МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

"_____"_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Мобільний додаток визначення оптимального шляху по визначним місцям (України)

Виконавець: Паламарчук В. В.

Керівник: Росінська Г. П.

Нормоконтролер: Тупота Є. В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Освітнього ступеня магістр
Спеціальність 123/3 "Комп'ютерна інженерія"
(шифр, найменування)

Спеціалізація 123.02 "Системне програмування"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О.Є.
«_____»_____2020 р.

ЗАВДАННЯ
на виконання дипломної роботи (проекту)

Паламарчука Владислава Вікторовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи Мобільний додаток визначення оптимального шляху по визначним місцям (України)

затверджена наказом ректора від «07» вересня 2020 р. № 1410 /ст.

2. Термін виконання роботи: з 14.10.2020 року по 12.12.2020 року

3. Вихідні дані до роботи: дослідити процес розробки мобільних додатків під різні мобільні системи. Дослідити існуючі програмні продукти та сформувати вимоги до власного програмного продукту.

4. Зміст пояснювальної записки:

1) аналіз проблематики області дослідження;

2) дослідження технологій створення мобільних застосунків;

3) проектування програмної системи.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) зображення алгоритму використання мобільного додатка;

2) макет *UI / UX* мобільного додатка;

3) макет інтерфейсу мобільного додатка;

4) архітектура *MVC* для екранів, які відображають обрані місця та історію попередніх
подорожей;

5) архітектура *MVC* для першого екрану та екрану, який відображає деталі про місце.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомлення з постановкою завдання дипломної роботи	05.10.2020 – 06.10.2020	
2	Аналіз літературних джерел та інтернет-ресурсів	10.10.2020 – 17.11.2020	
3	Дослідження та аналіз існуючих мобільних застосунків з подібним функціоналом	18.11.2020 – 19.11.2020	
4	Проектування архітектури мобільного додатка та інтерфейсу користувача	20.11.2020 – 23.11.2020	
5	Розробка та тестування мобільного додатка	24.11.2020 – 25.11.2020	
6	Підготовка пояснювальної записки	26.11.2020 – 11.12.2020	
7	Підготовка графічного матеріалу	12.12.2020 – 22.12.2020	

7. Дата видачі завдання: 05 жовтня 2020 р.

Керівник дипломної роботи доц. Росінська Г. П.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання Паламарчук В. В.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Мобільний додаток визначення оптимального шляху по визначним місцям (України)»: 90 с., 49 рис., 23 літературне джерело.

Ключові слова: МОБІЛЬНИЙ ДОДАТОК, *IOS*, ДИЗАЙН, АРХІТЕКТУРА *Model-View-Controller*, *USER FLOW*.

Об’єкт дослідження – мобільні застосунки з функцією побудови шляху до визначних місць.

Предмет дослідження – мобільний додаток з можливістю побудови оптимального шляху по визначним місцям.

Метою дипломної роботи є розробити мобільний додаток визначення оптимального шляху по визначним місцям.

Методи дослідження – теоретичне ознайомлення з існуючими мобільними додатками з функцією визначення геопозиції користувача та побудови шляху до заданого місця, моделювання макету дизайну застосунку за допомогою *Adobe XD*, методи створення архітектури мобільного додатка *MVC*, методи об’єктно-орієнтованого програмування, створення програмного коду у середовищі розробки *Xcode*.

Проведено аналіз мобільних застосунків з подібним функціоналом, здійснено порівняльну характеристику, виявлено переваги та недоліки.

Матеріали дипломної роботи рекомендується використовувати у особистих та комерційних цілях, для малих підприємств, тощо.

ЗМІСТ

ВСТУП

Після подорожі складно залишитися таким же, як раніше. Подорожі розширюють кругозір, змінюють наше життя і ставлення до неї, наші пріоритети і цінності. Поїздка в іншу країну – найцікавіший спосіб змінити себе в кращу сторону. Багато хто не підозрює, які зміни може приховувати одна маленька подорож.

Під час подорожі можна отримати не тільки ефект відновлення сил, а дещо більше, повне перезавантаження свого світосприйняття, зміну способу мислення, звичок і навіть характеру.

Подорожі допомагають відкрити в собі те, що було приховано за сірістю повсякденного життя, розширити свою свідомість, реалізувати свої мрії. Подорож має стати невід'ємною частиною життя кожної людини. У кожній поїздці можна завести нові знайомства, нових друзів, партнерів по бізнесу. Завдяки подорожам людина росте та виходить на новий рівень свого особистого розвитку.

Головна користь подорожей в тому, що людина, залишаючи свою звичне середовище перебування, втрачає ті чинники і їх силу, які щодня впливають на нього. Під час відпустки він починає життя з чистого аркуша:

– прокидається і лягає в інший час;

– харчується по-іншому;

– регулярно наповнюється новою інформацією з брошур, екскурсій, спілкування з незнайомцями, вивчає карту, географію, іншу мову;

– починає насолоджуватися естетичною красою архітектури, природи, слухає нову музику.

Найчастіше через брак часу, проблем з бізнесом, грошових питань люди відкладають подорожі на потім. А життя в режимі «робота-дім» створює замкнуте коло. З одного боку у нас немає часу на відпочинок, з іншого боку відсутність відпочинку унеможливлює якісно працювати, отримувати натхнення і будувати плани для досягнення нових здобутків. Тому потрібно терміново вибиратися з цього

кола. І не обов’язково їхати за кордон. Можна виїхати на вихідних за місто, відключитися від буденних думок і зробити повне перезавантаження своєї голови.

А ще, то почуття прекрасного, яке людина отримує під час подорожі спонукає працювати краще і старанніше. І це доведений факт. Згідно з дослідженнями Стенфордського університету почуття прекрасного спонукає людей до думок про загальний добробут. Тобто здатність захоплюватися, бажання оточувати себе красою, естетикою безпосередньо пов’язано з особистим розвитком людини та еволюцією людства в цілому.

Подорожі особливо важливі для дітей, адже саме в ранньому віці закладається характер дитини, коли він максимально допитливий, вчиться у батьків. Під час подорожей дитина знайомиться зі світом, у нього виробляється толерантність до інших народів, культур, рас. І відповідальні батьки це розуміють, тому плануючи відпочинок, складають його таким чином, щоб дитина безпосередньо брала участь у цьому. Якщо дитина разом з батьками буде погоджувати план маршруту, вибирати з запропонованої програми що йому було б цікавіше подивитися, то таким чином у неї виробляється відповідальність за свій вибір, уважність під час екскурсій. Швидше за все вона буде заздалегідь цікавитися тим місцем, куди поїде, вивчати географію, поставити питання про країну. Така поїздка для неї запам’ятовується та стане усвідомленою і максимально корисною.

Також можна використовувати свій досвід з подорожей в житті та навчитися в інших культур. Наприклад, у іспанців і греків можна навчитися правильно відпочивати. Так, відпочивати потрібно ще й вміти. У греків є таке правило не говорити про роботу під час дружніх зустрічей, під час сієсти, на вихідних. Якщо зателефонувати в неробочий час з робочих питань, нехай навіть терміново, то це буде вважатися поганим тоном.

Також під час подорожей можна вивчати мови. Це стане величезною перевагою під час комунікацій та взаємодії з людьми, це дозволить розв’язувати будь-яке питання швидко і легко. Якщо немає часу на курси вивчення мов вдома, можна вибрати такий тип відпочинку, який дозволить поєднати приємне з корисним. На сьогодні є безліч таборів як для дітей, так і для дорослих, де можна

вивчати мову, при цьому спілкуватися з носіями мови і природно ефект від такого роду навчання буде в багато разів швидше.

Смартфон – це стільниковий телефон із можливостями комп'ютера та іншими функціями, які спочатку не були пов'язані з телефонами, такими як операційна система, перегляд веб-сторінок та можливість запуску програмних додатків.

Смартфони можуть використовуватися приватними особами як у споживчому, так і в діловому контексті, і зараз вони майже невід'ємні елементи повсякденного сучасного життя.

Багато споживачів використовують свої смартфони для спілкування з друзями, родиною та брендами в соціальних мережах.

Соціальних медіа, такі як *Facebook, Instagram, Twitter* і *LinkedIn,* створюють свої мобільні додатки, які користувач може завантажити з магазину програм свого телефону. Ці програми дозволяють користувачам смартфонів публікувати особисті оновлення та фотографії.

Іншим поширеним використанням смартфонів є відстеження здоров'я та самопочуття. Наприклад, додаток *Health* для *iOS* може відстежувати поведінку у сні, харчування, вимірювання тіла, життєві показники, вправи на психічне здоров'я тощо.

Сторонні пристрої, такі як розумні годинники, можуть з'єднуватися зі смартфоном для моніторингу даних про стан здоров'я людини, таких як частота серцевих скорочень і надсилати інформацію, яка збирається в телефоні.

Платіж за допомогою смартфону – ще одне широке застосування для смартфонів. Функції гаманця дозволяють користувачам зберігати інформацію про кредитну картку на своїх телефонах для використання під час придбання предметів у роздрібних магазинах. Такі додатки, як *Apple Pay,* також дозволяють користувачам платити іншим користувачам *iOS* безпосередньо зі своїх телефонів.

На підставі звіту про стан цифрової сфери *Digital* 2020 року, який щороку готують *We Are Social* і *Hootsuite,* можна побачити, що кількість користувачів смартфонів збільшилася на 2% в порівнянні з початком року. Таким чином, на кінець 2020 року кількість власників смартфонів склало 67% від загального

населення людства. Така тенденція до зростання значною мірою супроводжує думку про те, що мобільний додаток буде доступно широкому користувачеві.

Наступним кроком став вибір мобільної операційної системи. Ґрунтуючись на даних *Statista* [1], можна сказати, що найбільшу частину ринку, а саме 99,5%, охоплюють операційні системи *Android* і *iOS*. Провівши аналіз інформації про інструменти і особливості розробки для даних операційних систем, вибір був зроблений на користь операційної системи *iOS*.

iOS – мобільна операційна системи, що випускається і розробляється компанією *Apple* на основі операційної системи *OS X* для пристроїв *iPhone*. Інтерфейс *iOS* заснований на концепції прямого маніпулювання з використанням жестів мультитач. Сама компанія *Apple* заявляє про свою операційну систему як про «найдосконалішою мобільної ОС у світі» [3], приділяючи особливу увагу її виключній функціональності.

Згідно з даними, вказаними на офіційному сайті розробників мобільних додатків для пристроїв під управлінням операційної системи *iOS* [4], 81% *iOS*-пристроїв використовують версію операційної системи не нижче *iOS* 13. Таким чином, слід було приділити увагу особливостям розробки *iOS*-додатків для версій *iOS* 13 і вище.

З усього вищесказаного випливає, що метою дипломної роботи на тему «Мобільний додаток визначення оптимального шляху по визначним місцям (України)» є вивчення особливостей мобільної розробки і створення мобільного *iOS*-додатка, який дозволяє зробити подорожування більш приємним та зручним заняттям.

Для досягнення мети дипломної роботи необхідно було виконати наступні завдання:

– дослідження та аналіз існуючих мобільних застосунків з подібним функціоналом;

– вивчення особливостей проектування архітектури та інтерфейсу мобільного додатка;

– проектування архітектури мобільного додатка та інтерфейсу користувача;

– розробка та тестування мобільного додатка.

Створення подібного продукту тягне за собою дослідження і пізнання в таких предметних областях, як мережеві протоколи передачі даних, програмування для мобільних пристроїв на мові *Swift*, розробка дизайну для мобільних застосунків, проектування інтерфейсів мобільних *iOS*-додатків.

Об’єкт дослідження – мобільні застосунки з функцією визначення геопозиції користувача та побудови шляху до визначних місць.

Предмет дослідження – мобільний додаток визначення оптимального шляху по визначним місцям.

У дипломній роботі було використано такі методи дослідження:

– теоретичне ознайомлення з існуючими мобільними додатками з функцією визначення геопозиції користувача та побудови шляху до заданого місця;

– моделювання макету дизайну застосунку за допомогою *Adobe XD*;

– методи створення архітектури «клієнт-сервер»;

– методи створення архітектури мобільного додатка *MVC*;

– методи об’єктно-орієнтованого програмування;

– створення програмного коду у середовищі розробки *Xcode*.

Практичне значення отриманих результатів. Матеріали дипломної роботи рекомендується використовувати у побутових та комерційних цілях, де необхідно мати список із визначними місцями міста та мати можливість швидко та зручно побудувати шлях подорожі по них. А також мати історію з минулих подорожей та мати можливість зберігати улюблені місця в окремий список.

Особистий внесок випускника. Особисто було здійснено аналіз існуючих систем збереження інформації, які реалізують подібні функції, здійснено їх порівняльну характеристику, виявлено переваги та недоліки. Було вивчено особливості проектування архітектури та інтерфейсу мобільного додатка. Розроблено та проведено тестування створеного мобільного додатка.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Як смартфони змінили наше життя

Менш десятиліття назад смартфон став незамінною річчю в нашому повсякденному житті. Для багатьох з нас смартфон – це остання річ, яку ми бачимо перед сном, і перша – яку беремо в руки, прокидаючись. Ми використовуємо його для зв'язку з людьми, спілкування, розваг і пошуку дороги. Ми купуємо і продаємо речі з його допомогою. Відзначаємо місця, куди ми ходили, розповідаємо про те, що робимо.

Незважаючи на те, що смартфон всюди з нами, він не такий простий, як здається. Він з'явився в нашому житті раптово і повністю змінив її. Щоб дійсно зрозуміти міру цих змін, нам потрібно зробити крок назад, до того моменту, коли ми жили в світі без смартфона в руці.

У 2006 році вчені Університету Кейо і група *People and Practices* корпорації *Intel* провели дослідження. Вони визначили речі, які мешканці Лондона, Токіо і Лос-Анджелеса найчастіше носять в сумках, в кишенях і в гаманцях.

«Найчастіше люди носили з собою речі, що нагадують про сім'ю, друзів і близьких. Речі релігійного значення, їжу для перекусу, предмети особистої гігієни, жуйки. Також у багатьох були ключі, посвідчення особи, проїзні квитки».

На момент проведення дослідження мобільний телефон використовувався тільки для того, щоб зв'язатися з кимось або відправити смс. І, звичайно ж, у всіх в кишенях були гроші.

Зараз же можна сказати, що смартфон замінив багато з цих речей. Фотографії рідних, квитки, цифрові копії паспортів – все це поміщається в мобільний телефон. Ця єдина платформа поглинула більшість інших речей, які люди колись носили в кишенях і гаманцях.

Найбільш очевидно, що смартфон замінив звичайні телефони, що призвело до зникнення з вулиць телефонних будок. А ті, що залишилися, тепер використовують, скоріше, як місце для реклами.

Смартфон витіснив плеєри, радіо, все портативні засоби, які ми використовували для доступу до новин і розваг. Квитки і проїзні так само на шляху зникнення, а з ними, можливо, і різні ключі. Паспортами і водійських прав поки вдається встояти під натиском смартфонів. Але хто знає, наскільки довго це триватиме. Навіть документи починають переміщуватися в телефон. Починаючи з 6 лютого 2020 року Міністерством цифрової трансформації України було запущено мобільний застосунок Дія (скорочення від «Держава і я»). Наразі додаток включає в собі такі електронні документи як внутрішній і міжнародний паспорти України, водійське посвідчення та свідоцтво про реєстрацію транспортного засобу, а також студентський квиток. За планами Мінцифри список документів з часом буде доповнюватися.

Як бачимо, з безлічі речей, які ще 12 років тому людям доводилося носити в кишенях, залишилися тільки жуйки, їжа і бальзам для губ. Тому можна сказати, що смартфон – це пристрій, який містить мобільні додатки з різним функціоналом для спрощення життя.

Звичайно, час тече по світу з різною швидкістю. Деякі з нас до сих пір вважають за краще спілкуватися з касиром в банку, але те, що смартфон замінив багато процесів, свідчить про тенденцію до дематеріалізації. Тому при вигляді телефонних будок та інших речей з минулого ми відчуваємо подив або здивування.

Якими б незграбними вони не здавалися, нам зараз важливо, що кожна з таких речей має на увазі весь спосіб життя – взаємопов’язану екосистему торгівлі, практики і досвіду. І оскільки ми замінили ці «екосистеми» новими (побудованими на основі смартфона), структура повсякденному житті не могла не змінитися. Багато процесів помістилися в єдиному пристрої, прибравши з нашого життя безліч процесів.

Якщо раніше для того, щоб досліджувати незнайоме місто, потрібно було носити з собою карту і будувати по ній маршрути, то зараз все стало простіше.

У кожного в смартфоні є, безкоштовна карта з високою роздільною здатністю кожної частини Землі, і це саме по собі є епохальним розвитком.

У наш час з’явилися перші карти в історії людства, які слідкують за нашими рухами і говорять нам, де ми знаходимося на них в реальному часі. Вони допомагають нам позбутися від страху, який заважає багатьом з нас досліджувати незнайомі шляхи або райони.

1.2. Стан цифрової сфери у 2020 році

Те що цифрові технології, мобільні пристрої і соціальні мережі стали невід’ємною частиною повсякденного життя людей у всьому світі підтверджує також звіт про стан цифрової сфери *Digital* 2020 року, який щороку готують *We Are Social* і *Hootsuite* [2].

Перед тим, як ми детальніше розглянемо тенденції, ось основні заголовки, щоб зрозуміти глобальний «стан цифрових технологій» у жовтні 2020 року (рис. 1.1, 1.2):

– загальна кількість населення світу – 7,81 млрд. людей;

– кількість людей, які користуються мобільними телефонами у всьому світі: 5,20 млрд;

– глобальні користувачі Інтернету – 4,66 млрд;

– користувачі соціальних мереж у всьому світі – 4,14 млрд.

Щоб розглянути ці цифри в перспективі, зараз понад дві третини світу користуються мобільним телефоном (67%), тоді як майже 60% усіх людей на Землі зараз користуються Інтернетом.

Тим часом лише декілька місяці тому *We Are Social* і *Hootsuite* у звіті «*Digital* 2020 *July*» повідомили, що проникнення соціальних мереж подолало позначку в 50%, але останні дані показують, що ця цифра вже зросла майже до 53%. Більше 9 із 10 користувачів Інтернету у всьому світі підключаються через мобільні пристрої, але дві третини все ще підключаються через комп’ютери.

Рис. 1.1. Звіт «*Digital* 2020 *October*»

Рис. 1.2. Порівняльний звіт «*Digital* 2020 *October*»

Однак, коли справа стосується соціальних мереж, користувачі явно віддають перевагу мобільним пристроям. Аналіз *KepiOS* показує, що 99% користувачів соціальних мереж у світі отримують доступ через мобільні пристрої, але лише кожен п’ятий використовує ноутбук або настільний комп’ютер.

Тим часом, в Україні за даними *We Are Social* і *Hootsuite* у звіті «*Digital* 2020 *Ukraine*» кількість користувачів Інтернету зросла на 1,5 мільйона (+5,7%) за 2019 і 2020 роки (рис. 1.3). Але на відміну від світових показників менше половини з них активно використовують соціальні мережі.

Рис. 1.3. Порівняльний звіт «*Digital* 2020 *Ukraine*»

Опираючись на порівняльний графік 2019 і 2020 років можна сказати, що тенденція збільшення мобільної індустрії зберігається. Адже за 2020 рік кількість мобільних користувачів збільшилася на 102 мільйони.

1.3. Огляд існуючих програмних рішень

1.3.1. *Google Maps*

Google Maps – це комплекс програм, створених на базі безкоштовного сервісу картографії та технології, що використовується *Google* (рис. 1.4). Цей додаток використовується для пошуку інформації на карті з відмітками пам’яток, організацій та інше.

Рис. 1.4. Інтерфейс *Google Maps* на *Android*

Google Maps являється кроссплатформеним сервісом і має схожий інтерфейс та функціонал для *iOS*, *Android* та веб-версій (рис. 1.5). Далі наведені основні можливості *Google Maps*.

Пошук. Список функцій застосунку містить пошук за адресами та підприємствами або точками інтересів за допомогою потужної утиліти локального пошуку *Google*, рейтингів та відгуків. Він синхронізує пошук та вибране при вході в систему за допомогою *Google*.

Карти та навігація в режимі офлайн. Одна найкращих функцій *Google Maps* – це можливість використовувати карти та навігацію без підключення до інтернету. В останній час офлайн-режиму в додатках значно поліпшено, користувачі можуть зберігати певні ділянки карт на пристрої, щоб прокладати маршрути в автономному режимі. При цьому працює навігація, пошук адреси, *POI* та іншої інформації. На жаль, в режимі офлайн недоступні маршрути громадського транспорту, а також побудова велосипедних та пішохідних маршрутів.

Рис. 1.5. Інтерфейс *Google Maps* на *iOS*

Вказівки. Додаток надає голосові, покрокові вказівки, включаючи подорож на поїзді, автобусі, метро або пішки. Він також пропонує виявлення та уникнення заторів.

Перегляд вулиць та зображення. Додаток *Maps* пропонує зображення на вулиці та сенсорні панорами місць по всьому світу. Він має глобальні супутникові знімки з високою роздільною здатністю, а також тісний зв’язок із земною програмою *Google* для обробки зображень та картографування *Google Earth.*

Транзитна, пішохідна та велосипедна навігація. Карти *Google* вже давно мають вбудовані маршрути громадського транспорту. Велосипедна навігація теж чудова, пропонуючи вам найбезпечніші маршрути з використанням велосипедних доріжок та велодоріжок. Карти *Google* навіть вказують місця, де ви повинні ходити на велосипеді, наприклад, через мости з вузькими пішохідними смугами.

Швидкість і простота. Карти *Google* використовують векторну графіку, щоб допомогти картам рендеритися та масштабуватися швидше, ніж растрова графіка.

Google управляє одними з найпотужніших і найшвидших центрів обробки даних у світі, і це з'являється в додатка *Maps* із надшвидким пошуком даних. Додаток також швидко обчислює та перераховує напрямки.

Інтерфейс відкриття. Знайомий рендеринг карти з кількома піктограмами, які контролюють значний набір функцій програми, включаючи пошук, покрокові вказівки та швидкий доступ до даних про дорожній рух, напрямків громадського транспорту, супутникових зображень та *Google Earth*. Існує досить багато навігаційних програм із захараченими інтерфейсами, тому це велике досягнення.

Розмовна назва вулиці, покрокові вказівки. Вимовна назва вулиці, покрокові вказівки є основою будь-якої навігаційної програми. Додаток швидко розраховує напрямки, забезпечуючи оптимальні маршрути та назви вулиць чітким і приємним голосом. Ви можете переглядати напрямки в традиційному поданні карти за допомогою синьої стрілки та лінії маршруту або за допомогою текстового списку напрямків, посиленого стрілками.

Google Планета Земля. Якщо ви не знайомі з програмою *Google* Планета Земля, це програма, яка дозволяє досліджувати детальні супутникові зображення земної кулі та тривимірні перспективи одним пальцем. Параметр меню дозволяє вибрати *Google* Планета Земля, яка також є безкоштовною програмою, щоб отримати різні точки зору на ваші пункти призначення.

Загалом, програма *Google Maps* для *iPhone* є швидкою, стрункою та точною. Користувачі важкої навігації оцінять її швидкість і точність.

1.3.2. *Apple Maps*

Apple Maps був запущений в 2012 році, і якийсь час він був єдиним вибором для користувачів *iPhone,* які шукають напрямки (рис. 1.6). Тепер в *App Store* з'явилося безліч альтернативних безкоштовних *GPS*-додатків, таких як *Google Maps, MapQuest* і *Waze*. *Apple Maps* доступний лише для операційних системах *Apple,* тобто на *iOS, iPadOS, WatchOS* та *MacOS*.

Рис. 1.6. *Apple Maps* на пристроях *Apple*

Карти *Apple* почали невдало. Початкові версії були повні помилок та інших проблем, що змусило *Apple* шукати виправлення. Але з того часу додаток змінився і тепер пропонує безліч корисних функцій, які допоможуть дістатися до місця призначення, незалежно від обраного транспорту. У *iOS* 14 додаток додав кілька нових опцій, які полегшать ваші подорожі (рис. 1.7). Далі наведені основні можливості *Apple Maps*.

Улюблені місця. З *iOS* 13 або новішої версії, *Apple Maps* дозволяє зберігати ресторани, магазини, компанії, адреси друзів та інші місця як обране.

Путівник. Чим більше місць відвідано та збережено, тим важче знайти місце у вибраному. Організуйте збережені місця, помістивши їх у путівник (раніше відомий як колекції, але ребрендований в *iOS* 14 як путівник).

Путівники по містах. З *iOS* 14 *Apple* також почала створювати путівники для надання туристичної та туристичної інформації про певні міста. На даний момент вибір обмежений лише Нью-Йорком, Сан-Франциско, Лондоном та Лос-Анджелесом.

Шукати місця поблизу (рис. 1.8). Коли ви отримуєте доступ до функції пошуку, подають список найближчих підприємств та інших місць, які ви, можливо, захочете відвідати. Ви можете знайти місцеві ресторани, заправні станції, кав’ярні, продуктові магазини, готелі, бари, торгові центри та лікарні.

Рис. 1.7. Інтерфейс *Apple Maps* на *iOS*

Перегляд вулиць. Ви можете перевірити своє місце призначення ще до того, як дістатися до нього за допомогою функції *Apple Maps Look Around,* представленої в *iOS* 13. Подібно до перегляду вулиць *Google, Look Around* забезпечує інтерактивний 360-градусний огляд місця.

Проліт. У застосунку є можливість побачити з висоти пташиного польоту певне місто за допомогою *Flyover*. Необхідно ввести у полі пошуку назву підтримуваного міста, наприклад Нью-Йорк, Париж, Лондон, Флоренція чи Токіо.

Велосипедні маршрути. Нове в *iOS* 14 – це опція, яка може направити вас до місця призначення велосипедними маршрутами. Отримавши вказівки, торкніться значка велосипеда. Якщо таким чином можна дістатись до пункту призначення, вказівки підкажуть, як їздити на велосипеді. Для певних міст можна навіть отримати вказівки по справжній велодоріжці.

Карти приміщення. *Apple Maps* дозволяє переглядати внутрішні карти торгових центрів, аеропортів та інших місць, де вам може знадобитися допомога в обігу. Шукайте за назвою міста для торгового центру, що підтримується, наприклад, Лос-Анджелеса, Бостона, Нью-Йорка чи Токіо, або за назвою аеропорту, наприклад, Міжнародний аеропорт Гонконгу, аеропорт *JFK*, аеропорт Мельбурна чи аеропорт Цюриха.

Рис. 1.8. Путівник в *Apple Maps*

Інформація про транзит у реальному часі. У *iOS* 13 і новіших версій ви можете бачити актуальну інформацію про транзит поїздів, автобусів та іншого громадського транспорту. Після пошуку маршрутів торкніться значка автобуса. Додаток покаже вам будь-які варіанти місцевого громадського транспорту з деталями про те, де і як його зловити, якою лінією взяти та чи є затримки.

1.3.3. *Kiev Offline Map and Travel Trip Guide*

Розглянемо додаток, який допомагає туристам та всім охочим дізнатися та відвідати визначні місця в Києві (рис. 1.9). *Kiev Offline Map and Travel Trip Guide* доступний лише на платформі *iOS*.

Рис. 1.9. Інтерфейс *Kiev Offline Map and Travel Trip Guide* на *iOS*

Далі наведені основні можливості *Kiev Offline Map and Travel Trip Guide*.

Доступність додатка без мережі Інтернет. Для іноземців це можливість заощадити гроші на вартості роумінгу. Функціонал додатка майже не обмежується в такому режимі використання.

Детальні карти. Векторна карта, яку можна збільшити на будь-якому рівні. Також, присутні безкоштовні безперервні оновленнями від *OpenStreetMap*. Назви вулиць та місця відображаються як місцевою мовою, так і мовою телефону за замовчуванням.

Наявність пошукової системи. За допомогою пошукової системи можна зорієнтуватися на місцевості, а також доступні тисячі місць для відкриття та відвідування: визначні пам’ятки, ресторани, магазини та заклади нічного життя.

Розширений маршрут та навігація. Карта відображається як в *2D*, так і в *3D* режимах. Застосунок будує найкоротший та безпечний маршрут для пішоходів, велосипедистів та автомобілів із зазначенням відстані та часу прибуття.

Можливість бронювання готелів. Користувач має можливість переглянути та забронювати готелі для зручної подорожі. Цей функціонал доступний лише з використанням Інтернет.

1.4. Висновки до розділу

У даному розділі було розглянуто вплив смартфонів на людей та стан мобільної індустрії в 2020 році. Ця інформація дає нам зрозуміти, що смартфони стали невід’ємною частиною повсякденного життя людей. А також тенденція до зростання наявності мобільних пристроїв значною мірою супроводжує думку про те, що мобільний додаток буде доступним широкому користувачеві.

Також, були розглянуті існуючі мобільні додатки із схожим функціоналом, проаналізовані їхні переваги та недоліки На підставі цього аналізу було зроблено висновок, що наведені мобільні застосунки мають багато корисних для користувача функцій, але не задовольняють завдання дипломного проекту. А саме у них відсутній зручний і простий інтерфейс для побудови маршруту по зарання підготовленому списку із визначними місцями.

РОЗДІЛ 2
ОГЛЯД ПРОГРАМНОЇ ЧАСТИНИ ПРОЕКТУ

2.1. Мобільний додаток

Як було зазначено в попередньому розділі, смартфон – це пристрій, який містить різні мобільні додатки. Але що таке мобільний додаток?

Мобільним додатком можна назвати комп'ютерну програму, створену спеціально для використання в мобільному телефоні, смартфоні або комунікаторі, яка призначена для виконання того чи іншого завдання.

Історія мобільних додатків налічує вже більше десяти років. Відправною точкою для створення мобільних додатків стала поява на мобільному телефоні екрану. Природно, перше програмне забезпечення для телефонів представляло собою вбудовані додатки, які призначалися для виконання конкретних функцій телефону і встановлювалися в пристрій самими виробниками.

2.1.1. *WAP* і перше підключення до інтернету

WAP (*Wireless Application Protocol*) з'явився в 1998 році, і саме він об'єднав інтернет і мобільний зв'язок. Тепер можна було вбудувати в телефон браузер, встановити з'єднання з серверами і отримати дані на пристрій.

Одним з перших телефоном з *WAP*-браузером був *Nokia* 7100, який випустили в 1999 році (рис. 2.1). Тоді ж почали з'являтися компанії, які розробляють продукти спеціально для мобільних пристроїв.

WAP дав людям не тільки гри, але і можливість читати новини з мобільного, користуватися електронною поштою, завантажувати карти і навіть бронювати квитки. Для цього створювалися *WAP*-сайти зі спеціальною розміткою для мобільних екранів. Це були прості сторінки з тексту і посилань, майже без картинок.

Рис. 2.1. *Nokia* 7100 перший телефон з *WAP*-браузером

У той час компанії ще не оцінили телефони як спосіб комунікації з клієнтами, тому таких рішень було катастрофічно мало.

Наприклад, *Ericsson* зробили путівник *Michelin*: через *WAP* була доступна база з 60 000 готелів і ресторанів Європи. Ще кілька прикладів використання *WAP* для бізнесу наведені в *White Paper Nokia* 1999 року:

– клієнти *Deutsche Bank* і *Visa International* могли отримувати інформацію про останні транзакції, переглядати баланс і оплачувати рахунки;

– пасажири авіакомпанії *Finnair* могли бронювати квитки і отримувати інформацію про рейси.

2.1.2. Перша відкрита ОС для розробників

У 2001 *Symbian* стала відкритою операційною системою, і водночас з'явилася *Nokia* 7650, на яку можна було встановлювати додатки від сторонніх розробників (рис. 2.2). Це повинно було стати проривом на ринку, але буму не сталося через складнощі розробки та обмежених можливостей смартфонів.

Рис. 2.2. *Nokia* 7650

У розробників був бідний вибір засобів розробки для *Symbian*. Основна мова *C++* була складною у вивченні і компіляції. Доводилося викручуватися, щоб написати додаток, яке було б сумісно з більшістю пристроїв на *Symbian*. Також багатьох відлякувала необхідність купівлі сертифікатів безпеки для підпису додатків.

В той самий час розвивався ринок *Java*-додатків. Розробка програми на *Java* займала менше часу і підходила для *Windows Mobile, Android, bada, Palm OS* і *BlackBerry OS*. У *Symbian* також підтримувалося підмножина *Java – J2ME,* але

функціональність таких програм була сильно обмежена, тому розробкою на *Java* під *Symbian* практично ніхто не займався.

В *Nokia* ніяк не прагнули допомагати розробникам розвивати ринок. Все було настільки погано, що в 2005 році вийшла *Symbian* 9.1, яка була несумісна з додатками, випущеними для попередніх версій. Кожна програма вимагала доопрацювання.

У розробників не було нормального середовища для створення проектів, більшість використовували *Eclipse*, призначеної спочатку для розробки на *Java*. *Nokia* випустили інструмент для розробки на *C++ – Carbide* на основі *Eclipse*, але більша частина його можливостей була платною. Ліцензія коштувала від 300 до 8000 євро, це сильно впливало на кінцеву вартість програми.

При цьому розвитком самої *Symbian* ніхто особливо не займався, більше уваги приділяли новим дизайнам смартфонів *Nokia*. На ринок виходили нові моделі, а саму ОС оновлювали приблизно раз на рік.

Спроби щось виправити в *Nokia* почали робити тільки в 2009 році. Вони намагалися розв'язати проблеми з недружнім *API,* дати більше можливостей для створення додатків і спростити розробку за допомогою фреймворка *Qt,* потім відкрили вихідний код *Symbian* і оголосили про створення *Symbian Foundation*, що повинно було допомогти популяризувати ОС.

Але це все не допомогло зібрати навколо *Symbian* спільнота розробників і партнерів-виробників смартфонів. В результаті ОС не змогла конкурувати з *iOS* і *Android*.

2.1.3. Вихід *iPhone* і запуск *App Store*

У 2007 році Стів Джобс представив світу перший *iPhone* (рис. 2.3). Тоді багато говорили про можливості нового смартфона, але ніхто не говорив, як цих можливостей вдалося досягти.

Архітектура *iOS* була схожа на *MacOS,* але система була повністю закритою. Джобс не хотів, щоб сторонні розробники могли розробляти програми для *iOS,* і не

збирався відкривати *SDK*. Замість цього він хотів, щоб розробники створювали веб-додатки, і дав можливість створювати браузерні закладки на домашньому екрані. Ми знаємо це з біографії Джобса, яку написав Волтер Айзексон.

Рис. 2.3. Перший *iPhone*

«Повноцінний движок *Safari* вже присутній всередині *iPhone*. Тобто ви можете створювати дивовижні *Web* 2.0 і *Ajax* додатки, які виглядають і поводяться так само, як рідні програми *iPhone*. І вони здатні прекрасно взаємодіяти з його сервісами: дзвонити, відправляти електронні листи, розшукувати розташування в *Google Maps*. І знаєте що? Для цього не потрібен *SDK*!», – Стів Джобс.

Але допитливі зламали файлову систему, почали писати інсталятори для нативних додатків і заодно – самі додатки. Так з'явився джейлбрейк.

Пізніше рада директорів *Apple* все ж переконала Джобса легалізувати сторонні додатки. У підсумку в березні 2008 року *iPhone SDK* став доступний всім бажаючим, а в липні презентували *App Store*. Це означало, що *Apple* бере на себе дистрибуцію продуктів розробки користувачам.

App Store став поштовхом до розвитку індустрії розробки додатків, але проблемою був *Objective-C*. Мова програмування для *iOS* кардинально відрізнявся від популярних тоді скриптових *JavaScript* і *Flash Action Script*. Мало хто хотів витрачати час на вивчення нового синтаксису, адже пристрої на *iOS* займали ще дуже маленьку частку ринку, а основна його частина належала смартфонам на *Symbian*.

На той момент двигуном прогресу були ігри, а розробка додатків – лише розвагою для фахівців в суміжних областях. На продажу ігор вже можна було заробляти, а попиту на розробку сервісів і додатків для бізнесу ще не було. Компанії не були зацікавлені в розробці додатків для клієнтів, тому що:

– бізнес не довіряв мобільним технологіям і ніхто не знав, як їх використовувати на благо компаній;

– існували обмеження смартфонів в відображенні, передачі і прийманні даних;

– додатки не підходили для серйозних завдань через проблеми з безпекою даних.

У 2008 з'явився *Android Market*, а в 2010 – *Windows Mobile Store*. На той час мобільний інтернет став доступнішим, а в *SDK* для різних платформ з'явилися рішення з безпеки та інтеграції. Це був новий етап розвитку в розробці додатків: від розваги для народу розробники перейшли до вирішення завдань бізнесу. Але попереду вже маячила інша проблема.

Швидкий паралельний розвиток *iOS* і *Android* створило двополярну систему, і розробникам потрібно було підтримувати кілька платформ одночасно. З кроссплатформенних інструментів були тільки *Flash* і звичайний мобільний браузер. І то в 2010 році *Apple* відмовилися від підтримки технології *Adobe Flash* в *iOS*.

Розробка одного навіть самого простого рішення під різні платформи вперлася в людські, часові та матеріальні ресурси. Хоча браузерні технології були розвинені досить добре, мобільний веб-розробку гальмувала низька продуктивність смартфонів.

На допомогу прийшли бібліотеки компонентів і фреймворки для створення додатків на *Android* і *iOS* на базі браузерних технологій без використання мов програмування: *Xamarin, Cordova, Phonegap*. З їх допомогою створюється подоба мобільного сайту, зверху накладається платформний код, який транслює виклики від системи до додатка і назад.

Ці інструменти розв'язали проблеми маленьких додатків з невеликим функціоналом і дали можливість бізнесу швидко і за відносно невеликі гроші протестувати свою присутність на мобільних платформах. Далі потрібно було дивитися в бік нативної розробки, тому що гостро стояли проблеми продуктивності, споживання ресурсів, чуйності кроссплатформенних додатків і «чужорідного» дизайну.

2.1.4. Розвиток кроссплатформенних рішень

У 2015 році розробники *Facebook* на конференції *React.js Conf* представили свій інструмент для кроссплатформенних рішень – фреймворк *React Native*. У ньому компоненти програми, написані на *JS,* транслюються в нативні *Android* і *iOS*. Цей інструмент принципово відрізняється від інших систем для створення кроссплатформенних додатків:

– відсутністю *WebView* і *HTML*-технологій;

– відображенням інтерфейсу. У *RN* її виконує ОС пристрою, а не браузер;

– відсутністю додаткової «обгортки» коду – замість неї *JS* взаємодіє з ОС через спеціальний міст. Так в додатка використовуються нативні компоненти інтерфейсу користувача.

Завдяки цим відмінностям додатки, зроблені на *React Native*, максимально схожі на нативну розробку, і у них менше проблем з продуктивністю.

Хоча стало простіше і краще, проблеми все одно залишилися:

– знань одного *JS* все одно не вистачить, тому що роботу з можливостями мобільної платформи потрібно реалізовувати через модулі на нативних мовами;

– *Facebook* постійно змінює архітектуру *RN*, через це нові релізи часто супроводжуються зворотною несумісністю в коді;

– швидкість роботи додатків вище, але все ще не на рівні нативних додатків.

2.1.5. Нативна розробка

Виробники намагаються мінімізувати складності в розробці і намагаються спростити мови розробки. В результаті з’явилися простіші *Swift* і *Kotlin* (рис. 2.4).

Swift представили на конференції *WWDC* в 2014 році. У ньому залишилося багато від *Objective-C*, але він працює за аналогією зі скриптовими мовами. Код визначається типами змінних, а не показниками. Це робить його вивчення легше для тих, хто вже володіє будь-яким скриптовою мовою.

Kotlin з 2010 року розробляла компанія *JetBrains*. Метою було зробити більш лаконічний і просту мову, ніж *Java*, в якому вже накопичився багаж невдалих рішень. З 2017 мова офіційно рекомендується для *Android*-додатків.

Тепер для розробки нативних додатків є дві мови з простим і гнучким синтаксисом, при цьому вони схожі між собою.

Вийшло спрощення не тільки від *Objective-C* до *Swift* і від *Java* до *Kotlin,* але і від *Swift* до *Kotlin* і навпаки. Це набагато більший крок до платформ, ніж розробка додатків на *Web-view*.

2.1.6. Порівняння нативної розробки з кроссплатформенною

Коли є кілька інструментів для вирішення однієї і тієї ж задачі, постає вибір, який з них використовувати. Зараз при створенні мобільного проекту виникає питання, робити додаток нативним або використовувати браузерні кроссплатформені технології.

Є кілька речей, які потрібно враховувати, обираючи підхід в розробці.

Довгостроковість проекту. Потрібно відразу обдумати, чи виникне необхідність розширювати додаток або додавати в нього нові функції. У нативних додатках це зробити легше і швидше, тому що код спочатку враховує всі особливості платформи. У випадку з *Web-view,* щоб додати функціонал, доведеться витрачати час на створення додаткового коду і оптимізацію його роботи.

Рис. 2.4. Порівняння нативних мов розробки *Kotlin* і *Swift*

Якщо проект спрямований на довгостроковий розвиток, нативний додаток – більш вдалий варіант. При цьому правило працює і у зворотний бік – нативна розробка не підходить для вирішення маленьких завдань. Наприклад, простий розділ з правилами легше зробити на *Web-view,* тому що не вийде просто так зробити текст з нумерований пунктами, в мовах навіть немає такого компонента – доводиться «винаходити колесо».

Сценарій взаємодії користувача і додатка. Якщо користувачеві потрібно постійно користуватися додатком, важливі чуйність і нативний інтерфейс.

З нативними додатками зручніше взаємодіяти і користувачеві, і смартфону. Вони швидше реагують і відповідають на дії користувача, їм потрібно менше мережевих ресурсів, а також менше вони навантажують процесор.

Щоб інтерфейс вийшов інтуїтивно зрозумілим для користувача, розробнику потрібно просто слідувати інструкціям *Apple* і *Google*. Розробка нативного інтерфейсу за допомогою *Web-view* займає більше часу, а відмінності все одно будуть помітні.

Чи потрібен доступ до сервісів пристрої. У нативного додатка простіше реалізувати доступ до камери, мікрофона або геолокації. Засобами *Web-view* це теж можливо, але тоді доведеться шукати можливість оптимізувати навантаження на процесор. До речі, це не завжди можливо, в *Web-view* немає готових оптимізованих компонентів і доводиться кожного разу винаходити щось нове. Спочатку для того, щоб працювало, а потім для оптимального використання батареї смартфона.

Бюджет. Вважається, що кроссплатформенна розробка дешевша. Замість двох додатків потрібно розробити всього один. Це працює з простими додатками, але якщо мова йде про складні продукти з безліччю функцій, то нативна розробка може виявитися вигіднішою.

Час. Буває, що в пріоритеті швидкість, а не якість і функціонал. Якщо немає часу чекати і потрібно хоч щось, використовують кроссплатформені рішення.

Коли потрібний додаток, який можна розвивати і яким будуть користуватися часто, краще запланувати більш пізній запуск і розробляти нативно.

Наявність *API* для бекенду. Якщо *API* є, це вже крок в сторону нативної розробки. Необхідність робити *API* збільшує ціну і термін нативної розробки, для *Web-view* його розробляти не потрібно. Але тут знову потрібно відштовхуватися від завдань і планів на цей проект. Щоб розвивати додаток надалі, краще виділити гроші і час на *API*.

Необхідність оновлень. Щоб внести зміни в нативний додаток, його потрібно оновити через публікацію в маркеті. Виходить, що швидко вносити зміни не вийде. Але найчастіше це можна вирішити звичайним плануванням або змішуванням

технологій: основну частину зробити нативною, а розділи, де потрібно часто щось оновлювати – на *Web-view*.

Історія розвитку розробки додатків показує, що браузерні технології спочатку почали використовувати, щоб обійти складності нативної розробки. У реальності немає ніякої кросбраузерності та використання *Web-view* – це частіше «милиця», ніж повноцінне рішення задачі. При цьому в якихось випадках *Web-view* більш придатний для використання. Є розділи, які дійсно слід писати на *Web-view*: ліцензійні угоди, договори та інше.

Розробникам варто виходити із завдань, потреб і можливостей бізнесу та використовувати ті чи інші технології, коли це виправдано. Або використовувати все з урахуванням обмежень так, щоб додаток допомагав бізнесу взаємодіяти з користувачем, а не обмежував комунікацію.

2.2. Розробка мобільних додатків для *iOS*

Операційна система *iOS* була випущена компанією *Apple* в 2007 році. До 2019 року на ній функціонували як *iPhone*, так і *iPad*, але пізніше це змінилося – для *iPad* розробили власну ОС. У цьому розділі розглянемо особливості платформи і розробки мобільних додатків конкретно для *iPhone*.

2.2.1. Специфіка мобільних додатків на *iOS*

Перша і головна відмітна риса *iOS* додатків – те, що парк пристроїв значно менше, ніж кількість смартфонів на *Android* (рис. 2.5). Це означає, що адаптувати зовнішній вигляд мобільного застосування під актуальні на ринки *iPhone* має бути простіше.

Водночас з виходом кожного нового пристрою і оновленням операційної системи переважну кількість мобільних додатків потрібно адаптувати під нові умови. Статистика *App Store* показує, що користувачі *iPhone* охоче оновлюються до

актуальної версії ОС. Тому додатки повинні відповідати її вимогам, наприклад, підтримувати темну тему, представлену в пристроях в 2019 році.

Рис. 2.5. Модельний ряд *iPhone*

На діаграмах нижче видно, що за один місяць з моменту презентації *iOS* 13 до неї оновилося 50% користувачів (рис. 2.6). А за станом на 17 червня 2020 року до останньої версії *iOS* оновилося 92% користувачів (рис. 2.7). Особливо цікаво це виглядає в порівнянні зі статистикою наведеною в наступному розділі про розробку під *Android*.

Екрани сучасних *iOS* пристроїв при цьому мають високу роздільну здатність. Це дозволяє використовувати тонкі шрифти: вони не спотворюються, як це буває на дисплеях низької якості.

Також однотипна архітектура пристроїв дозволяє не проводити додаткових перевірок при старті програми: не потрібно перевіряти наявність камери, *GPS* датчика або акселерометра.

2.2.2. Особливості розробки і створення додатків для *iOS*

І все ж незважаючи на те, що база пристроїв на платформі *iOS* значно менша, ніж на *Android* нюансів створення і розробки додатків для *iPhone* досить багато. Дизайн, актуальний код, використання особливостей смартфона при створенні і розробці додатків для *iPhone* можуть сильно підвищити лояльність користувачів, що призведе до підняття рейтингу додатка в магазині.

Рис. 2.6. Статистика оновлень операційної системи на *iOS* пристроях станом на 15 жовтня 2019 року

При розробці додатків для *iOS* необхідно враховувати не тільки розміри екрану, що важливо при проектуванні дизайну додатка, але і апаратні відмінності кожного пристрою.

Повернемося до діаграм зі статистикою оновлень: можна сказати, що при випуску свіжої версії операційної системи більшість користувачів практично відразу

до неї оновлюється. Але нова версія може сильно відрізнятися від попередньої і деякі функції програми можуть не працювати належним чином чи не працювати взагалі.

Відповідно код потрібно підтримувати в актуальному стані, при цьому *Apple* не надає повного списку всіх нововведень заздалегідь, даючи доступ лише до бета-версії з незавершеним функціоналом, реалізація якого в підсумку може відрізнятися. Тому протягом кількох тижнів після виходу фінальної версії нової ОС потрібно протестувати мобільний додаток і оперативно адаптувати його під нові умови, якщо це необхідно.

До цієї ж тематики можна віднести додавання нових можливостей в останніх версіях, так як прогрес не стоїть на місці і *Apple* намагається бути в тренді, впроваджуючи в свій продукт топові винаходи світу технологій. Наприклад, в *iOS* 13 з'явилася можливість забути про вхід в різні додатки за номером телефону або акаунту в соціальній мережі, не кажучи вже про давно забуті логін та пароль. Тепер можна просто натиснути кнопку «вхід з *Apple ID*» і, не заповнюючи нічого зайвого, потрапити в додаток. Більш того, користувач може вибрати опцію приховування свого *Apple ID,* і ніхто не отримає доступу до його *e-mail*.

Рис. 2.7. Статистика оновлень операційної системи на *iOS* пристроях станом на 17 червня 2020 року

Цей нюанс важливо враховувати не тільки при розробці додатків для *iOS,* але і при маркетинговій стратегії, так як збір таких даних для подальшого їх використання – часта практика в просуванні додатків. Надіслати листа на псевдоадресу, згенеровану *Apple* для користувача, можна через спеціалізований канал, але ось масові розсилки так працювати не будуть, у всякому разі наразі популярні сервіси розсилки не передбачають цю особливість.

Ще однією особливістю розробки мобільних додатків під *iOS* є реалізація динамічного розміру системного шрифту. Користувачі *iOS* пристроїв можуть змінювати розмір шрифту всієї ОС, який буде використовуватися по замовчуванню. Якщо ваш додаток не змінює шрифт динамічно і відображає його весь час

фіксованого розміру, застосунок буде сприйматися як неякісний продукт у цього сегмента аудиторії.

При розробці дизайну також важливо враховувати, що деякі користувачі відкривають додаток в режимі сумісності на планшетах. На деяких моделях *iPad* при цьому дозвіл робочої області стає таким же, як і на старих моделях *iPhone*: 4 і 4*s*. Служба перевірки додатків *Apple* часто робить саме так, навіть призначені тільки для *iPhone* додатки перевіряють на *iPad*.

Це означає, що потрібно враховувати компонування інтерфейсу додатка і на розмірах екрану 960 × 640 пікселів, а не тільки на *iPhone* 6 і вище. Багато елементів можуть накладатися один на одного, текст переноситися некоректно і т.д.

2.2.3. Політика *Apple* по публікації додатків в *App Store*

Після того, як ви додаток розроблений настає момент публікації. *App Store* відрізняється жорсткими вимогами, які необхідно дотримуватися при публікації програми. Важливо не тільки дотримання всіх стильових інструкцій при розробці дизайну, але і забезпечення конфіденційності особистих даних користувача, стабільності додатка під час роботи і, найголовніше, його корисності.

Однією з причин відмови в публікації може послужити те, що перевіряючий вважатиме додаток марним або ж недостатньо корисним на тлі безлічі аналогічних програм в *App Store*. В такому випадку функціонал додатка варто переглянути, додавши в нього в тому числі елементи, що зачіпають нововведення операційної системи. Такі додатки пропускають в магазин краще, заохочуючи розробників і далі підтримувати останні функції ОС.

Варто відзначити і ряд формальних моментів, які дуже важливо врахувати при створенні мобільного додатка для *iOS*:

1. Для розміщення в *App Store* будь-якого продукту потрібно зареєструватися як розробник, сплатити внесок $99 за рік і коректно заповнити всі дані про себе. У разі, якщо ви плануєте випускати продукт як юридична особа, процедура реєстрації буде довше і закладати на цей процес потрібно від тижня до місяця. Тому акаунт

розробника краще створювати відразу ж, як тільки стартувала розробка – це допоможе в момент готовності додатка розмістити його в магазині без зайвих зволікань.

2. Необхідно підготувати матеріали для сторінки додатка. Знімки екрану строго заданого формату і розміру (навіть 1 зайвий піксель відіграє роль і картинки не будуть прийняті), рекламний і загальний опис програми, ключові слова, політику конфіденційності (написати і розмістити файл так, щоб він був доступний за посиланням). А ще потрібно передбачити легкий вхід в додаток для тих, хто буде перевіряти, так як в разі необхідності реєструватися за номером телефону з смс-підтвердженням потрапити всередину і перевірити додаток вони не зможуть, і відразу відхилять його. В такому випадку можна генерувати заданий номер телефону і код, за яким можна увійти в додаток, минаючи отримання смс.

3. Сам додаток може перевірятися від одного дня до тижня. У виняткових випадках цей строк може бути і більше, але, як правило, додаток проходить перевірку протягом декількох днів. Це також варто враховувати, плануючи маркетингову кампанію – варто залишати запас між днем надсилання на перевірку і стартом реклами хоча б на кілька днів.

4. Вимоги *Apple* не обмежуються магазином, наприклад, розміщення на екрані додатка кнопки «*Download on the App Store*» також регламентується правилами, так що пофарбувати її в кольори продукту не вдасться (рис. 2.8) [5].

Хоча правил і умов розміщення чимало, слід просто діяти по вивіреному плану і дотримуватися рекомендацій, тоді ризик зіткнутися з проблемами знизиться до мінімуму.

Рис. 2.8. Правила *Apple* стосовно кнопки «*Download on the App Store*»

2.2.4. Висновки

При розробці *iOS* додатків з нуля важливо враховувати такі особливості:

– різноманіття пристроїв відносно невелике і вони мають однотипну архітектуру;

– разом з тим потрібно забезпечити зручну роботу з інтерфейсом і на невеликих екранах, в режимі сумісності з якими додатки відкриваються на планшетах;

– пристрої оновлюються на останню версію операційної системи швидко, необхідно забезпечити стабільність додатка на різних версіях;

– на всіх актуальних пристроях використовується *Retina* дисплей з високою роздільною здатністю. Можна використовувати тонкі шрифти і елементи та не боятися їх некоректного колірного або контурного відображення;

– при проектуванні інтерфейсу необхідно керуватися *Apple Human Interface Guidelines;*

– при публікації додатка необхідно враховувати всі вимоги *App Store Review Guidelines,* їх багато і вони досить різноманітні;

Список вимог досить великий, крім того, для кожного типу додатків він розширюється в свою специфіку, але дотримання цих принципів розробки дозволяє робити якісні продукти.

2.3. Розробка і створення мобільних додатків для *Android*

Операційна система *Android* була випущена компанією *Google* 23 вересня 2008 року. 11 липня 2005 року корпорація купила стартап *Android Inc,* що займається цією розробкою та перетворила цей напрям в один з ключових. З тих пір *Android* швидко розвивається і тепер встановлений на більш ніж 83% всіх мобільних пристроїв в світі.

2.3.1. Специфіка створення і розробки мобільних додатків під *Android*

По-перше, це велика фрагментація пристроїв. Це прекрасно для користувачів. Можна вибрати телефон на будь-який смак і під будь-які технічні вимоги. Але дуже непросто для розробників і це стосується як апаратної, так і програмної частини.

Апаратно пристрій може мати фронтальну камеру, може і ні. Сімкарт може бути будь-яка кількість. Фізичні кнопки можуть бути присутніми чи ні. Екранів може бути два додатковий з тильного боку або на чохлі. Існуючі елементи також мають різні параметри. Наприклад, датчик акселерометра може бути встановлений по-різному у різних пристроїв (рис. 2.9).

Здавалося б – дрібниця. Але якщо ви робите гру, керовану нахилами пристрою (наприклад, перегони), то спочатку вам треба попросити користувача повертати телефон в руках у заданих напрямках, щоб додаток розпізнав позицію встановленого датчика. Інакше на одному смартфоні для поворотів потрібно буде робити нахили вліво-вправо, а на іншому – вперед-назад.

Рис. 2.9. Розташування осей акселерометра може виявитися не таким, як в документації

Розмір екрану і його роздільна здатність – окрема проблема. Наприклад, якщо вам потрібно розмістити зображення на весь екран *iOS* пристрою, ви використовуєте кілька зображень під типові розміри *iPhone* 6 і вище, *iPhone* 6 *Plus* і вище, *iPhone X, iPhone* 12 і *iPhone X Max*. У випадку ж з *Android* екрани мають і різний дозвіл, і різне співвідношення сторін, і різну щільність (рис. 2.10).

У зв’язку з цим для *Android*-розробників існують різні інструменти, наприклад 9 *Patch* – схема розмітки, що дозволяє задати правила розтягування зображення при зміні його розміру (рис. 2.11). Без неї складно коректно відобразити в тому числі і фонові зображення в зв’язку з різними розмірами екранів.

Рис. 2.10. Схематична демонстрація основних розмірів екранів *Android* і *iOS*

Рис. 2.11. Приклад використання 9 *Patch* в *Android*

Відповідно якщо ви поставили собі за мету заповнити зображенням весь екран, вам потрібно або використовувати кілька картинок і обрізати їх на нестандартних розмірах екранів, або розрізати так, щоб окремі частини утворювали собою одне ціле, але могли зміщуватися відносно один одного (наприклад, земля, хмари, ліва і права частини).

По-друге, користувачі мають різні версії операційної системи *Android*. Це породжує безліч проблем:

1. Необхідно враховувати особливості відображення інтерфейсу на різних версіях ОС та різних оболонках. Так, системні елементи управління можуть виглядати зовсім по-різному на різних версіях *Android* і різних оболонках однієї і тієї ж версії *Android*.

2. Різні версії в ряді моментів мають різну логіку роботи (рис. 2.12). Наприклад, до версії 6.0 програми не повинні були вимагати кожний дозвіл окремо (доступ до камери, мікрофона і так далі), вони вказувалися списком в *Google Play* і, малося на увазі, що користувач ознайомлюється з ними до моменту завантаження. Починаючи з 6.0 кожний дозвіл має бути запитано окремо вже в момент роботи програми. Відповідно, якщо додаток не підтримує обидва варіанти логіки, програма не буде працювати або до версії 6.0, або в більш пізніх.

3. Програмні методи та бібліотеки змінюються якісь з них визнаються застарілими та їх потрібно замінювати на новіші. Таким чином завжди постає вибір або підтримувати останні функції ОС, або дозволити якомога більшій кількості користувачів встановити мобільний додаток (рис. 2.13).

4. В останніх версіях ОС додалася багатозадачність робочої області. Користувач може відобразити на екрані одночасно кілька додатків і вашому може бути виділена довільна за розміром область. Це також необхідно враховувати.

По-третє, це архітектура самого додатка. На відміну від *iOS*, де додатки архітектурно представляють собою щось єдине ціле, в *Android* вони збираються з логічно самостійних і відокремлених частин – «*activity*» і фрагментів (рис. 2.14).

Такий підхід був обраний саме для того, щоб забезпечити роботу додатків на абсолютно різних пристроях, в тому числі з дуже малим об’ємом оперативної пам’яті і слабкими процесорами. Якщо частини програми незалежні, будь-яку з них можна в потрібний момент викинути з пам’яті і не витрачати на підтримку її життєвого циклу дорогоцінні ресурси.

Рис. 2.12. Розподіл версій ОС *Android* станом на 7 травня 2019

Рис. 2.13. Різні теми відображення одного і того ж додатка в *Android*

Наприклад, на екрані додатка відображається список ресторанів, потім користувач натискає на якийсь елемент і провалюється в нього. Другий екран, картка ресторану, нічого не повинна знати про попередній екран списку, тому що в будь-який момент часу, в тому числі відразу після переходу в картку, він може бути вивантажений з оперативної пам'яті. Це станеться, наприклад, якщо в фоні запущено багато додатків або на екрані картки ви починаєте програвати відео в хорошій якості.

Щоб додаток працював коректно і без збоїв, екран картки не повинні звертатися до жодної інформації попереднього екрана, приймаючи на вхід лише певні дані. Якщо, наприклад, у користувача є можливість перейти на наступний ресторан, не повертаючись у список, то картка повинна самостійно отримати необхідну інформацію. У той же час екран списку не повинен нічого знати про саму картку, тому що після повернення з неї вона так само може бути знищена.

Рис. 2.14. «*Activity*» і фрагменти являють собою логічно відокремлені елементи зі своїм життєвим циклом

Цей аспект архітектури додатків звучить занадто технічно, але він дає зрозуміти чому, наприклад, далеко не всі типи додатків можливо реалізувати кроссплатформенно: якщо це щось об'ємне по функціоналу, то воно повністю вивантажується з пам'яті при нестачі місця і на слабких пристроях можливість роботи з ними просто відсутня.

В процесі розробки додатків для *Android* є ряд особливостей:

1. На відміну від *iOS*, мобільні додатки для *Android* являють собою взаємозв'язок окремих, логічно відокремлених елементів, як про це говорилося вище. Тобто не можна просто взяти і перенести додаток на іншу мобільну операційну систему, переписавши код з однієї мови програмування на іншу. Потрібно закладати зовсім іншу архітектуру. Інший підхід спостерігається також і в інших аспектах. Наприклад, сучасна іконка *Android* додатків може мати різну форму в залежності від налаштувань операційної системи (рис. 2.15). Дизайнер повинен це

враховувати і переконатися, що логотип виглядає прекрасно і гармонійно у всіх варіантах.

Рис. 2.15. Іконка додатка для різних налаштувань операційної системи

2. *Material Design* – ціла філософія побудови призначеного для користувача інтерфейсу (рис. 2.16). Офіційна документація по цьому підходу містить сотні документів, детально описують як його принципи, так і конкретні приклади правильного і неправильного використання правил для кожного елемента інтерфейсу. Іконка програми може мати різну форму на різних екранах. Але вона завжди повинна гарно виглядати.

Кнопка, яка виступає над активною областю, повинна бути тільки одна (одного кольору). Не можна використовувати кілька виступаючих кнопок різних кольорів.

Кнопка, панель навігації, іконка і всі інші елементи повинні слідувати цим правилам, якщо ви хочете сконструювати гарний *Material* інтерфейс і отримати пропозицію від *Google* на просування вашого мобільного додатка в *Google Play*.

3. Рекомендованою *Google* мовою програмування для *Android* в даний час є *Kotlin*, не *Java*. Різниця між ними суттєво менше, ніж між *Objective-C* і *Swift* для *iOS*, але все ж це різні підходи до розробки.

Рис. 2.16. Приклад правильного застосування *Material Design*

4. Тестування на великій різноманітності фізичних пристроїв (не емуляторів) при цьому має надзвичайно важливе значення. Навіть воно в силу величезної кількості різних телефонів на ринку не забезпечує безпроблемне функціонування на всіх доступних моделях, але принаймні знижує ймовірність проблем на найбільш популярних пристроях.

2.3.3. Політика публікації додатків в *Google Play*

Перед публікацією в магазин додатків *Google Play* збірки проходять набагато менш ретельний контроль з точки зору дотримання вимог щодо побудови інтерфейсу, вибору тематики і запиту персональних даних користувачів (рис. 2.17).

Незважаючи на те, що *Google* недавно змінив підхід до перевірки додатків, зробивши його ретельнішим і більш ручним, середній час огляд додатків становить 2-4 години. Це істотно швидше, ніж 2-3 дні у випадку з *App Store*.

Рис. 2.17. Загальна схема процесу перевірки додатків

Раніше багато хто цим користувалися з метою розкрутки додатка. В *Android Market*, попередньої версії *Google Play*, була вкладка «Нові додатки», що відображає поновлення в їх хронологічній послідовності. Таким чином, чим частіше ви оновлювали версію свого додатка, тим більше отримували завантажень.

Зараз просування додатків стало набагато складнішим, але в першу чергу потрібно зробити класний продукт, в тому числі з точки зору вимог концепції *Material Design*. Якщо їх не дотримуватися такий додаток зможе рекламуватися в *Google Play*, але у виняткових випадках.

2.3.4. Висновки

Розробка мобільних додатків для *Android* має наступну специфіку:

1. *Android* – найпопулярніша операційна система в світі. Як наслідок, диверсифікація пристроїв керованих нею величезна. Потрібно переконатися в тому, що мобільний додаток підтримує переважна більшість моделей пристроїв цільової аудиторії.

2. *UI* / *UX* повинен враховувати не тільки різні розміри екранів пристроїв, але і роботу в режимі багатовіконності, і щільність пікселів екранів: тонкий шрифт на неякісних дисплеях буде спотворений або зовсім зникне.

3. Кількість актуальних версій *Android*, які перебувають у використанні величезна. Треба враховувати всі з тих, якими користується ваша цільова аудиторія.

4. При проектуванні інтерфейсу слід керуватися концепцією *Material Design*.

5. Рекомендованою *Google* мовою програмування для *Android* є *Kotlin*.

6. Тестування на великому різноманітті фізичних пристроїв – дуже важливо.

7. Налаштувати особистий кабінет розробника і опублікувати додаток можна за 2-3 години.

Розробити якісне мобільний додаток для *Android* непросто, потрібно враховувати основні перераховані і безліч інших чинників. Разом з тим не менш важливим є й вирішити реальну, існуючу проблему людей, і не допустити найчастіші помилки.

2.4. Висновки до розділу

У даному розділі були розглянуті різні особливості розробки мобільних додатків під операційні системи *iOS* і *Android.* На підставі цього, можна сказати, що доступ до апаратних функцій і підтримка сторонніх бібліотек найбільш повно здійснюється в нативному підході розробки мобільних додатків. За продуктивністю нативний підхід також лідирує. Крім того, він надає найбільш зручний в експлуатації призначений для користувача інтерфейс. Однак, нативний підхід поступається іншим за охопленням аудиторії. Але цей недолік втрачає актуальність при розробці програми для декількох операційних систем одночасно. Найбільший відсоток (близько 99,5%) ринку смартфонів охоплюють платформи *iOS* і *Android* і, якщо планувати довгу підтримку проекту та здійснювати реалізацію програми для обох платформ, то можна зупинити свій вибір на нативних технологіях розробки. Проаналізувавши все вищесказане, робимо вибір на користь нативної розробки додатка для операційної системи *iOS*.

РОЗДІЛ 3
ОПИС ПРОЦЕСУ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКА

3.1. Розробка дизайну

3.1.1. Процес розробки дизайну

У створенні мобільного застосунку, найважливішу роль виконує розробка дизайну додатка. Користувачів в першу чергу утримує саме дизайн, його зовнішній вигляд, функціональність, зручність і простота.

Дизайн програми – це візуальне оформлення програми, а також створення структури, заснованої на логіці користувача поведінки. Іншими словами, це не тільки зовнішній вигляд, але і зручність використання. Дизайн додатка можна поділити на *UX* і *UI* частини.

Дизайн мобільного додатка повинен бути не тільки привабливим зовні і функціональним. Користування програмою, має бути інтуїтивно зрозумілим для користувача. Вчинення будь-яких дій в додатка не повинно займати багато зусиль або часу, адже користувачі від такого додатка найімовірніше відмовляться, не бажаючи розібратися в тому, як це працює.

Виявити, проаналізувати, протестувати і впровадити інтуїтивно зрозумілий дизайн, допоможе *UX* (*User eXperience*). Ґрунтуючись на досвіді попередніх програм, додатків конкурентів, *UX* дизайн додатків дозволить виробити найбільш ефективний і інтуїтивно зрозумілий інтерфейс.

Корисний мобільний додаток має бути орієнтованим в першу чергу на клієнта. Допомагаючи вирішити всі необхідні завдання без зайвих зусиль. Тому користувацький інтерфейс повинен бути простим і зрозумілим.

Не менш важливий елемент – це *UI* дизайн програми. *UI* визначає кольори і загальне візуальне оформлення додатка, то наскільки зручно користувачеві буде натискати на кнопки і наскільки читабельним буде текст.

Весь процес розробки дизайну можна поділити на декілька етапів:

1. Насамперед необхідно визначити цілі, що і навіщо потрібно користувачам додатка. Треба обдумати реалізацію з боку дизайну.

2. Створити начерки ескізів дизайну додатка, для подальшого втілення в життя цих ескізів. Заздалегідь опрацювати на папері приблизний зовнішній вигляд програми і функціонал який в ньому буде. Це дозволить уникнути зайвої роботи при повноцінному відображенні екранів.

3. Після того, як функціонал визначений, необхідно створити схему того як цей функціонал буде пов'язаний. Зручніше за все, буде реалізувати цю схему у вигляді пов'язаних між собою блоків різних екранів з описом функцій в кожному з них.

Дана схема має свою назву – *user flow*. Основне призначення створення *user flow* – це опрацювання логіки функціоналу додатка, того як, що і навіщо має працювати (рис. 3.1).

Зазвичай *user flow* складається з трьох типів фігур:

– прямокутники – використовуються для представлення екранів;

– ромби – використовуються для умов (наприклад, натискання кнопки входу в систему, свайп вліво, збільшення);

– стрілки – з'єднують екрани і умови разом.

User flow дуже корисний, тому що він дає логічне уявлення про те, як програма має працювати і вирішувати завдання.

4. *Wireframes* – створення ескізів для всіх екранів (рис. 3.2). Необхідний для створення каркасної структури пов'язаних між собою сторінок додатка, визначити розташування кнопок, іконок, картинок і ярликів. Бажано його створити на етапі створення скетчів.

Найчастіше, дизайнери використовують готові шаблони з сервісів на зразок *UI Stencils*. Це дозволяє їм пристойно заощадити час на малюванні ескізів *wireframes*.

Рис. 3.1. Приклад *user flow*

Рис. 3.2. Приклад прототипу

5. Розробка дизайну додатка і прототипів. Для повноцінної розробки дизайну, раніше використовувався *Adobe Photoshop*. Однак з розвитком технологій, більшість дизайнерів мобільних додатків від нього відмовилися через непотрібність. Зараз найчастіше використовують *Figma*, *Sketch*, *Adobe XD*.

3.1.2. Огляд програми для створення дизайну *Adobe XD*

Adobe Experience Design (*Adobe XD*) – програма розробки інтерфейсів від *Adobe Systems* (рис. 3.3). Підтримує векторну графіку й веб-верстку та створює невеличкі активні протоколи.

Adobe XD створювався як комплексне рішення для дизайну інтерфейсів, і робота над ним велася спільно з спільнотою професійних *UX*-дизайнерів. Дизайнери мислили набагато далі графічної оболонки програми або програми, торкаючись завдань оптимізації та вдосконалення взаємодії користувачів з продуктом. В *Adobe XD* свій внесок внесли понад 5000 дизайнерів в рамках предрелізної програми.

Рис. 3.3. Інтерфейс програми *Adobe XD*

Програма робить багато чого з того, що повинен вміти інструмент діджитал-дизайну, і дозволяє користувачам створювати макети, проектувати і ділитися прототипами. Він підходить для веб-розробки або розробки додатків. Інструмент став відповіддю компанії на проектування в векторному форматі для цифрових пристроїв, а не просто використання *Photoshop* (растровий інструмент).

Дві головні причини популярності інструменту – він працює в середовищі *Mac* або ПК і є безкоштовним для користувачів *Creative Cloud*. Для всіх інших користувачів інструмент пропонує безкоштовний план з обмеженим сховищем і кількістю активних проектів.

Основний функціонал *Adobe XD*:

– інструмент зроблений для дизайну. Є можливість редагувати елементи, створювати стилі або повторювану сітку;

– у *XD* можна відкривати файли з інших продуктів *Adobe* і *Sketch*;

– векторне редагування з точною функціональністю дизайну;

– створення інтерактивних прототипів з анімацією, підтримкою ігор і попереднім переглядом на мобільних пристроях;

– можливість спільної роботи з файлами або обміну в режимі «тільки для перегляду»;

– можливість збереження в хмарі з пов’язаними матеріалами;

– працює з безліччю плагінів;

– є мобільний додаток;

– він виглядає і відчувається як інші продукти *Adobe*, тому користувачі швидше можуть засвоїти інструмент;

– постійні оновлення.

3.1.3. Створення дизайну

Насамперед, я розробив *user flow*, в якому описав основну логіку додатка (рис. 3.5). У ньому міститься поведінка застосунку для головного екрану, екрану з деталями вибраного місця, екрану з картою.

На другому етапі, я роздрукував макет із сервісу *UI Stencils* на якому намалював ескіз для головного екрану, а також визначив загальний стиль користувацького інтерфейсу для всього додатка (рис. 3.4).

Після визначення основної логіки та створення ескізу, я зобразив наявну інформацію на макеті за допомогою *Adobe XD*. Користуючись цією програмою я розробив детальний макет, який містить дизайн всіх екранів застосунку, а також задав логіку переходу між ними. А ще я мав змогу протестувати створений

інтерфейс за допомогою вбудованого функціонала *Adobe XD* (рис. 3.6).

Рис. 3.4. Макет інтерфейсу мобільного додатка

Рис. 3.5. Зображення алгоритму використання мобільного додатка

Рис. 3.6. Макет *UI/UX* додатка у *Adobe XD*

3.2. Розробка архітектури системи

3.2.1. Архітектура «клієнт-сервер»

Архітектура «клієнт-сервер» є обчислювальною архітектурою, де завдання і мережеве навантаження розподілені між програмним забезпеченням. Постачальник ресурсів і послуг – сервер, а замовник ресурсів і послуг – клієнт.

Сервер являє собою сховище даних і надає доступ до цих даних іншим об’єктам мережі за їхніми запитами.

Клієнт – це робоча станція, яка використовує ресурси сервера і надає користувачеві зручний інтерфейс для управління цими ресурсами. Клієнт не повинен мати безпосередніх зв’язків з базою даних для забезпечення безпеки даних і бути навантаженим основною логічною складовою додатка для можливого здійснення масштабування системи [7].

Сервер працює по запитах клієнтів і керує обробкою цих запитів. Після виконання кожного запиту, незалежно від результату обробки, сервер відправляє відповідь клієнту, який послав цей запит. Зазвичай клієнт і сервер розташовуються на різних обчислювальних машинах, але можуть виконуватися також і на одній машині.

Архітектура «клієнт-сервер» дозволяє значно знизити обсяги мережевого трафіку шляхом надання відповіді на запит, а не повної передачі даних.

Існує два різновиди архітектури «клієнт-сервер»:

– дворівнева архітектура – зберігання і обробка даних відбувається на одному і тому ж сервері (рис. 3.7);

– багаторівнева архітектура – функції обробки даних винесені на інші сервера (рис. 3.8).

Використання багаторівневої архітектури дозволяє розділити функції обробки, зберігання та подання інформації для більш ефективного використання можливостей клієнтів і серверів. Найчастіше використання багаторівневої архітектури обумовлено великими обсягами даних або необхідністю проведення складних обчислень.

Рис. 3.7. Дворівнева архітектура «клієнт-сервер»

Рис. 3.8. Багаторівнева архітектура «клієнт-сервер»

У нашому випадку ми будемо користуватися сторонніми сервісами, які в свою чергу використовують багаторівневу архітектуру.

Взаємодія сервера і клієнта відбувається через комп’ютерну мережу за допомогою мережевих протоколів. Існує велика кількість мережевих протоколів, які використовується в мережі Інтернет. Кожен з них призначається для досягнення певних цілей і вирішення конкретних завдань. Розглянемо найбільш відомі та зробимо вибір на користь одного з протоколів, який буде відповідати поставленим цілям і завданням:

– *HTTP* (*HyperText Transfer Protocol*) – протокол передачі даних, який використовується в мережі Інтернет для отримання інформації з веб-сайтів.

– *HTTPS* (*HyperText Transfer Protocol Secure*) – протокол передачі даних через мережу, спрямований на забезпечення безпеки інформації. Протокол *HTTPS* є розширенням протоколу *HTTP*, де передана інформація піддається шифруванню.

– *FTP* (*File Transfer Protocol*) – протокол обміну файлами між комп'ютерами мережі. *FTP* дозволяє як копіювати файли з віддаленого комп'ютера на свій, так і навпаки, зі свого – на віддалений.

– *POP (Post Office Protocol), SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol)* – протоколи для пересилки і отримання поштових повідомлень.

– *SSH* (*Secure Shell*) – протокол, який дозволяє робити керування операційною системою на віддаленому комп'ютері.

Для досягнення поставлених цілей необхідно організувати обмін даними між клієнтської і серверними частинами програми. Таким чином, зупиняємо вибір на використанні протоколів *HTTP* і *HTTPS*. Оскільки принципи передачі даних між частинами системи цих протоколів не мають відмінностей, розглянемо особливості роботи на прикладі протоколу *HTTP*.

3.2.2. Протокол *HTTP*

HTTP (англ. «*HyperText Transfer Protocol*» або протокол передачі гіпертексту) – протокол прикладного рівня передачі даних для спільних, розподілених інформаційних систем (рис. 3.9). Основою *HTTP* є архітектура «клієнт-сервер» і взаємодія клієнтської і серверної частин відбувається у два етапи [8].

1 етап. Клієнт (браузер або мобільний додаток) посилає рядок запиту (*HTTP*-запит), яка повинна бути створена за певними правилами, з проханням отримання необхідних даних з сервера.

2 етап. Сервер отримує і обробляє запит, знаходить запитувані клієнтом дані і формує відповідь клієнту (*HTTP*-відповідь).

Якщо запит був успішно оброблений і запитувана інформація була знайдена, то у відповіді буде передана ця інформація з деякими службовими даними. Якщо стався збій, то клієнт отримає від сервера відповіді з кодом помилки.

Рис. 3.9. Схематичне зображення роботи *HTTP* протоколу

HTTP-запит складається з трьох основних частин:

– рядок запиту (*Request-Line*). Рядок запиту містить у собі інформацію про метод, який застосовується до ресурсу, запитуваний *URI*, версію протоколу. Точний ресурс, ідентифікований запитом, визначається за допомогою інформації запитуваної *URI* (*Request–URI*) і даних поля заголовка *Host*. Якщо *Request-URI* являє собою абсолютний *URI*, то значення *Host* буде ігноруватися;

– поля заголовка запиту (*Request-Header*). Поля заголовка запиту дозволяють клієнту передавати серверу додаткову інформацію про запит і самого клієнта;

– тіло переданого повідомлення (*Entity-Body*). Тіло переданого повідомлення не є обов’язковим параметром і може бути відсутнім.

HTTP-відповідь також містить три основних компоненти:

– рядок стану (*Status-Line*). Рядок стану складається з версії протоколу, числового коду стану і яка б пояснила фрази.

– поля заголовка відповіді (*Response-Header Fields*). Поля заголовка відповіді дозволяють серверу передати додаткові дані про сервер і надалі доступі до ресурсу.

– тіло переданого повідомлення (*Entity-Body*).

Методи взаємодії з ресурсом визначаються функціоналом системи. Для роботи з серверною частиною системи нам знадобляться чотири методи: *GET, POST, PUT, DELETE*.

Метод *GET* дозволяє отримувати будь-яку інформацію, ідентифіковану запитуваною *URI*. Якщо запитуваний *URI* звертається до процесу, який виробляє дані, то як відповідь повинні бути отримані вироблені дані [9].

Метод *POST* використовується для передачі даних сервера. Інформація, включена в запит, передається на обробку ресурсу, ідентифікованого запитуваною *URI*.

Метод *PUT* дозволяє зберігати об’єкти, включені в запит, під запитуваною *URI*. У разі, якщо *Request-URI* робить запит до вже наявного ресурсу, отримані дані будуть розглядатися як модифікована версія об’єкту, що знаходиться на сервері.

Метод *DELETE* використовується для запитів на видалення ресурсів, ідентифікованих запитуваною *URI*.

3.2.3. *Google Maps SDK*

За допомогою *Maps SDK* для *iOS* можна додавати карти на основі даних карт *Google* до своєї програми. *SDK* автоматично обробляє доступ до серверів *Google Maps*, відображення карт і відповідь на жести користувачів, такі як клацання та перетягування. Також можна додати маркери, полілінії, накладення на землю та інформаційні вікна на свою карту. Ці об’єкти надають додаткову інформацію про розташування на карті та дозволяють взаємодіяти користувача з картою.

За допомогою *API* та *SDK Google Maps Platform* надає можливість для створення індивідуальних, гнучких налаштувань відображення карт, які представляють користувачам реальний світ за допомогою статичних і динамічних карт, точних адрес, зображень вулиць та переглядів в режимі 360°.

Перелік доступних *API* та *SDK*:

– *Maps SDK for Android* – додає карти *Google* до *Android*-застосунку.

– *Maps SDK for iOS* – додає карти *Google* до *iOS*-застосунку.

– *Maps Static API* – вбудовує прості зображення карт з мінімальним кодом у веб-сайт.

– *Maps JavaScript API* – додає інтерактивну карту до веб-сайту з опціями налаштування карт за допомогою власного вмісту та зображень.

– *Street-View Static API* – вставляє зображення реального світу з панорамами 360°.

– *Maps URLs* – запускає карти *Google* і ініціює дії, наприклад, пошук або напрямки, використовуючи схему кроссплатформних *URL*-адрес.

– *Maps Embed API* – за допомогою простого *HTTP*-запиту додає інтерактивну карту або перегляд вулиць з панорамами на сайт.

– *Directions API* – надає маршрути для транзиту, велосипедної, автомобільної чи пішої подорожі між кількома локаціями.

– *Distance Matrix API* – розраховує час подорожі та відстань між кількома локаціями.

– *Roads API* – визначає точний маршрут, яким подорожує транспортний засіб.

– *Places SDK for Android* – додає збагачені деталі для мільйонів місць до вашого *Android*-застосунку. Надає автозаповненні результати на запити користувачів. Конвертує між собою значення адрес та географічних координат.

– *Places SDK for iOS* – так само додає деталі, автозаповненні результати запитів, конвертує значення адрес та географічних координат для *iOS*-застосунків.

– *Places Library, Maps JavaScript API* – додає збагачені деталі мільйонів місць до веб-сайту. Надає результати автозаповнення на запити користувачів. Конвертує між собою значення адрес та географічних координат.

– *Places API* – отримуйте актуальну інформацію про мільйони локацій, використовуючи *HTTP* запити.

– *Geocoding API* – конвертуйте адреси в географічні координати і навпаки.

– *Geolocation API* – отримуйте локацію пристрою, не покладаючись на *GPS*, використовуючи дані локацій зі стільникових веж або вузлів *Wi-Fi*.

– *Time Zone API* – отримуйте часовий пояс для певної широти і довготи.

Функціональні вимоги нашого застосунку задовольняють наступні *API*: *Maps SDK for iOS, Places SDK for iOS, Directions API.*

3.2.4 *MediaWiki API*

Для отримання інформації про вибране місце у додатка буде використовуватися *MediaWiki API.* Для того, щоб отримати інформацію про місце, необхідно сформувати запит до *http://uk.wikipedia.org/w/api.php* з наступними параметрами:

– *action*: *query*. Для поля «*action*» необхідно вказати ключ «*query*», щоб отримати дані з *MediaWiki*;

– *format*: *json*. У полі «*format*» вказується тип даних, який прийде у відповідь на запит;

– *pageids*: 740736. У полі «*pageids*» необхідно вказати *id* сторінки, яка містить необхідну інформацію про шукане місце.

У відповідь ми отримаємо дані у *JSON* форматі (рис. 3.10).

Рис. 3.10. Приклад відповіді у форматі *JSON*

3.2.5. Зберігання і обробка даних

Існує кілька інструментів для роботи з базами даних в *iOS*-додатках, вони розрізняються завданнями, для яких призначені. Завдання, яке визначає додаток дипломної роботи, можуть бути вирішені декількома підходами до обробки інформації, але в зберіганні даних вимагають одного – використання реляційної бази даних *SQLite* як сховище даних.

Наступним кроком стає вибір однієї з реляційних баз даних для вбудовування в додаток. Вибір в сторону бази даних *SQLite* обумовлений її портативністю і можливістю взаємодії за допомогою викликів функцій бібліотеки *SQLite*. Цими характеристиками *SQLite* відрізняється від *MySQL* і *PostgreSQL*, які використовують

парадигму «клієнт-сервер». Робота без зв'язку з сервером дозволяє скоротити час відгуку бази даних на запити і тим самим надати можливість швидкої обробки великих обсягів даних. За інформацією офіційного сайту *SQLite* є одним з найбільш широко поширених рішень зберігання даних на стороні клієнта [6].

Обробка даних в *iOS*-додатка може бути здійснена:

– за допомогою *SQL*-запитів;

– інструментами фреймворка *Core Data*.

Щоб зробити правильний вибір, розглянемо завдання, на вирішенні яких спеціалізується кожен з представлених підходів.

Робота з даними за допомогою *SQL*-запитів використовується при роботі з великими обсягами даних і дозволяє здійснювати вибірки з них швидше, а збереження – ефективніше. Це відбувається тому, що процес безпосереднього формування *SQL*-запитів надає можливість виробляти оптимізацію запитів, що, в свою чергу, суттєво позначається на швидкості зворотного зв'язку бази даних і, відповідно, всієї системи в цілому. Крім того, метод роботи з даними за допомогою *SQL*-запитів дозволяє зробити індексування таблиць бази даних, що в значній мірі прискорює процес пошуку даних в базі при великій кількості записів. Таким чином, взаємодія з інформацією бази даних за допомогою *SQL*-запитів виправдано, перш за все, коли інформація є статичними даними великого обсягу, які не будуть змінені в процесі роботи програми. Таким чином, взаємодія являє собою читання і пошук необхідної інформації.

Фреймворк *Core Data* надає розробнику інструменти для спрощеного взаємодії з базою даних, які дозволяють зменшити кількість коду, необхідного для реалізації роботи з даними, в кілька разів. Крім цього, *Core Data* забезпечує розробника інфраструктурою для управління змінами, вилучення, збереження і видалення інформації з бази даних. Звідси випливає, що використання інструментів фреймворка *Core Data* для роботи з базою даних має сенс при обробці динамічних даних, які мають властивість змінюватися в процесі роботи додатка, що призводить до необхідності вносити зміни – додавати, редагувати, видаляти значення полів в таблицях бази даних [10].

База даних, вбудована в *iOS*-додаток дипломного проекту, містить в собі інформацію про улюблені визначні місця та минулі подорожі. Робота з базою даних має на увазі сценарій додавання, зміни або видалення інформації з програми. Таким чином, можемо сказати, що *Core Data* задовольняє вимоги додатка.

Тому необхідно детальніше познайомитися з функціоналом *Core Data.*

Головний клас *Core Data* – *NSPersistenceContainer*, що являє собою повноцінний *Core Data Stack*, який просто створити і використовувати. Клас *NSPersistenceContainer* має простий *API* (рис. 3.11).

```
public var viewContext:NSManagedObjectContext {get}
public func newBackgroundContext() -> NSManagedObjectContext
public func performBackgroundTask
            (_ block: @escaping(NSManagedObjectContext) -> Void)
```

Рис. 3.11. *API* фреймворку *Core Data*

Як видно з назви, управління «видимими» об'єктами *Core Data* здійснюється *View Controllers* на *main queue* з використанням *viewContext*. Також легко працювати з об'єктами *Core Data* в фоновому (*background*) режимі за допомогою контексту, що повертається методом *newbackgroundContext()*. Але ще краще, простіше і оптимальніше виконувати різні операції з *Core Data* в фоновому (*background*) режимі за допомогою методу *performBackgroundTask(_:)*, який сам надає розробникам *NSManagedObjectContext* для виконання цих операцій (рис. 3.12).

```
container.performBackgroundTask { ( NSManagedObjectContext ) in
    code
}
```

Рис. 3.12. Робота з *Core Data* у фоновому режимі

Всі контексти є незалежними, тут немає відносин підпорядкування *parent* / *child*. Для отримання змін з іншого контексту потрібно просто встановити прапор *automaticallyMergesChangesFromParent* в *true* для контексту (рис. 3.13). Незважаючи на те, що в імені прапора присутнє слово «*FromParent*», він правильно працює з рівноправними контекстами.

З *iOS* 10 *Core Data* дозволяє виконувати одночасно безліч «читань» та один «запис» даних без блокування сховища (*SQLlite*) і контекстів. У попередніх версіях

iOS, коли якийсь контекст читався з сховища, всі інші контексти повинні були чекати, щоб виконати свої «читання» або «записи». Це відбувалося через те, що за логікою побудови *Core Data* контакт з сховищем проходив через екземпляр класу *NSPersistentStoreCoordinator*, який був своєрідним «шийкою пляшки», через яке в певний момент часу міг пройти тільки один контекст, а це означало, що незважаючи на всі прикладені зусилля не можна було повністю викорінити уповільнення *UI* на *main queue* в той час, коли відбувається імпорт даних і запис даних в базу в фоновому (*background*) режимі.

Рис. 3.13. Схема відносин *Core Data* з контекстами

З *iOS* 10 безліч контекстів може читати дані з сховища одночасно, так як *Apple* перемістила завдання блокування контекстів з *NSPersistentStoreCoordinator* безпосередньо на базу даних *SQLite*, яка може мати безліч «читаючих» каналів і один який «пише». Це призвело до більш чуйного *UI*, коли *viewContext* здатний вибирати дані в той час, як *backgroundContext* пише дані на диск [11].

З *iOS* 10 розробник отримує ще більший контроль над контекстом за допомогою супер можливості *Query Generation*, яка дозволяє нам зробити миттєвий «знімок» з нашої бази даних і приєднати її до існуючого контексту за допомогою одного простого рядка коду:

try! container.viewContext.setQueryGenerationFrom(.current)

З цього моменту ніяких зміни в інших контекстах не будуть проникати в *viewContext*. Для того, щоб оновити *viewContext* після того, як зробили необхідні зміни, вам потрібно знову викликати *setQueryGenerationFrom(.current)*. Контекст також автоматично оновлюється до самої останньої версії, коли його зберігають і заново встановлюють або коли вставляються зміни з іншого контексту. Для використання *Query Generations* в *Core Data* необхідно використовувати сховище у вигляді *SQLite*.

Така організація *Core Data* з *iOS* 10 дозволяє позбутися від двох істотних недоліків, які були присутні в попередніх версіях:

1. *NSPersistentStoreCoordinator* керував доступом до сховища і контексти були змушені чекати один одного, в результаті *main queue* (тобто *UI* додатка) блокувався.

2. Часто траплялося аварійне завершення додаток через помилки «*CoreData could not fulfill a fault for...*» (*Core Data* не може відновити об'єкт ...). Це відбувається через те, що об'єкт міг бути вилучений в іншому контексті та доступ до нього здійснено раніше, ніж прийшло повідомлення про його зміну.

3.3. Розробка архітектури додатка

Основним етапом в реалізації мобільного додатка є розробка архітектури. Архітектура, побудована належним чином, здатна полегшити майбутнє розширення і розвиток програми.

3.3.1. Мова розробки *Swift*

Для створення додатків під системи *Apple* в основному використовується дві мови – *Swift* та *Objective-C*.

Objective-C є розширенням мови *C*, в який були додані нові можливості для об'єктно-орієнтованого підходу програмування. Мова програмування *Objective-C*, розроблена на початку 1980-х років, була основною мовою, що використовувалася компанією *NeXT* для операційної системи *NeXTSTEP*, від якої пішли *MacOS* і *iOS*. Мова використовує об'єктну модель *Smalltalk*. Повністю сумісна з мовою програмування *C*. Компанія *Apple* довгий час використовувала *Objective-C* як основну мову програмування для розробки своїх продуктів [12].

Swift є універсальною, мовою програмування, розробленою *Apple* і спільнотою з відкритим вихідним кодом. Мова була представлена в 2014 році. *Swift* була розроблена в якості заміни *Objective-C* для *Apple*, тому що остання практично не змінилася з початку 1980-х років, і в ній були відсутні сучасні мовні особливості. *Swift* працює з фреймворками *Apple Cocoa* і *Cocoa Touch*, та ключовим аспектом дизайну *Swift* була здатність взаємодіяти з величезним обсягом існуючого коду *Objective-C*, розробленого для продуктів *Apple* за попередні десятиліття. Вона побудована на основі компілятора *LLVM* з відкритим вихідним кодом і включена в *Xcode* з версії 6. На платформах *Apple* вона використовує бібліотеку *Objective-C*, яка дозволяє запускати код *C, Objective-C, C++* і *Swift* в рамках однієї програми.

Swift має функції, що усувають деякі поширені помилки програмування, такі як розіменування нульового покажчика. *Swift* підтримує концепцію розширюваності протоколу. Систему розширюваності, яка може застосовуватися до типів, структур і

класів, яку *Apple* просуває як реальну зміну парадигм програмування, яку вони називають «протокольно-орієнтованим програмуванням».

Swift був представлений на Всесвітній конференції розробників *Apple* (*WWDC*) в 2014 році. Він пройшов оновлення до версії 1.2 протягом 2014 року і більш серйозне оновлення до *Swift* 2 на *WWDC* 2015. У версії 3.0 синтаксис *Swift* зазнав значної еволюції, і основна команда зробила наголос на стабільність вихідного коду в більш пізніх версіях. У першому кварталі 2018 року *Swift* перевершив *Objective-C* по виміряної популярності.

Таким чином, було прийнято рішення використовувати мову програмування *Swift* при написанні додатка.

3.3.2. Середовище розробки *Xcode*

Xcode – це пакет засобів розробки, який використовується для створення, тестування, налагодження і налаштування *iOS*-додатків (рис. 3.14). Середовище розробки *Xcode*, яке є оболонкою для інструментів створення додатків, інструментів для налагодження та *iOS*-симулятор. Для написання і налагодження коду використовується додаток *Xcode*, тестування відбувається шляхом запуску додатків на *iOS*-симуляторі або на безпосередньо підключеному пристрої під управлінням операційної системи *iOS*. Для вимірювання продуктивності додатків використовуються інструменти, які можуть бути запущені з *Xcode*-додатків [13].

Рис. 3.14. Інтерфейс *Xcode*

З самого початку на що можна звернути увагу користуючись *Xcode*, це одне-єдине вікно, що об’єднує в собі всі завдання по розробці. Робоча область володіє декількома унікальними елементами призначеного для користувача інтерфейсу, що роблять інструментарій простим в роботі при виконанні безлічі різних завдань. Навіть робота над декількома проектами не ускладнить робочої області. Редактор завжди знаходиться в центрі і поверх інших вікон.

Зліва знаходиться колекція навігаторів, що включає в себе список файлів проекту, пошуковий інтерфейс, обробку помилок, налагоджувальні дані, активні і неактивні контрольні точки, а також колекція логів проекту (рис. 3.15). Уніфікований навігатор призначеного для користувача інтерфейсу має можливість «живої» фільтрації контенту і результатів пошуку.

Чим простіше і зрозуміліше реалізована система звернення до різних елементів проекту, тим більше уваги розробник зможе приділити своє головне завдання – створення корисного додатка. Таким чином *Apple* не тільки спрощує завдання програміста, але і істотно заощаджує його час, тим самим наближаючи довгоочікувану мить виходу нової програми.

Вгорі кожної панелі редактора знаходиться адресний рядок, що показує відносне розташування поточного файлу (рис. 3.16). Клацанням по будь-якій адресі можна переходити на будь-який інший файл того ж рівня. Це називається «*Jump Bar*» («Панель переходу») і вона дозволяє ефективно показувати на екрані програмний код шляхом простого «переходу» від файлу до файлу.

Рис. 3.15. Навігація в *Xcode*

Навігація по файлах проекту вважається одною з дій, яка найбільше відволікає увагу розробника. Замість того, щоб утримувати в свідомості більш важливі речі, безпосередньо відносяться до роботи над програмним продуктом, він змушений забивати собі голову плутаниною шляхів до різних файлів. Це просто чудово, що розробники *Apple* серйозно задумалися над цією проблемою і запропонували її гідне рішення.

Рис. 3.16. Адресний рядок в *Xcode*

Також у *Xcode IDE* повністю інтегрований *Interface Builder* («конструктор інтерфейсу») і не є окремим додатком.

При виборі файлу інтерфейсу (*.nib* / *.xib*) проекту, він відкриється безпосередньо в *Xcode IB*-редакторі. При відкритті праворуч знаходиться область утиліт
(«*Utility area*»), в якій буде відображений повний набір інспекторів інтерфейсу, а також бібліотека елементів управління і об’єктів користувацького інтерфейсу (рис. 3.17).

Найцікавіша можливість перенесення об’єкта безпосередньо з призначеного для користувача інтерфейсу в програмний код. У *Xcode* перенесення об’єктів інтерфейсу в програмний код здійснюється одним рухом (рис. 3.18). Якщо готового коду для встановлення взаємозв’язку немає, то *Xcode* може створити його самостійно. Просто необхідно перетягнути його на порожній простір файлу вихідного коду і *Xcode* згенерує код без безпосереднього втручання.

У сучасному візуальному програмуванні той чи інший елемент керування або об’єкт є наочно представленим фрагментом програмного коду. Те, над чим програмісти минулого працювали наосліп, сьогодні перед очима розробника. Але в багатьох випадках код елемента управління потребує редагування. Розробнику буде зручно спочатку вставити стандартний код об’єкта, а потім в редакторі коду внести необхідні зміни.

Xcode базується на новому поколінні компіляторів, представленому *Apple LLVM*. Компілятор базується на вихідному матеріалі проекту *LLVM.org,* який руками інженерів *Apple* перетворився в компілятор *Apple LLVM,* створений спеціально для *iPhone, iPad* і *Mac*.

Рис. 3.17. Інтегрований конструктор інтерфейсів у *Xcode*

Apple LLVM надзвичайно швидкий. Він компілює код вдвічі швидше, ніж *GCC* і це тягне за собою прискорення запуску готових додатків. Компілятор побудований на фундаменті набору оптимізованих, легко розширюваних і легко оптимізуються бібліотек, спроектованих відповідно до сучасної архітектурою чіпів. У *Xcode Apple LLVM* володіє чудовою підтримкою мов програмування *C, Objective-C, C++, Swift* на рівнях від обробки парсеру до оптимізації коду.

Підкреслення синтаксису, фіналізація програмного коду та інші функції індексації управляються парсером в *LLVM*. Мови *C, C++, Objective-C і Swift* коректно сприймаються під час редагування так само, як і в процесі розробки програми. Якщо символ відомий компілятору, то і *Xcode IDE* сприймає його правильно.

Рис. 3.18. Перенесення об’єкта з графічного інтерфейсу в код

Движок *Apple LLVM* постійно працює у фоновому режимі і розпізнає написаний код. У редакторі функція *Live Issues* використовує це розпізнавання для того, щоб повідомляти про помилки введеного програмного коду. Точно так, як текстовий редактор підкреслює помилки правопису, *Xcode* виділяє помилки, допущені програмістом прямо в процесі роботи без попередньої компіляції програми.

Жоден навіть найуважніший програміст не застрахований від помилок і друкарських помилок. Вкрай бажано, щоб вони виявлялися на стадії написання програми, а не в процесі компіляції. Адже куди простіше виправити помилку і піти далі, ніж потім розбиратися в цілій купі помилок, багато з яких просто незначні. Такий підхід відволікає увагу розробника від дійсно значних недоліків, допущених в процесі роботи над додатком.

При надходженні повідомлення про помилки, розумне *IDE* може їх автоматично виправити. У багатьох випадках *Xcode* не обмежується повідомленням про помилку, а одразу пропонує спосіб, яким її можна виправити. Для цього треба клацнути на помилку, щоб побачити відповідні варіанти вирішення проблеми, в тому числі коригування та виправлення невірно введених символів і додавання пропущених розділових знаків.

Fix-it безцінний помічник у процесі введення, а завдяки функції *Analyze* забезпечується скрупульозне тестування. Статичний аналізатор *Xcode* перевіряє тисячі шляхів на наявність ділянок коду, які викликають ту чи іншу проблему. Код зверне на себе увагу аналізатора навіть якщо він коректний, але при цьому поводиться несподіваним чином, викликаючи, наприклад, помилку звернення до пам’яті або ж містить невірно побудований цикл. Пара незамінних помічників, *Fix-it* і *Xcode Analyze*, виявлять помилку раніше користувачів програми.

Як вже було сказано, значна частина помилок це просто наслідок неуважного введення. Такі помилки і непроставлені знаки пунктуації куди простіше надати виправити автоматиці. Якщо мозок людини краще справляється з творчою роботою, то увага до найдрібніших деталей властиво комп'ютерній техніці куди більшою мірою, ніж людині, яка, втомлюючись, перестає концентрувати свою увагу на дрібницях (рис. 3.19).

Рис. 3.19. *Xcode* пропонує автоматичне виправлення помилок

Редактор версій («*Version Editor*») в *Xcode* дозволяє легко і просто одночасно бачити будь-які дві версії вашого вихідного коду. Дуже важливо, що редактор версій забезпечує роботу над кодом елементів управління в *IDE* оскільки зіставлення можливо також з різними версіями коду.

В процесі роботи програміст багаторазово вносить зміни в код програми. У підсумку виходить кілька версій одного і того ж фрагмента. Згодом розробник може усвідомити, що те чи інше рішення, яке він застосував раніше, було більш вдалим, ніж використане згодом. (рис. 3.20) В такому випадку завжди буде зручно повернутися до попередньої версії, щоб перемістити з неї ділянку коду .

3.3.3. Концепція *Model-View-Controller*

Ключове місце в створенні програмного продукту взагалі і мобільних додатків для операційної системи *iOS* зокрема грає дотримання якогось патерну проектування. Такі шаблони спрощують повторне використання вдалих архітектурних і проектних рішень. Крім того, за допомогою патернів можна спростити процес підтримання розроблених систем і поліпшити якість документації, дозволяючи явно описати функції тих чи інших класів і об’єктів, а також їх взаємодію між собою.

При проектуванні мобільних додатків для операційної системи *iOS,* ґрунтуючись на особливостях і інструментах розробки, слідують патерну проектування *MVC*.

Рис. 3.20. Контроль версій в *Xcode*

3.3.4. Поняття *Model-View-Controller*

MVC (англ. «*Model-View-Controller*» або модель-представлення-контролер) являє собою шаблон проектування, згідно з яким модель даних, призначений для користувача інтерфейс і механізм взаємодії з користувачем розділені на три окремих компоненти, що спеціалізуються на своїх завданнях. Таким чином, зміна одного з компонентів не робить істотного впливу на інші, і додатки, побудовані відповідно до концепції *MVC,* розширюються легше, ніж інші програми. Крім того, дотримання

концепції *MVC* дозволяє повторно використовувати ті чи інші елементи системи [14].

Шаблон *MVC* визначає не тільки завдання, які виконує кожен з цих об’єктів, а й способи взаємодії об’єктів між собою. Кожен з трьох типів об’єктів відділений від інших абстрактними межами і взаємодіє з об’єктами інших типів через ці межі.

3.3.5. Класична взаємодія компонентів *MVC*

Незважаючи на те, що згідно з концепцією *MVC* модель, представлення і контроллер відокремлені один від одного, їх взаємодія між собою має велике значення для інтерактиву додатка. Ця взаємодія може здійснюватися кількома способами [15].

У найпростішому випадку для моделі не передбачається будь-якої активної участі у взаємодії компонентів *MVC*. Хорошим прикладом такого випадку служить простий текстовий *WYSIWYG* редактор. Основною особливістю даного редактора є те, що ми завжди повинні бачити текст рівно так само, як якщо б він був написаний на папері. Таким чином, об’єкти представлення повинні бути інформовані про кожну зміну тексту для того, щоб текст міг бути оперативно відображений в текстовому полі. Проте, модель (для простоти будемо вважати об’єктом моделі екземпляр класу *String*) не бере на себе відповідальність за повідомлення змін об’єкту представлення, оскільки ці зміни відбуваються тільки за запитами користувача. Так як запити користувача обробляє контроллер, він і несе відповідальність за повідомлення об’єкта уявлення про будь-які зміни. На цьому етапі виникає два варіанти взаємодії або контроллер просто повідомляє представлення про те, що щось зазнало змін і об’єкт представлення запитує поточний стан рядка у моделі, або контроллер вказує представленню які зміни відбулися. У будь-якому випадку, модель є абсолютно пасивним сховищем строкових даних, які обробляються контроллером і представленням. Модель додає, видаляє або замінює підрядки на вимогу контроллера і видає відповідні підрядки за запитом від представлення. Таким чином, модель не має детального знання про існування об’єктів представлення або контроллера і їх участі в *MVC*. Проте, ця ізоляція не є показником простоти моделі, але говорить про те, що модель змінюється тільки з волі одного з інших компонентів тріади.

Однак не всі моделі можуть бути пасивними. Припустимо, що рядок у наведеному вище прикладі змінюється в результаті запитів компонентів, відмінних від об’єктів представлення і контроллера. Наприклад, якщо інформація була отримана з бази даних. У даному випадку об’єкт, який залежить від стану моделі – власне, об’єкт представлення – повинен бути повідомлений про те, що в моделі відбулися зміни. Так як в даному випадку виключно модель може відстежувати, які

зміни з нею відбуваються, модель повинна мати якийсь канал взаємодії з представленням. Згідно з концепцією *MVC*, модель не може взаємодіяти безпосередньо з об'єктами представлення, таким чином, вся взаємодія здійснюється за допомогою контроллера.

А саме модель сповіщає всі залежні від неї контроллери про те, що в ній відбулися зміни. Одним з варіантів оповіщення про зміну може бути надсилання безпосередньо отриманих даних в контроллер.

На відміну від моделей, які можуть бути пов'язані з декількома об'єктами представлення за допомогою контроллерів, кожен з контроллерів пов'язаний з унікальним представлення і навпаки. Таким чином, контроллери і представлення можуть звертатися один до одного безпосередньо, уникаючи будь-яких посередників.

3.3.6. Концепція *MVC* в проектуванні *iOS*-додатків

Модель (англ. «*Model*»), згідно з концепцією *MVC*, містить дані додатка і визначає правила, принципи і залежність поведінки цих даних (рис. 3.21). Наприклад, об'єкт моделі може являти собою персонажа гри або контакт в адресній книзі. Об'єкт моделі може мати зв'язку «один до одного» або «один до багатьох» з іншими об'єктами моделі і бути пов'язаний до одного або більше об'єктів представлення [16]. Велика частина даних програми, яка зберігається в файлах або базі даних, повинна знаходитися в об'єктах моделі після того, як дані були завантажені в додаток. Оскільки об'єкти моделі являють собою свого роду знання і досвід, пов'язані з певною проблемною областю, вони можуть бути використані повторно в інших подібних проблемних областях. Об'єкти моделі не повинні мати явний зв'язок з контроллерами, які відображають дані додатки і дозволяють користувачам редагувати ці дані. Об'єкти моделі не повинні мати відношення до призначеного для користувача інтерфейсу і питань відображення.

Взаємодія: дії користувачів в представленні, що створюють або змінюють дані додатка, передаються через контроллер і призводять до створення або зміни об'єкта моделі. Коли об'єкт моделі буде змінено (наприклад, по мережі будуть отримані нові дані), він надсилає повідомлення контроллеру, який оновлює відповідний об'єкт представлення.

Представлення (англ. «*View*») містить елементи програми, які може бачити користувач, а саме елементи призначеного для користувача інтерфейсу. Основна мета об'єктів представлення – відображення даних об'єктів моделі програми та надання інструментів для редагування цих даних.

Взаємодія: об'єкти представлення дізнаються про зміни об'єктів моделі через контроллерів програми та відповідно до них оновлюють дані, які бачить користувач. І навпаки, ініційовані користувачем зміни пов'язуються з об'єктами моделі за допомогою контроллера. Наприклад, текст, який Ви самі ввели в текстове поле, через контролер надходить в об'єкт моделі.

Рис. 3.21. Концепція *MVC*

Контроллер (англ. «*Controller*») виступає в якості посередника між одним або декількома об'єктами представлення і одним або декількома об'єктами моделі. Контроллер є каналом, через який об'єкти представлення дізнаються про зміни в об'єктах моделі і навпаки.

Взаємодія: контроллер інтерпретує дії користувача, зроблені в об'єктах представлення, і передає нові або модифіковані дані в об'єкт моделі. Коли відбуваються зміни в об'єктах моделі, контроллер передає нові дані в об'єкт представлення для подальшого відображення.

Важливо відзначити, що як представлення, так і контролер залежать від моделі. Однак модель не залежить ні від того, ні від іншого.

3.3.7. Архітектура додатка

З урахуванням всіх розглянутих рішень процес проектування архітектури додатка завершився результатом представленим на рис. 3.22, 3.23. На рисунках вказані об'єкти представлення, з кожним з них асоційований один контроллер і об'єкти моделі.

3.3.8. Тестування додатка

Важливим етапом створення програмного продукту взагалі і мобільного застосування зокрема є тестування, яке значно знижує ризик виникнення помилок після випуску продукту в експлуатацію [18].

Тестування мобільних додатків для операційної системи *iOS* може проводитися двома способами:

– з використанням *iOS Simulator* – програми для запуску та налагодження *iOS*-додатків на комп'ютері через симулятор *iPhone* або *iPad*;

– на пристроях під управлінням операційної системи *iOS*.

Тестування програми на *iOS*-пристрої передбачає наявність сертифіката розробника в програмі *iOS Developer Program*, пристрої під управлінням операційної системи *iOS*, зареєстрованого в цій програмі і сертифіката *Provisioning Profile*, де вказуються ідентифікатори тестованої програми та пристрої, на якому буде проходити тестування.

Рис. 3.22. Архітектура *MVC* для екранів, які відображають обрані місця та історію попередніх подорожей

Рис. 3.23. Архітектура *MVC* для першого екрану та екрану, який відображає деталі про місце

Значущим компонентом в тестуванні додатків є програма *Instruments*, яка дозволяє відстежувати продуктивність і витрати додатки для вивчення і аналізу його поведінки. Крім того, ця програма може бути використана для запису дій користувача і їх відтворення для здійснення збору даних про програму.

Instruments містять наступні можливості:

– вивчення поведінки одного або декількох процесів;

– отримання даних про витоки пам'яті і витраті заряду акумулятора;

– виконання загального пошуку несправностей на рівні системи.

Використання *Instruments* в процесі тестування програми дозволяє знаходити вади в продуктивності програми та оперативно їх коригувати.

Тестування мобільного додатка проводилося на *iOS*-пристрої *iPhone* 12 під операційною системою *iOS* 14.2, а також на різних версіях *iOS Simulator* – *iPhone* 8, *iPhone X*, *iPhone* 12, *iPhone* 12 *Pro Max* під керуванням *iOS* 12, *iOS* 13, *iOS* 14.

3.4. Висновки до розділу

У третьому розділі було:

– розглянуто процес розробки дизайну мобільного додатка;

– створено *user flow* мобільного застосунку;

– розглянуто програмне забезпечення для створення інтерфейсів мобільних додатків *Adobe XD*;

– розроблено *UI / UX* макет;

– розглянуто та проаналізовано інтерфейс середовища розробки *Xcode*;

– розглянуто зовнішні *API*, які використовувалися при розробці;

– розглянуто концепцію архітектури *MVC*;

– розроблено програмний код та проведено тестування мобільного застосунку визначення оптимального шляху по визначним місцям.

ВИСНОВКИ

Метою дипломного проекту було розробити мобільний додаток визначення оптимального шляху по визначним місцям.

В першу чергу була розглянута проблема відсутності подібного мобільного застосунку на ринку. Було проведено аналіз існуючих мобільних додатків з подібним функціоналом, здійснено їх порівняльну характеристику, виявлено переваги та недоліки.

Під час виконання дипломного проекту були розв’язані наступні задачі:

– розглянуто існуючі мобільні додатки із схожим функціоналом і проаналізовані їхні переваги та недоліки;

– проаналізовано різні особливості розробки мобільних додатків під операційні системи *iOS* і *Android*;

– проведено ознайомлення з процесом розробки дизайну мобільного додатка;

– створено *user flow* мобільного застосунку;

– розроблено *UI / UX* макет у програмі для створення інтерфейсів мобільних додатків *Adobe XD*;

– розглянуто зовнішні *API*, які використовувалися при розробці;

– розглянуто концепцію архітектури *MVC*;

– розроблено програмний код та проведено тестування мобільного застосунку визначення оптимального шляху по визначним місцям.

У дипломному проекті було проаналізовано вплив смартфонів на людей та стан мобільної індустрії в 2020 році. Сьогодні смартфони стали невід’ємною

частиною повсякденного життя людей. А також тенденція до зростання наявності мобільних пристроїв значною мірою супроводжує думку про те, що створений мобільний додаток буде доступним широкому користувачеві.

Також було проведено ознайомлення із процесом нативної розробки мобільного додатка під операційну систему *iOS*. Мобільним додатком можна назвати комп'ютерну програму, створену спеціально для використання в мобільному телефоні, смартфоні або комунікаторі, яка призначена для виконання того чи іншого завдання.

В результаті роботи був реалізований мобільний додаток визначення оптимального шляху по визначним місцям. Реалізований застосунок можна використовувати для вирішення особистих задач при подорожуванні, так і в комерційних цілях.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Mobile operating systems' market share worldwide from January 2012 to October 2020.* – Електрон. дан. – Режим доступу: *https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/*

2. *Digital in 2020.* – Електрон. дан. – Режим доступу*:* *https://wearesocial.com/digital-2020*

3. Офіційний сайт компанії *Apple*. – Електрон. дан. – Режим доступу: *https://www.apple.com/ru/ios/what-is/*

4. *App Store*. – Електрон. дан. – Режим доступу: *https:// developer.apple.com/support/appstore/*

5. *iOS Human Interface*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/ios/documentation/userexperience/conceptual/MobileHIG/index.html*

6. Офіційний сайт *SQLite*. – Електрон. дан. – Режим доступу: *http://www.sqlite.org*

7. В. Е. Дементьев. Информационно-вычислительные сети. – Ульяновск: УлГТУ, 2011. — 141с.

8. *R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1. – The Internet Society*, 1999. – 114 с.

9. *R.T.Fielding Architectural Styles and the Design of Network-based Software Architectures. – Hyderabad: University College of Science*, 2000. – 180 с.

10. *iOS Technology Overview*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html*

11. *Cocoa Core Competencies*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html*

12. *Steve Burbeck Applications Programming in Smalltalk-80TM: How to use Model- View-Controller (MVC) – ParcPlace Systems,*1992. – 22 с.

13. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес, Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.

14. *Google Maps SDK for iOS.* – Електрон. дан. – Режим доступу: *https://developers.google.com/maps/documentation/ios/?hl=ru*

15. *Map Kit Framework Reference*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/_index.html*

16. *Cocoa Application Competencies for iOS*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/ios/documentation/general/conceptual/Devpedia-CocoaApp/Storyboard.html*

17. *Instruments User Guide*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/library/mac/documentation/developertools/conceptual/instrumentsuserguide/Introduction/Introduction.html*

18. *App Store Resource Center*. – Електрон. дан. – Режим доступу: *https://developer.apple.com/appstore/index.html*

19. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

20. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення»