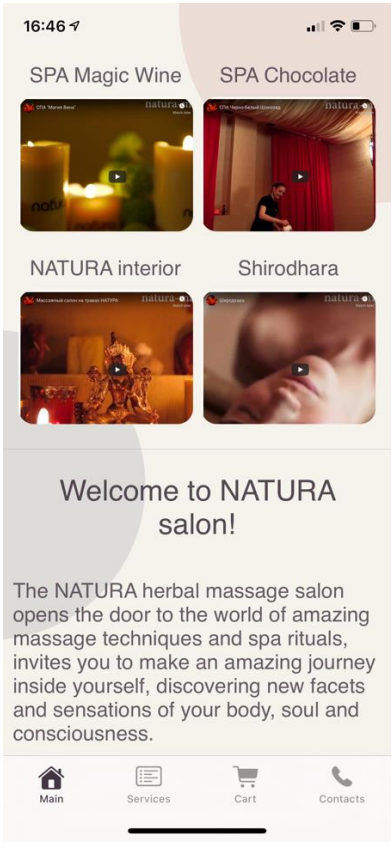

Рис. 3.18

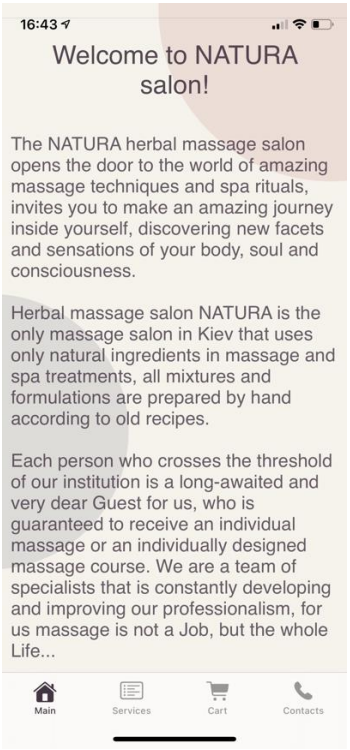

Рис. 3.19

На рис. 3.20 представлено ServicesView, коли секції згорнуті. А на рис. 3.21 секція у розгорнутому стані, де знаходяться комірки, на перших двох я натиснула на кнопку додавання у кошик.

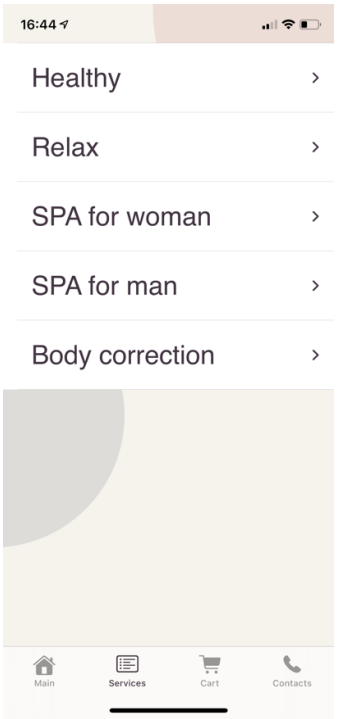

Рис. 3.20

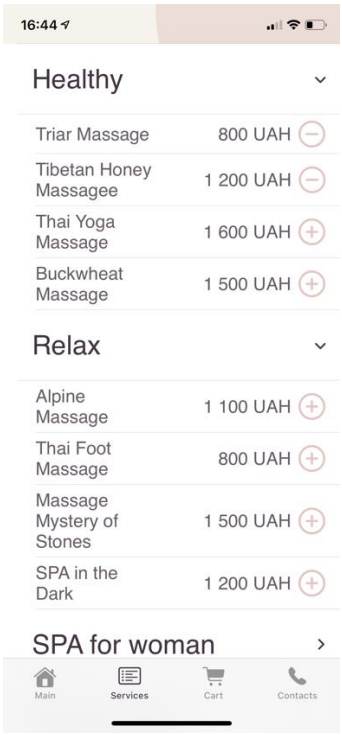

Рис. 3.21

На рис. 3.22 представлено CartView у стані, коли кошик порожній.

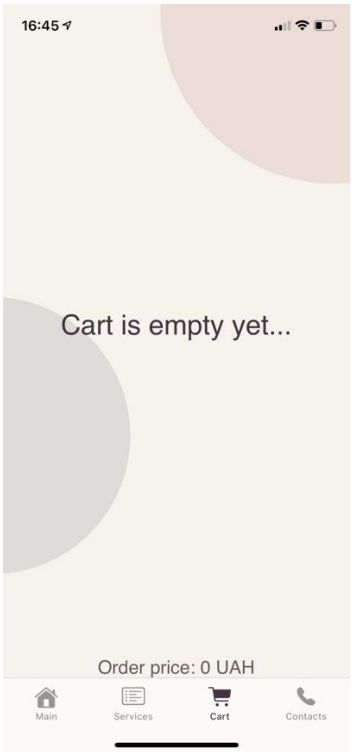

Рис. 3.22

А на рис. 3.23 до кошика було додано дві послуги.

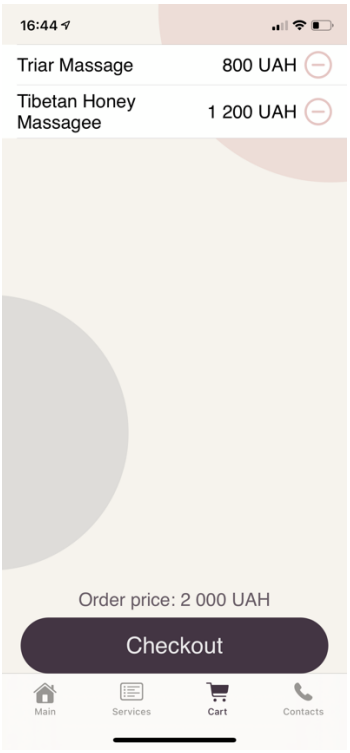

Рис. 3.23

На рис. 3.24 зображено ContactsView.

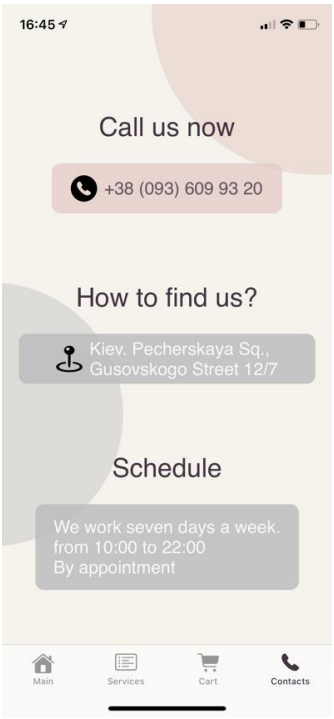

Рис. 3.24

А на рис. 3.25 була натиснута кнопка з номером телефону.

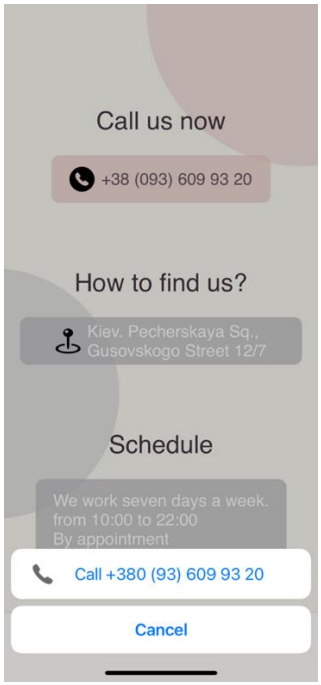

Рис. 3.25

У ServicesView натискаючи на комірку з’являється AboutView(рис. 3.26), де розташована детальна інформація про послугу.

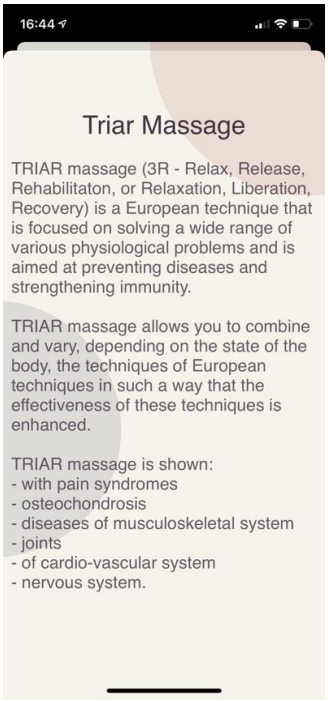

Рис. 3.26

У CartView натискаючи на кнопку «Checkout», користувач потрапляє на новий екран – CheckoutView(рис. 3.27).

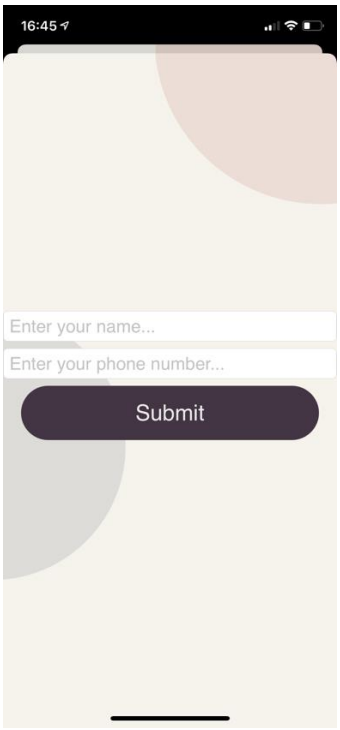

Рис. 3.27

Після введення даних, користувач натискає на кнопку «Submit», після чого з’являється Alert(рис. 3.28).

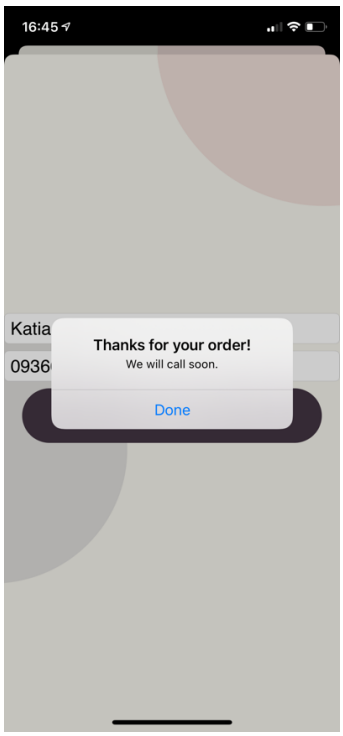

Рис. 3.28

Висновки

Отже, використовуючи архітектурний патерн MVVM та фреймворк SwiftUI мені вдалося розробити мобільний додаток для платформи IOS.

Існують три типи найбільш часто використовуваних архітектурних шаблонів дизайну інтерфейсу, такі як MVC, MVP та MVVM. MVP – це абревіатура від Model-View-Presenter. MVC – це абревіатура Model-View-Controller, тоді як MVVM розшифровується як Model-View-ViewModel. Усі ці шаблони проектування відіграють важливу роль у розробці додатка, оскільки найкращі практики формалізують їх, які вільно поєднуються, простіші для тестування та обслуговування та сприяють багаторазовому об'єктно-орієнтованому розвитку. Ці шаблони архітектури призначені для модерування складних кодів та роблять інтерфейс чистішим та керованішим.

Мобільний додаток, розроблений мною, здатний збільшити продажі компанії, тому що служить засобом залучення і утримання клієнтів. За допомогою вбудованих

функцій можна мотивувати користувачів купувати товари або послуги саме в цій компанії, а також пропонувати різні акції і знижки.

Щоб роздобути нових клієнтів, компанія може запропонувати їм цікавий бонус за установку програми. Це нове свіже рішення, і ніхто ще його не застосовує. Бонуси і знижки люблять все. І мобільні пристрої є практично у всіх. Додаток допомагає запускати нові цікаві маркетингові акції і таким чином вигідно відрізнятися від конкурентів і завойовувати довіру клієнтів.

ВИСНОВКИ

Наявність мобільного додатка невід'ємна частина сучасного ведення бізнесу. Створюючи додаток для iOS і Android значно збільшується кількість потенційних клієнтів. На даний момент кількість користувачів смартфонів у світі становить 59% відсотків дорослого населення, і з них більше 80% використовують смартфони саме на базі Android і iOS.

Останні графіки показують, що браузером користується лише 14% користувачів, в той час як інші 86% для покупки товарів і замовлення послуг використовують саме мобільні додатки. Чому? Бізнес в інтернеті це добре, але не достатньо для сучасних користувачів. Справа в тому, що мобільний додаток має більш широкий і зручний функціонал, адаптивний дизайн і є більш простим у використанні.

Тільки уявіть, наскільки ви збільшите охоплення вашої компанії, кількість продажів і брендинг всього лише створивши додаток. За статистикою, розробка мобільного застосування приносить від 15 до 35 відсотків додаткового доходу.

Swift - це фантастичний спосіб писати програми для телефонів, для десктопних комп'ютерів, серверів, та й чого-небудь ще, що запускає і працює за допомогою коду. Swift - безпечний, швидкий і інтерактивний мову програмування. Swift увібрав в себе кращі ідеї сучасних мов з мудрістю інженерної культури Apple. Компілятор оптимізований для продуктивності, а мова оптимізований для розробки, без компромісів з однієї чи іншої сторони.

Swift доброзичливий по відношенню до новачків в програмуванні. Це перша мова програмування промислового якості, який також зрозумілий і цікавий, як скриптова мова. Написання коду в пісочниці дозволяє експериментувати з кодом Swift і бачити результат миттєво, без необхідності компілювати і запускати додаток.

Swift виключає великий пласт поширених програмних помилок за допомогою застосування сучасних програмних паттернів.

Код на Swift скомпільовано і оптимізований, щоб отримувати максимальну віддачу від сучасного обладнання. Синтаксис стандартної бібліотеки спроектований грунтуючись на керівництві, що найочевидніший і простий спосіб написання коду є найкращим варіантом. Комбінація безпеки і швидкості робить Swift кращим кандидатом для написання програм від рівня "Hello, World!" і до цілої операційної системи.

Swift поєднує висновок типів і патерн-матчінг з сучасним простим синтаксисом, дозволяючи складним ідеям бути вираженим просто і коротко. І як результат не тільки стає простіше писати код, але і читати його і підтримувати так само стає просто.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что такое Apple IOS. Краткий обзор операционной системы IOS для мобильных телефонов [Електронний ресурс]. – Режим доступу: https://mobile-testing.ru/what_is_ios/(дата звернення: 14.05.2021) – Назва з екрану.
2. About Objective-C [Електронний ресурс]. – Режим доступу: https://developer.apple.com/(дата звернення: 15.05.2021) – Назва з екрану.
3. Swift [Електронний ресурс]. – Режим доступу: https://developer.apple.com/swift/(дата звернення: 15.05.2021) – Назва з екрану.
4. Introducing XCode 12 [Електронний ресурс]. – Режим доступу: https://developer.apple.com/xcode/(дата звернення: 18.05.2021) – Назва з екрану.
5. UIKit [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/uikit(дата звернення: 17.05.2021) – Назва з екрану.
6. Introducting SwiftUI [Електронний ресурс]. – Режим доступу: https://developer.apple.com/tutorials/swiftui(дата звернення: 17.05.2021) – Назва з екрану.
7. MVVM: проектирование приложений [Електронний ресурс]. – Режим доступу: https://skillbox.ru/media/code/(дата звернення: 19.05.2021) – Назва з екрану.